On tensor orderings for HiCOO (and other data structures)

Bora Uçar

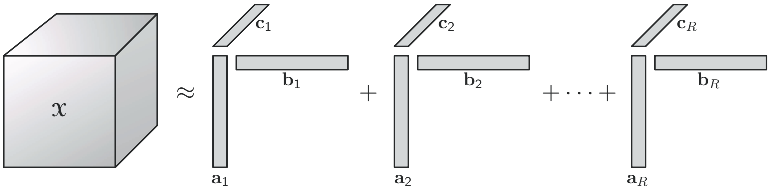
CNRS & ENS Lyon, France

25 February–1 March, 2019 @SIAM CSE 2019, Spokane

Joint work with

Jiajia Li and K. Barker	PNNL
U. V. Çatalyürek, J. Sun, and R. Vuduc	GaTech

Motivation



Given a tensor $\boldsymbol{\mathcal{X}}$, and a number R, Candecomp/Parafac (CP) decomposition approximates $\boldsymbol{\mathcal{X}}$ as a sum of R rank-1 tensors.

Many applications: data analysis & mining for health care, natural language processing, machine learning, social network analytics,...

We will look at the method CP-ALS for computing CP decompositions.

CP-ALS

Algorithm 1: CP-ALS for 3D

Input : $\mathcal{X}$: $I \times J \times K$ tensor
R: The rank

Output: CP decomposition
$[\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$

Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$

repeat

$\mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{B}^T\mathbf{B} * \mathbf{C}^T\mathbf{C})^\dagger$

Normalize columns of $\mathbf{A}$

$\mathbf{B} \leftarrow \mathbf{X}_{(2)} \dots$

Normalize columns of $\mathbf{B}$

$\mathbf{C} \leftarrow \mathbf{X}_{(3)} \dots$

Normalize columns of $\mathbf{C}$ and store the norms as $\boldsymbol{\lambda}$

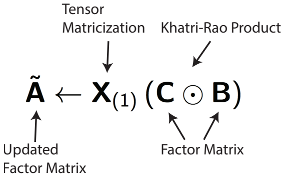
until ...

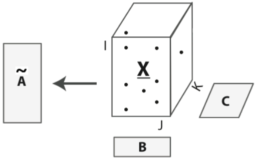
- $\mathbf{X}_{(1)}$ sparse matrix, $I \times J \cdot K$
- $\mathbf{A}$ is $I \times R$; $\mathbf{B}$ is $J \times R$; $\mathbf{C}$ is $K \times R$.
- $(\mathbf{B}^T\mathbf{B} * \mathbf{C}^T\mathbf{C})^\dagger$ is $R \times R$, Hadamard product, pseudo-inverse.
- $\mathbf{C} \odot \mathbf{B}$ Khatri-Rao product, $J \cdot K \times R$.
- $\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$ is Matricized Tensor-Times Khatri-Rao Product (MTTKRP)

MTTKRP is the computational core.

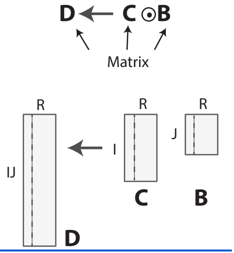
MTTKRP Operation

Matricized tensor times Khatri-Rao product (MTTRKP):





- Khatri-Rao Product



foreach $x_{i,j,k} \in \boldsymbol{\mathcal{X}}$ **do**

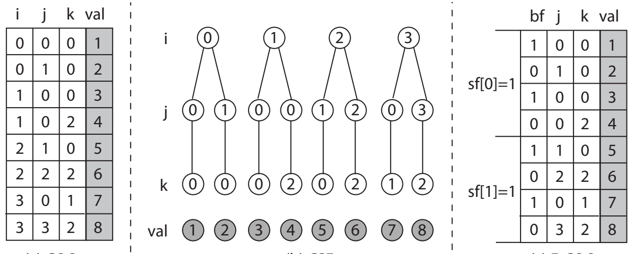
$\quad \mathbf{M}_A(i,:) \leftarrow \mathbf{M}_A(i,:) + x_{i,j,k}[\mathbf{B}(j,:) * \mathbf{C}(k,:)]$

end

Perform MTTKRP as efficiently as possible.

