

University of Alabama in Huntsville

LOUIS

Honors Capstone Projects and Theses

Honors College

4-30-2023

Optimizing Sparse Tensors Computations via Orderings

Trevor Joseph Garnett

Follow this and additional works at: <https://louis.uah.edu/honors-capstones>

Recommended Citation

Garnett, Trevor Joseph, "Optimizing Sparse Tensors Computations via Orderings" (2023). *Honors Capstone Projects and Theses*. 801.
<https://louis.uah.edu/honors-capstones/801>

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.

Optimizing Sparse Tensors Computations via Orderings

by

Trevor Joseph Garnett

An Honors Capstone

**submitted in partial fulfillment of the requirements
for the Honors Diploma**

to

The Honors College

of

The University of Alabama in Huntsville

4/30/2023

Honors Capstone Director: Dr. Joshua Booth

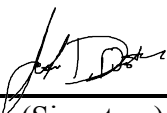
Assistant Professor of Computer Science



4/30/2023

Student (Signature)

Date



4/30/23

Director (Signature)

Date

Department Chair (Signature)

Date

Honors College Dean (Signature)

Date



Honors College Frank Franz
Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the final manuscript. In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Trevor Garnett

Student Name (printed)

Trevor Garnett

Student Signature

4/30/2023

Date

Contents

Dedication	4
Abstract	5
Introduction	6
Previous Work	10
Methodology	10
Results	11
Conclusion	17
Acknowledgements	18
References	19

Dedication

I want to thank my advisor, Dr. Joshua Booth for supporting me these past few years and encouraging me to continue my education into graduate school. I would also like to thank Dr. Wilkerson, Dean of the Honors College for being my mentor, guiding me through tough semesters and being a friend. Lastly, I want to thank my family for believing in me and helping me get to where I am today.

Abstract

Multi-dimensional arrays - or *tensors* - are used in many fields, such as “signal processing and recommender systems” (Smith et al.). Being used in both commonplace and scientific applications, a lot of effort is focused on optimizing tensor computations. In application, tensors tend to be enormous and very sparse. If a tensor “is sparse, ... most of its entries are zero and, therefore, need not be stored nor explicitly computed upon” (Li et al., 2019). Thus, there are opportunities to improve time performance in tensor computations by better utilizing the non-zeros patterns. In order to utilize non-zero patterns, we commonly reorder tensors. Lately, there has been a lot of effort being focused on creating highly optimized tools that exploit the sparsity of tensors in trying to improve performance.

The purpose of this research is to study some of these tools. Specifically, the *efficiency* and *effectiveness* of different reordering schemes are examined. This was done by ordering a sparse tensor with different reordering schemes, using the time to do so as a measurement of efficiency. Then, we perform canonical polyadic decomposition (CPD), a common tensor computation performed in many fields, on both the original tensor and on the reordered tensor. The difference in time taken between the operation on the original tensor and the ordered tensor was used as a measurement of effectiveness.

In this study, we will compare SPLATT and LEXI-order as orderings for sparse tensors. In the paper that LEXI-order was introduced, Li et al., 2019 find “BFS-MCS and LEXI-Order schemes both outperform graph and hypergraph partitionings” from SPLATT. This research is the first work to compare different ordering schemes, other than the paper that introduced LEXI-Order. Thus, this study serves to confirm their results and it will also be a guideline for future research in the field. It was found that LEXI-Order is much more efficient, and that LEXI-Order and 3-Partition SPLATT are similarly effective, since they see similar average speedup.

Introduction

Tensors are a generalization of scalars, vectors, and matrices to higher dimensions, meaning they are just tensors that span 0, 1, and 2 dimensions, respectively. When a tensor spans three or more dimensions, they are simply known as n -dimensional tensors, where n is the number of dimensions that are spanned. Terms such as *mode* and *rank* are used to specify the number of dimensions that are spanned by the tensor. In application, tensors can be represented as multi-dimensional arrays and operations are performed on them. They are used in many fields in both mathematics and computer science – especially in the field of machine learning. According to IEEE fellow Nicholas D. Sidiropoulos et al., “tensors have already found many applications in signal processing (speech, audio, communications, radar, biomedical), machine learning (clustering, dimensionality reduction, latent factor models, subspace learning), and well beyond”. Being used in so many fields, there is a lot of effort being put into developing robust and flexible tool kits that make tensor computations more efficient.

