An Efficient Mixed-Mode Representation of Sparse Tensors

Israt Nisa
Ohio State University
nisa.1@osu.edu

Jiajia Li
Pacific Northwest National Laboratory
Jiajia.Li@pnnl.gov

Aravind Sukumaran-Rajam
Ohio State University
sukumaranrajam.1@osu.edu

Prasant Singh Rawat
Ohio State University
rawat.15@osu.edu

Sriram Krishnamoorthy
Pacific Northwest National Laboratory
sriram@pnnl.gov

P. Sadayappan
University of Utah
saday@cs.utah.edu

ABSTRACT

The Compressed Sparse Fiber (CSF) representation for sparse tensors is a generalization of the Compressed Sparse Row (CSR) format for sparse matrices. For a tensor with d modes, typical tensor methods such as CANDECOMP/PARAFAC decomposition (CPD) require a sequence of d tensor computations, where efficient memory access with respect to different modes is required for each of them. The straightforward solution is to use d distinct representations of the tensor, with each one being efficient for one of the d computations. However, a d-fold space overhead is often unacceptable in practice, especially with memory-constrained GPUs. In this paper, we present a mixed-mode tensor representation that partitions the tensor's nonzero elements into disjoint sections, each of which is compressed to create fibers along a different mode. Experimental results demonstrate that better performance can be achieved while utilizing only a small fraction of the space required to keep d distinct CSF representations.

CCS CONCEPTS

• **Theory of computation** → *Parallel algorithms*;

KEYWORDS

Sparse tensors, MTTKRP, GPU, CANDECOMP/PARAFAC decomposition

ACM Reference Format:
Israt Nisa, Jiajia Li, Aravind Sukumaran-Rajam, Prasant Singh Rawat, Sriram Krishnamoorthy, and P. Sadayappan. 2019. An Efficient Mixed-Mode Representation of Sparse Tensors. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3295500.3356216

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356216

1 INTRODUCTION

Tensors are multidimensional data commonly used in machine learning [2], text analysis [4], healthcare analytics [16], [17], telecommunications [36], [37], and numerous other applications. Tensors are useful because they provide a generalization of storing data for arbitrary number of dimensions, where each dimension is termed a *mode*. Real world tensors are extremely large and sparse, with high irregularity in shape and distribution of nonzeros. Unlike their dense counterparts, sparse tensors need a compressed storage format to be space efficient.

There exists a vast research history on efficiently representing sparse matrices, which are special tensors with two modes. A natural way of representing sparse matrices is to just store the indices for the non-zero elements, along with its value. One can further optimize the storage by reusing the same row pointer for all the non-zeros in the same row. This format is called Compressed Sparse Row (CSR), and is universally regarded as the *de facto* representation for sparse matrices. For hyper-sparse matrices with many empty rows, Doubly Compressed Sparse Row (DCSR) format [10] further compresses CSR by storing the row pointers for only the non-empty rows. Compressed Sparse Fiber (CSF) is a generalization of CSR (or DCSR) for higher dimensional tensors.

A full iteration of CANDECOMP/PARAFAC decomposition (CPD) or Tucker Decomposition requires performing Matricized Tensor Times Khatri-Rao Products (MTTKRP), or Tensor-Times-Matrix products (TTM) on every mode. Therefore, many state-of-the-art tensor factorization frameworks create a compact representation of a tensor at each mode to achieve an overall high performance. For illustration, consider an application that performs sparse matrix-vector multiplication (SpMV), $y = Ax$, in tandem with sparse matrix-transpose-vector multiplication (SpMTV), $z = A^T x$. If A is stored in CSR format, then parallelism can be achieved across rows while computing $y = Ax$. However, computing $z = A^T x$ with CSR would require explicit locks or atomic operations to update z. Similarly, storing A in Compressed Sparse Column (CSC) format will achieve parallelism for $z = A^T x$, but introduce atomics for $y = Ax$. Explicit synchronization is usually prohibitively expensive on multiple architectures, including GPUs. A naïve solution to this conundrum is to store A in both CSR and CSC formats. The same logic extends to tensors: to achieve parallelism and efficient accesses across d modes, d representations of the tensor are maintained. Clearly, the storage overhead will increase with the number of modes, making this solution impractical for higher order tensors.

This paper attempts to reconcile two conflicting objectives: reducing the overall storage overhead for tensors by using a single representation for all the modes, and achieving equal or better performance compared to the naïve approach of storing d representations. A previous effort in this direction by Smith et al. was proposed in the SPLATT library [32] – given d CSF representations for d modes, their implementation selects the CSF where the shortest dimension is at the outermost level as the only representative. Computation on all the modes will use the same representative CSF. We term this storage method SPLATT-ONE, in contrast to SPLATT-ALL, which represents the strategy of creating a different CSF representation for each mode. Even though selecting one out of d CSF representations is an easy technique to reduce storage overhead, often further analysis in identifying the sparsity structure of the real-world tensors can further improve both parallelism and compression.