Some existing sparse tensor formats

- COO: coordinate
- CSF: Compressed sparse fiber (extension of CSR) [Smith et al.'15]
- F-COO: Flagged COO [Liu et al.'17]



(a) COO (b) CSF (c) F-COO

Mode-Generic

Mode-Specific
prefer different representations for different modes.

Sparse tensor storage challenges

A compact, space-efficient representation

efficient computations on all modes, for many typical computations, mode-genericity(-obliviousness).

HiCOO is a more recent storage format

- retains mode-genericity of COO.
  while being space efficient.
- cache friendly.

Baskaran et al.'12 for the term mode-genericity; Jiajia Li et al.'18 for HiCOO

HiCOO

Store the tensor in units of small sparse blocks (generalized from CSB)

- shorten the bit-length of indices
- compress the block indices

block indices | element indices

32-bit

i	j	k	val
0	0	0	1
0	1	0	2
1	0	0	3
1	0	2	4
2	1	0	5
2	2	2	6
3	0	1	7
3	3	2	8

COO

32-bit | 8-bit

	bptr	bi	bj	bk	ei	ej	ek	val
B1	0	0	0	0	0	0	0	1
					0	1	0	2
					1	0	0	3
B2	3	0	0	1	1	0	0	4
B3	4	1	0	0	0	1	0	5
					1	0	1	7
B4	6	1	1	1	0	0	0	6
					1	1	0	8

HiCOO

i = bi * B + ei

COO indices:
= nnz * 3 * 32

HiCOO indices:
= nnz * 3 * **8** + **nnb** * (3 * 32 + 32)

nnz: #Nonzeros; nnb: #Non-zero blocks

CSB: Compressed sparse blocks, Buluc et al.'09

Our aim

Goal: (Further) Improve the performance of HiCOO by reordering the mode indices.

Why reordering: Number of blocks will reduce if we do it right.

$\Rightarrow$ improve data locality in the tensor and the factor matrices.
COO and CSF will benefit.

No changes in the MTTKRP code.

Our aim: What exactly

HiCOO

- Reduce the number of blocks by closely packing nonzeros
- Yield denser blocks
- Reduce storage

The performance gain: increased block density, reduced num blocks, and improved cache use.

COO and CSF

- No difference in COO storage
- No difference in CSF's tree structure (the order of children changes)

The performance gain: from the improved data locality.

How we go about it?

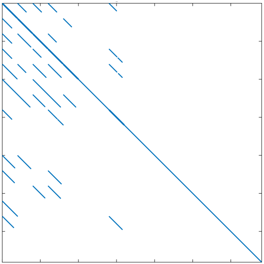
We propose two heuristics

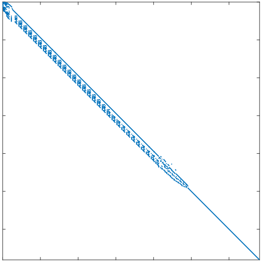
$\Rightarrow$ arrange the nonzeros close to each other, in all modes.

Reasoning with matrices

Reorder the rows and columns so that nonzeros are around the diagonal.

- Nonzeros in a row or column will be close to each other.
- Any (regular) blocking will have nonzero blocks around the diagonal.



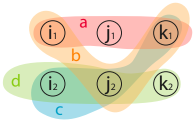


First heuristic: BFS-MCS

- A breadth-first-search-like heuristic, using maximum cardinality search.
- Build a hypergraph: a set of vertices per mode, & every nonzero is a hyperedge
- Order each mode independently.

i	j	k	val
i_1	j_1	k_1	a
i_1	j_2	k_1	b
i_2	j_2	k_1	c
i_2	j_2	k_2	d

(a) Tensor



(b) Hypergraph

Second heuristic: Lexi-Order

An extension of doubly lexical ordering of matrices to tensors.