In this paper, we are specifically observing *sparse tensors*. This is actually a big branch in the research into tensor computations, as tensors tend to be sparse in real world applications and being sparse yields opportunities to produce speedups in certain operations. To understand the term *sparse*, we first need to understand the term *density*. The *density* of a tensor A ($Density(A)$) is defined as the proportion of non-zeros elements (nnz) relative to the total number of elements in a tensor, A ($|A|$). Thus, mathematically:

$$Density(A) = \frac{nnz}{|A|}.$$

The quantity of non-zero elements cannot be less than zero or greater than the size of the tensor, i.e., $0 \leq Density(A) \leq 1$. The term *sparse* is used when the density of a tensor is close to 0, which would imply a tensor has very few non-zero elements compared to the quantity of 0 elements. Similarly, a tensor is known as *dense* when the density is close to its maximum, 1. There is not a defined cutoff on what makes a tensor sparse. However, in this

Tensor	# of Non-Zeros	Dimensions	Density
nell-1	143,599,552	2,902,330 x 2,143,368 x 25,495,389	9.054E-13
nell-2	76,879,419	12,092 x 9,184 x 28,818	2.402E-05
nips Publications	3,101,609	2,482 x 2,862 x 14,036 x 17	1.830E-06
crime	5,330,673	6,186 x 24 x 77 x 32	1.457E-02
enron	54,202,099	6,066 x 5,699 x 244,268 x 1,176	5.458E-09
flickr	112,890,310	319,686 x 28,153,045 x 1,607,191 x 731	1.068E-14
delicious	140,126,181	532,924 x 17,262,471 x 2,480,308 x 1,443	4.256E-15

Table 1: the tensor collection and their density statistics

research, the density of the tensors were all less than 0.05. Specifically, they ranged in value from 4.256E-15 to 1.457E-02, the densities of delicious and crime, respectively. Observe Table 1, which contains the number of non-zero elements, the size, and the density of each tensor used in this study. All the tensors, and their statistics, used for this study can be found at <http://frostdt.io/tensors/>. These tensors were chosen as they are also by comparable works (Li et al. 2019, Smith et al. 2015).

Spatial and Temporal Locality

We are specifically interested in sparse tensors because there are opportunities to produce speedups by improving either *temporal locality* or *spatial locality* (ideally both). These terms simply represent the concept of making better memory management decisions. A computer's memory hierarchy and memory management is analogous to the challenge of physical file storage, in that we have much greater capacity to store files in a filing cabinet or a storage room (hard drive), compared to what can be carried on our person (memory close to the CPU). Additionally, it takes more time to retrieve a file from a filing cabinet than it does retrieving a file on your person, since the filing cabinet is more distant. The term *spatial locality* simply refers to the concept of storing related files closely together when they are in the filing cabinet so that they can be retrieved at the same time more easily. Over the course of a program, this can yield speedups because time can be saved if the related data entries are stored close to one another. *Temporal locality* refers to the concept of not having to retrieve the same file multiple times from the filing cabinet over the course of a day. Because we can only have so many files on our person, we may need to return a file to the filing cabinet prior to us being done with it for the day. Ideally, we would like to minimize the number of trips retrieving the same file over the course of the day. Over the course of a program, we could produce speedups if we minimize the number of trips taken to retrieve the same file.

To summarize, spatial locality is a measure of the proximity of related data entries, physically, in memory. Spatial locality is good when data entries that are mutually relevant are stored closely together physically so that they can be retrieved together. Spatial locality is poor when mutually relevant data entries are stored distantly and take more than one trip to retrieve. Additionally, temporal locality is a measure of how data is used throughout the lifetime of a program. Temporal locality is good when, for most data entries, we can minimize the number of trips taken to retrieve the same data from the hard drive. Temporal locality is poor when the data entries must be retrieved multiple times, sporadically, throughout the lifetime of the program. Thus, if we can improve temporal locality or spatial locality, speedups

should follow.

Methods of Improving Temporal and Spatial Locality

In general, there are two overall approaches when it comes to trying to improve spatial and temporal locality. Some work is in favor of using storing the tensors in a different formats. The “(COO) format, [which would be storing a tensor in a multi-dimensional array, is] arguably the de facto standard for general sparse tensor storage” (Li et al., 2018). There are alternative storage formats. The other approach is to order the indices of a tensor, using an ordering. In this paper, we will be evaluating two different orderings, SPLATT and LEXI-Order, while also varying the input parameters to SPLATT.

Reorderings

A *reordering*, or ordering, is the process of “permuting the indices within one or more modes”(Smith et al.) of a tensor. This is equivalent to rearranging the rows and columns of a matrix when doing row operations. A reorderings “can lead to significant performance gains as it can potentially... [introduce speedups] by exploiting spatial and temporal locality”(Smith et al.). Thus, the goal is to rearrange the indices of a tensor to arrive at an equivalent representation of a tensor, but with optimal spatial and temporal locality. Figure 1 depicts a tensor undergoing such an ordering. However, the problem of finding a perfect reordering schema is NP-hard and thus does not have a solution. There are several heuristics that have been introduced that attempt to find an approximate solution. Meaning, although they are not guaranteed to provide an optimal ordering, they should be able to provide an ordering with better spatial and temporal locality. In this study, we will be comparing orderings SPLATT, introduced in *SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication* by Smith et al., and LEXI-Order, which was introduced in *Efficient and Effective Sparse Tensor Reordering* by Li et al., 2019.

0	3	0	3		0	0	0	0		2	0	0	2		3	3	0	0		0	0	0	0		0	0	0	0
0	0	0	0		1	0	1	0		0	0	0	0	→	3	3	0	0		0	2	2	0		0	0	0	0
0	0	0	0		1	0	1	0		0	0	0	0		0	0	0	0		0	2	2	0		0	0	1	1
0	3	0	3		0	0	0	0		2	0	0	2		0	0	0	0		0	0	0	0		0	0	1	1

Figure 1: example of a 3-dimensional tensor reordered to show greater spatial locality

Previous Work

There is plenty of research into the optimization of tensor algorithms; however, many of these papers do not work with orderings. Smith et al. in *SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication* introduce two ordering schemes that are *mode-independent* and *mode-dependent* orderings; these are known, respectively, as the graph ordering and hypergraph ordering. According to Smith et al., “Mode-dependent reorderings offer further opportunities for optimization at the cost of additional work during the reordering stage”. Additionally, *Efficient and Effective Sparse Tensor Reordering* (Li et al., 2019) introduces both BFS-MCS and LEXI-Order. In their findings, “LEXI-Order improves performance more than BFS-MCS in most cases” (Li et al., 2019). Because LEXI-order performs better than BFS-MCS and is mode independent, we want to compare LEXI-order to SPLATT’s mode independent ordering.

Methodology

In this study, we observe SPLATT’s mode-independent ordering with both 3-way and 6-way partitionings and LEXI-order. We first test the efficiency of the orderings by measuring the time it takes to reorder the tensor. Since both of these orderings are parallel, this was done with 1, 8, and 32 cores. To test the effectiveness, we perform canonical polyadic decomposition (CPD), on both the ordered and the reordered tensor to measure time performance. Tensor decompositions is an important kernel that is necessary for most tensor computations. Although there are many decompositions, CPD is the one most commonly used in literature. A reordering is considered more effective if it improves time performance compared to the original. For statistical significance, each of these processes are repeated a total of four times