Doubly lexicographic ordering of matrices:

The d_is are read in the shown order, to form a string of $\{0, 1\}$s of length $m \cdot n$.

We want the smallest string in the dictionary order (1s are before 0s).

Every real-valued matrix has a doubly lexicographic ordering.

Not unique.

$$\begin{bmatrix} d_1 & d_3 & d_6 & \\ d_2 & d_5 & & \cdot \\ d_4 & & \cdot & \\ & \cdot & & d_{m\cdot n} \end{bmatrix}$$

Lubiw'87

Lexi-Order: Matrix case

- Known methods are "direct". Run time of $\mathcal{O}(\text{nnz} \log(I+J)+J)$ and $\mathcal{O}(\text{nnz}+I+J)$ space, for an $I \times J$ matrix with nnz nonzeros.
  Too high for our purposes.
- The data structures are too complex ("rather elaborate").
  Hard to achieve efficient generalizations for tensors.

We propose `matLexiOrder`

- An iterative algorithm; an iteration sorts either rows or columns.
- It obtains a solution; simpler and probably more efficient.

- We do not need an exact lexico-ordering; a close-by one will likely suffice (to improve the MTTKRP performance).
- Assume an ordering of the rows, sort the columns lexically in linear time with an order preserving variant of Partition Refinement.

Lubiw'87; Paige and Tarjan'87

Lexi-Order: Proposed variant for tensors

Order one mode by assuming the others are ordered.

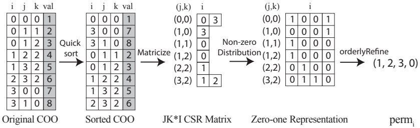
Algorithm 2 LEXI-ORDER for a given mode.

Input: An Nth-order sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, mode n;
Output: Permutation perm_n;

```
                                          ▷ Sort all nonzeros along with all but mode n.
quickSort(𝔛, coordCmp);
                                          ▷ Matricize 𝔛 to X_(n).
r = compose (inds ([−n]), 1);
for m = 1, . . . , M do
    c = inds(n, m);                       ▷ Column index of X_(n)
    if coordCmp(𝔛, m, m − 1) == 1 then
        r = compose (inds ([−n]), m);     ▷ Row index of X_(n)
    X_(n)(r, c) = val(m);
                                          ▷ Use a variation of partition refinement in [28]
perm_n = orderlyRefine (X_(n));
return perm_n;
                                          ▷ Comparison function for two indices of 𝔛
Function: coordCmp(𝔛, m1, m2)
for n' = 1, . . . , N do
    if n' != n then
        if m1(n') < m2(n') then
            return −1;                    ▷ Entry m1 < entry m2
        if m1(n') > m2(n') then
            return 1;                     ▷ Entry m1 > entry m2
return 0;                                 ▷ Entry m1 = entry m2
```

For each dim

- Matricize to have dim as the columns
- Order the columns lexically



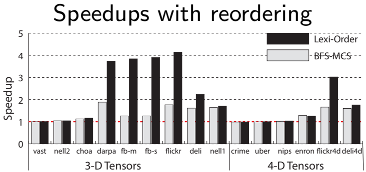
Experiments: Set up

- Linux-based Intel Xeon E5-2698 v3 multicore platform with 32 physical cores distributed on two sockets, each with 2.3 GHz.
- Haswell microarchitecture,
  32 KiB L1 data cache and 128 GiB memory.
- C using OpenMP parallelization; compiler icc 18.0.1.
- sparse tensors from `http://frostt.io/` (Smith, Choi, Li, et al.)

Experiments: Configuration

- Best configurations to obtain the highest MTTKRP performance:
  - the superblock size L and the block size B of HiCOO format,
  - best practices for COO & CSF.
- Five lexi-ordering iterations.
- Approximation rank of $R = 16$.
- The parallel experiments use 32 threads.
- The total execution time of MTTKRPs in all modes.
- Speedup is the ratio over a run on a randomly reordered tensor.
- Run times are averaged over five runs.