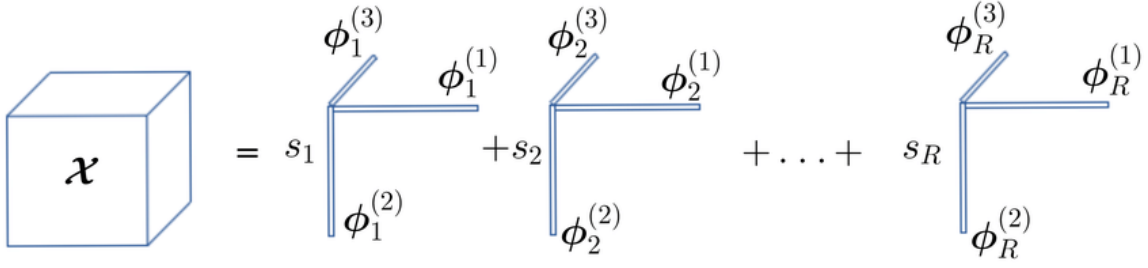


Figure 2: depiction of cpd breaking a tensor into rank-one tensors

and then averaged.

Canonical Polyadic Decomposition

Several decompositions could have been chosen for this study. However, Canonical Polyadic Decomposition (CPD) is a very popular computation utilized in many fields. Additionally, CPD is commonly used as a benchmark in the study of both sparse and non-sparse tensor computations. Thus, we have chosen to use CPD to be our benchmark. For the purposes of this study, we only need to know that, according to Evert et al., “The canonical polyadic decomposition (CPD) is a fundamental tensor decomposition which expresses a tensor as a sum of rank one tensors. In stark contrast to the matrix case, with light assumptions, the CPD of a low rank tensor is (essentially) unique”(Evert et al.). Unique means that there is only one such decomposition for a given tensor. Figure 2 depicts a tensor X being decomposed into a summation of rank-one tensors. It should be noted that the literature sometimes refer to CPD as “CANDECOMP/PARAFAC Decomposition”, if the reader would like to continue researching CPD.

Results

The purpose of this paper was to study the efficiency and effectiveness of SPLATT 3-way partition ordering, SPLATT 6-way partition ordering, and LEXI-Order. Figure 3 showcases the amount of time taken to order the tensors using LEXI-Order for 1, 8, and 32 cores. From this graph, we can easily see that LEXI-order is highly parallelizable, as the time to complete