HiCOO-MTTKRP sequential



(a) Sequential

LEXI-ORDER: 0.99–4.14× speedup (2.12× on average).

BFS-MCS: 0.99–1.88× speedup (1.34× on average).

---

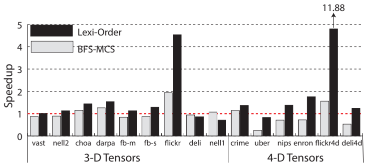
flickr4d is constructed from the same data with flickr, with an extra short mode. LEXI-ORDER obtains 4.14× speedup on flickr while 3.02× speedup on flickr4d.

Similar behavior on deli and deli4d.

Hard to get good data locality on higher-order tensors.

HiCOO-MTTKRP parallel

Speedups with reordering



(b) Parallel

LEXI-ORDER: 0.71–11.88× speedup (2.14× on average).

BFS-MCS: 0.25–1.94× speedup (0.98× on average).

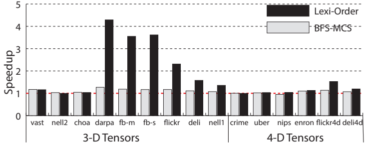
The benefit is generally less than that in the sequential case.

11.88× speedup on flick4d is because of a different superblock size *L*.

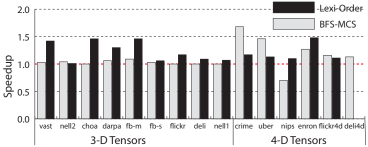
- Need a better thread scheduling in HICOO?—Done recently
- Automatically tuning the parameters of HICOO will be very helpful.

COO-MTTKRP sequential and parallel

Speedups with reordering



(a) Sequential



(b) Parallel

COO-MTTKRP from Parti!, following TensorToolbox

Sequential

LEXI-ORDER: 1.00–4.29× speedup (1.79× on average).

BFS-MCS: 0.95–1.27× speedup (1.10× on average).

Parallel

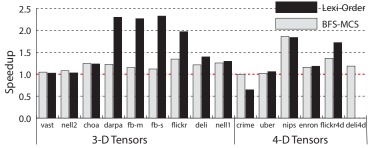
LEXI-ORDER: 1.01–1.48× speedup (1.21× on average).

BFS-MCS: 0.70–1.68× speedup (1.11× on average).

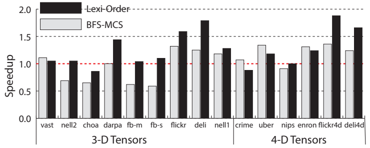
The qualitative effect on performance:
less for COO-MTTKRP than HICOO-MTTKRP.

CSF-MTTKRP sequential and parallel

Speedups with reordering



(a) Sequential



(b) Parallel

CSF-MTTKRP from SPLATT v1.1.1 with all CSF representations.

Sequential

LEXI-ORDER: 0.65–2.33× speedup (1.50× on average).

BFS-MCS: 1.00–1.86× speedup (1.22× on average)

Parallel

LEXI-ORDER: 0.86–1.88× speedup (1.27× on average).

BFS-MCS: 0.59–1.36× speedup (1.04× on average).

The qualitative effect on performance:
less for CSF-MTTKRP than HiCOO-MTTKRP.

Other reordering methods

On tensors nell2, nell1, and deli

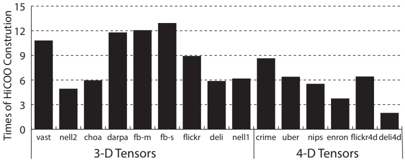
- The reordering methods used in SPLATT:
  - The speedups using graph partitioning: 1.06, 1.11, and 1.19×
  - The speedups using hypergraph partitioning: 1.06, 1.12, and 1.24×.

- For comparison (HiCOO sequential):
  - BFS-MCS: 1.04, 1.64, and 1.61× speedups.
  - LEXI-ORDER: 1.04, 1.70, and 2.24× speedups.

Smith et al. IPDPS'15.

Experiments: Reordering overhead

The overhead of LEXI-ORDER with five iterations.



The ratio of parallel LEXI-ORDER time to parallel HICOO construction time.

$\Rightarrow$ in the range of 1.97–12.91$\times$.

HICOO times	2.36	14.00	4.42	2.71	13.49	17.91	11.06	29.14	44.47	0.84	0.71	0.71	17.53	20.91	105.71

Can do less iterations.

Concluding remarks

- The problem of reordering a tensor to improve block density for tensor computations (for HiCOO).
- Two heuristics: BFS-MCS and Lexi-Order.
- Lexi-Order obtains large MTTKRP speedup for both sequential and multicore implementations of HiCOO, and some speedup on COO and CSF formats.
- BFS-MCS has lower overhead but does not improve as much.

- Future work:
  - Automatic performance tuning: different storage formats and HiCOO parameters.
  - Better and faster heuristics.

Thanks for your attention.

More information:

Jiajia Li http://www.jiajiali.org/
http://perso.ens-lyon.fr/bora.ucar/

- Jiajia Li, J. Sun, and R. Vuduc, "HiCOO: Hierarchical storage of sparse tensors," in *Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Dallas, TX, USA, November 2018, (best student paper award).
- Jiajia Li, BU., U. V. Çatalyürek, J. Sun, K. Barker, R. Vuduc, "Efficient and effective sparse tensor reordering", in preparation.

matLexiOrder: Proposed variant for matrices

Assume an ordering of the rows, sort the columns lexically in linear time.

Key: an order preserving variant of the partition refinement method.

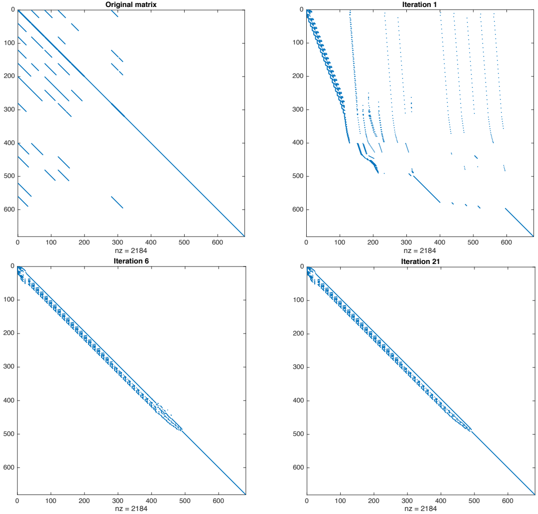
Order preserving partition refinement: High level description

- All columns are initially in a single part.
- $\mathbf{A}$'s nonzeros are visited row-by-row.
- At a row i, each column part C is split into two parts
$$C_1 = C \cap \mathbf{A}(i,:) \text{ and } C_2 = C \setminus \mathbf{A}(i,:)$$
and these two parts replace C in the order $C_1 \succ C_2$.

- The given partition is refined with row i in $\mathcal{O}(|\mathbf{A}(i,:)|)$ time.
- $\mathcal{O}(\text{nnz} + I + J)$ time per iteration and $\mathcal{O}(J)$ space, for an $I \times J$ matrix $\mathbf{A}$.