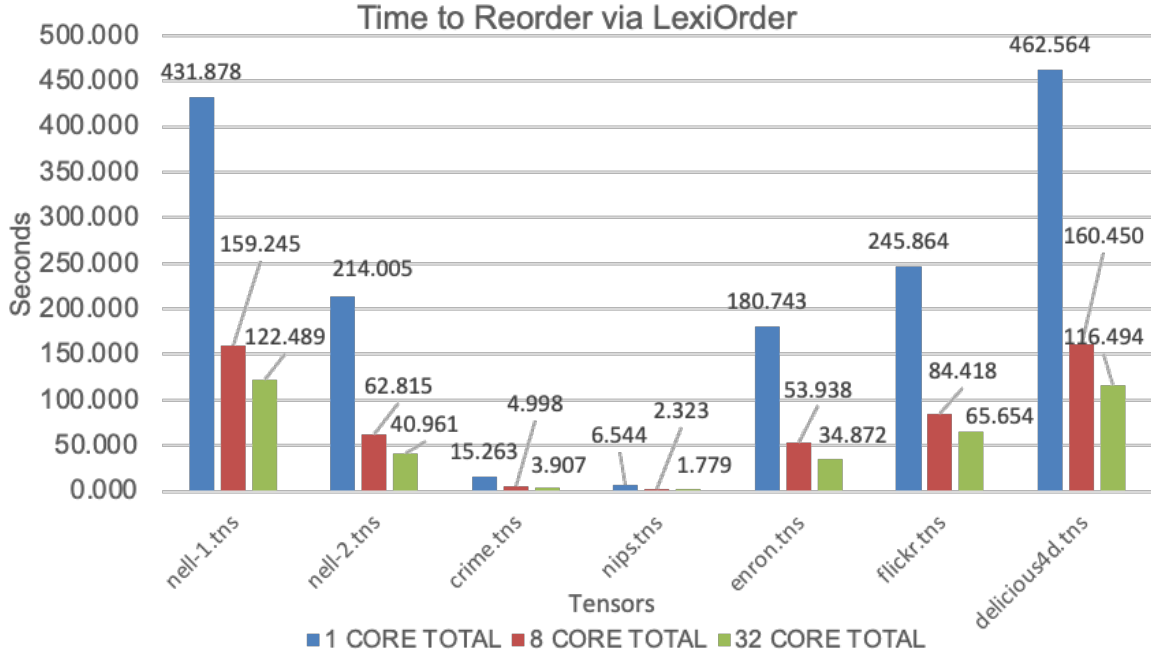


Figure 3: the time it takes for lexi-order to complete

LEXI-Order is decreasing dramatically, seemingly exponentially. Now, observe Figure 4 that displays the amount of time utilized to permute the tensors using SPLATT’s 3-way and 6-way partition ordering for 1, 8, and 32-cores. The first thing to note is that SPLATT’s 3-way partition ordering is consistently faster than its 6-way partition ordering. Excluding nips, which was ordered 22x faster via a 3-way partition than a 6-way partition, a 3-way partition averages to be over 3% faster than a 6-way partition ordering.

Comparing the efficiency of the two, we observe that SPLATT, both 3-way and 6-way partitioning, is faster sequentially on nell-2, crime, and enron while LEXI-Order is faster sequentially on nell-1, nips, flickr, and delicious. Thus, the performance on one core is dependent on the tensor. However, when parallelism is introduced, LEXI-Order appears to scale really well while SPLATT does not. LEXI-Order is consistently faster with the higher core counts. Thus, with both 8 and 32 cores, LEXI-Order is much more efficient than either of SPLATT’s orderings. Overall, we observe that LEXI-Order is more efficient.

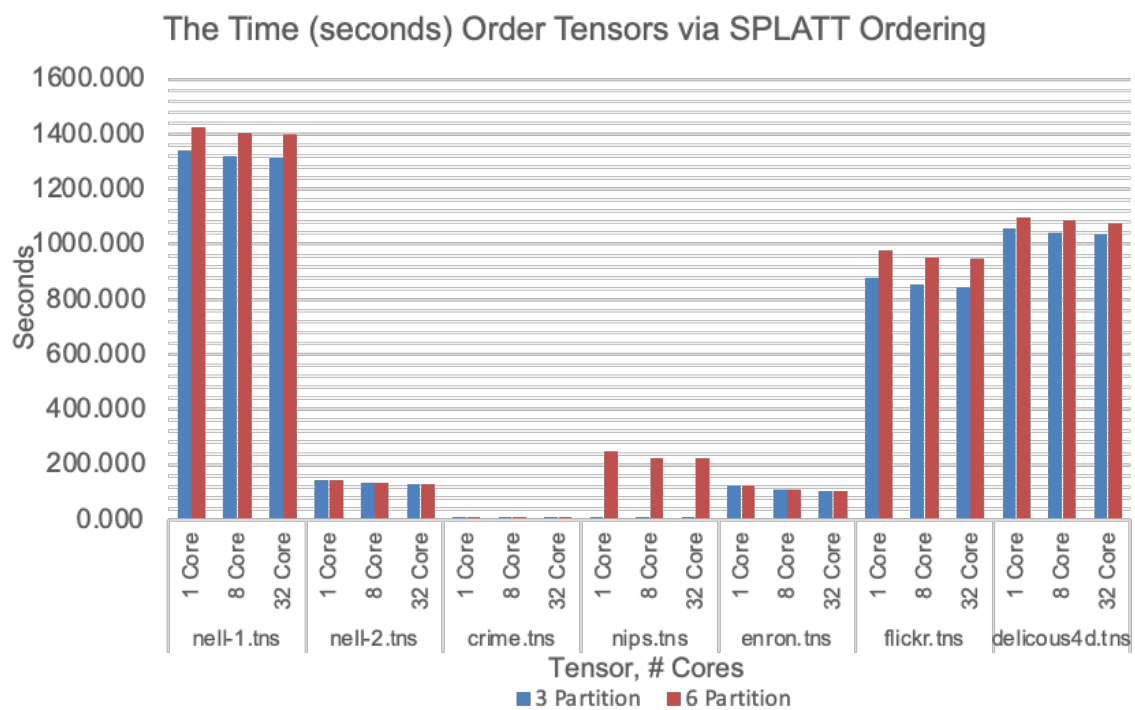


Figure 4: the time it takes to partition a graph 3-way and 6-way

We used time performance as a measure of effectiveness. Because CPD requires more time on larger tensors, we must define how to compare the performance improvements between tensors of different sizes. We will define speedup as follows:

$$Speedup = (timewithnoordering)/timewithordering.$$

If an ordering improves time performance of CPD on a tensor, CPD would take less time. Thus, in this case, $speedup > 1$. If CPD takes more time to complete on the ordered tensor than on the original, $speedup < 1$. Speedup can be used to compare the effectiveness of orderings on different tensors because it measures differences in time performance relative to the time it took on the original tensor, relative being the keyword.

Figure 5, Figure 6, and Figure 7 all display the speedup acquired by an ordering on each tensor, performing CPD with 1, 8, and 32 cores, respectively. In each figure, points above the blue line have values greater than one, and therefore indicate an improvement in performance produced by the ordering. Points below the blue line indicate a decrease in performance. It is interesting to see that so many points lie below the blue line in each figure, meaning that the orderings produced slowdowns ($speedups < 1$).

In sequential CPD, seen in Figure 5, both nips and nell-1 experience performance improvements for all three orderings. Then, in crime, performance is improved with LEXI-order and SPLATT's 3-way partition ordering. Every other tensor experience a slowdown under sequential. When CPD was computed with 8 and 32 cores, all tensors except for nips, experienced a slowdown for at least one of the orderings, and no ordering produced a significant performance improvement. nips did see a speedup of around 1.9 on a SPLATT 3-way partition ordering.

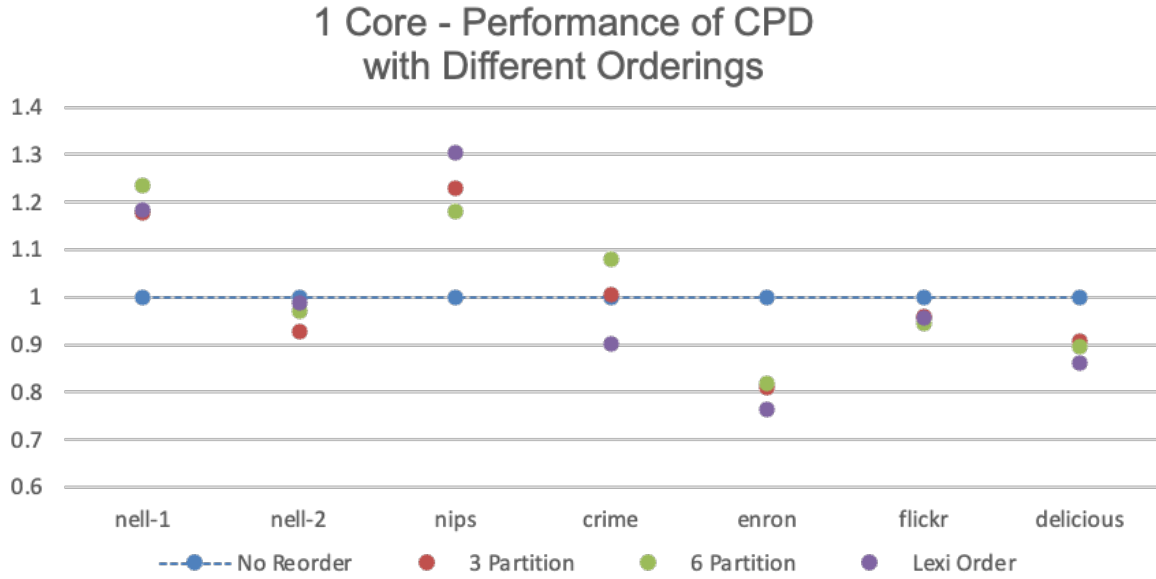


Figure 5: the speedups produced by the orderings when performing cpd with 1 core

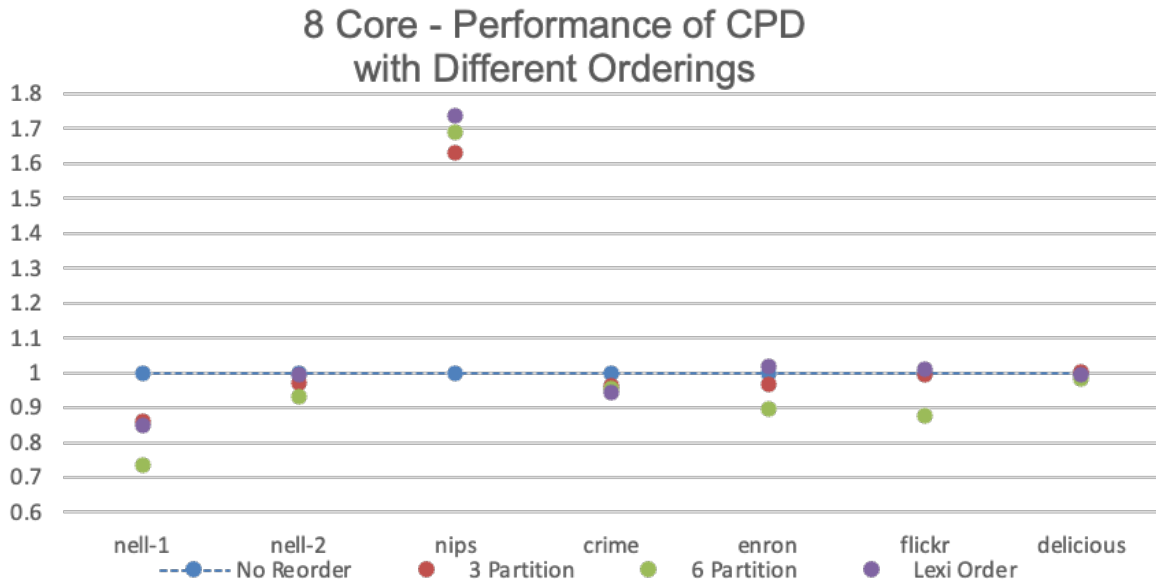


Figure 6: the speedups produced by the orderings when performing cpd with 8 cores

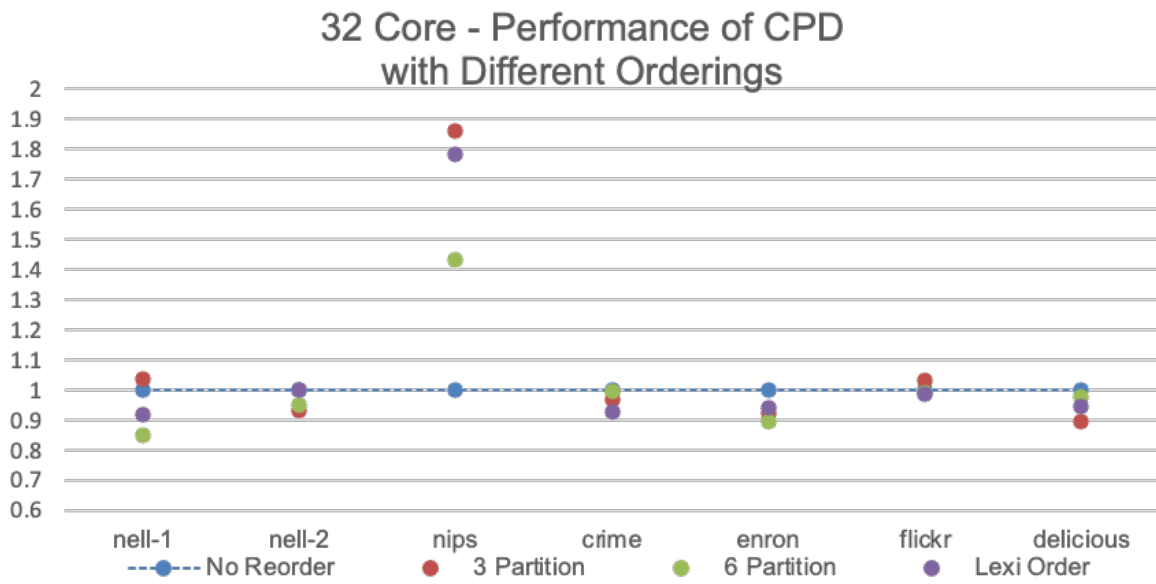


Figure 7: the speedups produced by the orderings when performing cpd with 32 cores