Paige and Tarjan'87 formalize the partition refinement method

Demonstrating `matLexiOrder`



Experiments: Data set

From FROSTT, http://frostt.io

Tensors	Order	Dimensions	#nnzs	Density
vast	3	$165K \times 11K \times 2$	26M	6.9×10^{-3}
nell2	3	$12K \times 9K \times 29K$	77M	2.4×10^{-5}
choa	3	$712K \times 10K \times 767$	27M	5.0×10^{-6}
darpa	3	$22K \times 22K \times 24M$	28M	2.4×10^{-9}
fb-m	3	$23M \times 23M \times 166$	100M	1.1×10^{-9}
fb-s	3	$39M \times 39M \times 532$	140M	1.7×10^{-10}
flickr	3	$320K \times 28M \times 2M$	113M	7.8×10^{-12}
deli	3	$533K \times 17M \times 3M$	140M	6.1×10^{-12}
nell1	3	$2.9M \times 2.1M \times 25M$	144M	9.1×10^{-13}
crime	4	$6K \times 24 \times 77 \times 32$	5M	1.5×10^{-2}
uber	4	$183 \times 24 \times 1140 \times 1717$	3M	3.9×10^{-4}
nips	4	$2K \times 3K \times 14K \times 17$	3M	1.8×10^{-6}
enron	4	$6K \times 6K \times 244K \times 1K$	54M	5.5×10^{-9}
flickr4d	4	$320K \times 28M \times 2M \times 731$	113M	1.1×10^{-14}
deli4d	4	$533K \times 17M \times 3M \times 1K$	140M	4.3×10^{-15}

HiCOO data structure

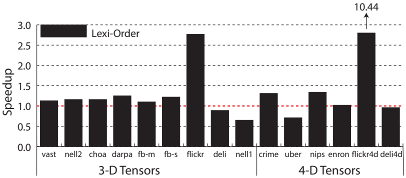
HiCOO parameters α_b and $\overline{c_b}$.

Tensors	Random reordering α_b	Random reordering $\overline{c_b}$	Lexi-Order α_b	Lexi-Order $\overline{c_b}$	Speedup seq	Speedup omp	Storage ratio
vast	0.004	1.758	0.004	1.562	1.01	1.01	0.999
nell2	0.020	0.314	0.008	0.074	1.04	1.13	0.966
choa	0.089	0.057	0.016	0.056	1.16	1.44	0.833
darpa	0.796	0.009	0.018	0.113	3.74	1.54	0.322
fb-m	0.985	0.008	0.086	0.021	3.84	1.13	0.335
fb-s	0.982	0.008	0.099	0.020	3.90	1.29	0.336
flickr	0.999	0.008	0.097	0.025	4.14	4.54	0.277
deli	0.988	0.008	0.501	0.010	2.24	0.86	0.634
nell1	0.998	0.008	0.744	0.009	1.70	0.71	0.812
crime	0.001	37.702	0.001	8.978	0.99	1.37	1.000
uber	0.041	0.469	0.011	0.270	1.00	0.83	0.838
nips	0.016	0.434	0.004	0.435	1.03	1.38	0.921
enron	0.290	0.017	0.045	0.030	1.25	1.76	0.573
flickr4d	0.999	0.008	0.148	0.020	3.02	11.88	0.214
deli4d	0.998	0.008	0.596	0.010	1.76	1.24	0.697

- The block ratio (α_b) and the average slice size per block ($\overline{c_b}$).
- Smaller α_b and larger $\overline{c_b}$ are good.
- HiCOO storage gets compressed.

When both parameters α_b and $\overline{c_b}$ are largely improved, we see a good speedup and storage ratio.

Experiments: CPD speedup



CPD performance on reordered tensors.

LEXI-ORDER has similar effect on the parallel HICOO-CPD.

0.65–10.44× speedup (1.81× on average).

Reordering enhances the performance of CPD.

References I

M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Efficient and scalable computations with sparse tensors," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, Sept 2012, pp. 1–6.

A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in 21st SPAA, 2009, pp. 233–244.

T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi, "A unified optimization approach for sparse tensor operations on GPUs," in 2017 IEEE International Conference on Cluster Computing (CLUSTER), Sept 2017, pp. 47–57.

A. Lubiw, "Doubly lexical orderings of matrices," *SIAM Journal on Computing*, vol. 16, no. 5, pp. 854–879, 1987.

References II

- R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM Journal on Computing*, vol. 16, no. 6, pp. 973–989, Dec. 1987.
- S. Smith, N. Ravindran, N. Sidiropoulos, and G. Karypis, "SPLATT: The Surprisingly ParalleL spArse Tensor Toolkit (Version 1.1.1)," Available from `https://github.com/ShadenSmith/splatt`, 2016.