Conclusion

This study wants to compare LEXI-Order with SPLATT’s 3-way and 6-way partition ordering in terms of efficiency and effectiveness. Efficiency is defined as the time to perform the ordering on the graph, while effectiveness is compared by using measurements of improvements in time performance.

The data dictates that LEXI-Order is more efficient than SPLATT’s partitioning, both 3-way and 6-way. This is because, in a multi-core environment, the LEXI-Order ordering transmutates the tensor quicker than SPLATT’s 3-way and 6-way partition ordering. Additionally, sequential LEXI-Order is generally faster, although it is dependent on the tensor. Thus, LEXI-order is more efficient than SPLATT, since it takes less time to perform the ordering in most cases.

Although LEXI-Order is more efficient, the compute power utilized to perform the ordering may not be justified. All orderings fail at producing significant speedups in the tensor suite as a whole. Thus, it is difficult to dictate which ordering is more effective. From our results, most tensors do not experience a significant speedup with either ordering, in general. Only the nips tensor experienced a significant speedup in the multi-core environment. In the sequential environment, only nips and nell-1 experience speedups. Outside of these tensors, most tensors experience a slowdown. This means that performing an ordering on the tensor is likely to decrease the performance of CPD.

It is interesting that, in *Efficient and Effective Sparse Tensor Reordering*, Li et al., 2019 find better performance improvements for LEXI-Order for the same tensors. Instead of using the raw tensor, they used a randomized ordering as their base case for the CPD computations. This raises a question whether there is some pattern that is formed in the data collection process that yields non-zero patterns that is beneficial for computations such as CPD. This is a potential area for future study.

Acknowledgements

We would like to thank the UAH Office of the President, Office of the Provost, Office of the Vice President for Research and Economic Development, The Dean of the College of Science, the Dean of the College of Engineering, the Alabama Louis Stokes Alliances for Minority Participation, and the Alabama Space Grant Consortium for funding during the RCEU program. This reserach continues under funding by NSF Grant 2044633 “CAREER: Fast, Energy Efficient Irregular Kernels via Neural Acceleration”

References

- Evert, E., Vandecappelle, M., & De Lathauwer, L. (2022, February 23). Canonical polyadic decomposition via the generalized Schur decomposition. arXiv.org. Retrieved April 28, 2023, from <https://arxiv.org/abs/2202.11414>\#:~:text=The%20canonical%20polyadic%20decomposition%20(CPD,tensor%20is%20(essentially)%20unique.
- Li, J., Sun, J., & Vuduc, R. (2018). HiCOO: Hierarchical Storage of Sparse Tensors. SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 2018, pp. 238-252, doi: 10.1109/SC.2018.00022.
- Li, J., Uçar, B., Çatalyürek, Ü. V., Sun, J., Barker, K., & Vuduc, R. (2019, October 24). Efficient and effective sparse tensor reordering. Accueil - Archive ouverte HAL. Retrieved April 28, 2023, from <https://hal.archives-ouvertes.fr/hal-02306569>
- Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., & Faloutsos, C. Tensor Decomposition for Signal Processing and Machine Learning. In IEEE Transactions on Signal Processing, vol. 65, no. 13, pp. 3551-3582, 1 July1, 2017, doi: 10.1109/TSP.2017.2690524.
- Smith, S., Ravindran, N., Sidiropoulos, N. D., & Karypis, G. (2015). SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication. IEEE Explore. Retrieved April 28, 2023, from <https://ieeexplore.ieee.org/document/7161496>