In this paper, we propose a novel *mixed-mode* format termed MM-CSF, where long fibers[1] on each mode are first identified and then stored on their most suitable modes. Doing so achieves better compression, which not only reduces space requirement, but also provides performance improvement. Revisiting the illustrative example, while performing SpMV (SpMTV) with the mixed-mode format, the nonzeros in the CSR (CSC) representation can exploit the parallelism and compression in the long rows (columns), and the rest of the nonzeros in the CSC (CSR) representation will require atomics. Figure 1 further elucidates our insight behind mixed-mode. The matrix in the figure is sparse, with only one non-empty row and column. Using either CSR or CSC representation would require storing 38 elements in row pointers, column indices, and nonzeros; simultaneously maintaining both CSR and CSC representations for efficient computation would require storing 76 elements. In contrast, the mixed-mode format will store the dense row in CSR format, and the dense column in CSC format, reducing the overall storage to 32 elements. Furthermore, as illustrated later in the paper, mixed-mode storage incurs fewer global memory transactions due to better compression when compared to SPLATT-ONE, and uncovers a scope for finer grained parallelism when compared to SPLATT-ALL. In summary, this work makes the following contributions.

- It proposes MM-CSF, a novel storage format that exploits the underlying sparsity structure of real-world tensors, and provides compact storage without compromising on computational efficiency of tensor operations.
- It provides experimental data demonstrating that compared to the storage formats used in the state-of-the-art CPU frameworks, MM-CSF can save up to 75% space, while improving MTTKRP performance by up to 63×.
- On a NVIDIA Volta GPU, it demonstrates that MM-CSF outperforms the state-of-the-art GPU storage format, BCSF [26], by up to a factor of 2× in computing MTTKRP, and reduces up to 85% of the space requirement.

The rest of the paper is organized as follows. Section 2 describes the tensor notations and gives an overview of tensor decomposition. Section 3 gives a brief overview of the sparse tensor storage formats used by the state-of-the-art tensor factorization frameworks.

[1]Generalization of matrix rows and columns, created by holding all but one index constant.

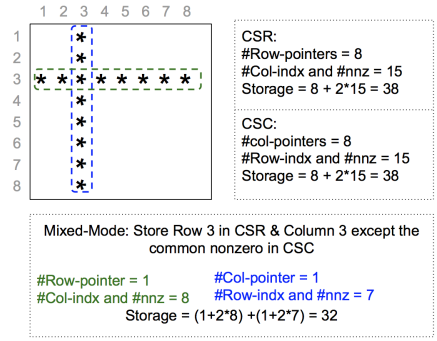


Figure 1: Scope of compression using mixed-mode representation in a matrix

Terminologies	Description
$\mathcal{X}$	Tensor
d	Tensor order
M	Number of nonzeros in $\mathcal{X}$
S_n	Number of slices in $\mathcal{X}$ on mode-n
F_n	Number of fibers in $\mathcal{X}$ on mode-n
F_{B-n}	Number of fibers in BCSF on mode-n
A,B,C	Factor matrices
R	Tensor rank
CSF-n	CSF representation of $\mathcal{X}$ on mode-n
*-ALL	Implementation with d different CSF representations of $\mathcal{X}$
*-ONE	Implementation with one CSF representation of $\mathcal{X}$
MM-CSF	Mixed-mode single CSF representation of $\mathcal{X}$

Table 1: Tensor notations

Sections 4 and 5 discuss our proposed MM-CSF representation in detail and describe the acceleration of MTTKRP computation on GPUs with the MM-CSF representation. Section 6 presents experimental evaluation, Section 7 discusses the related work, and Section 8 concludes.

2 TENSOR BACKGROUND

2.1 Tensor Notation

The tensor notations used in this paper are adapted from Kolda and Sun [21]. We represent tensors as $\mathcal{X}$. Let us assume that $\mathcal{X}$ is a third-order tensor of dimension $I \times J \times K$, and the rank of the tensor is R. The dimensions of a tensor are also known as *modes*. This third-order tensor $\mathcal{X}$ can be decomposed into three factor matrices, $A \in \mathbb{R}^{I \times R}$, $B \in \mathbb{R}^{J \times R}$, and $C \in \mathbb{R}^{K \times R}$. The nonzero elements of $\mathcal{X}$ can be represented as a list of coordinates, $\langle i,j,k,vals \rangle$. Individual elements of $\mathcal{X}$ are expressed as $\mathcal{X}_{ijk}$.

Fibers are vectors generated by fixing all but one index constant. $\mathcal{X}_{:,j,k}$, $\mathcal{X}_{i,:,k}$, $\mathcal{X}_{i,j,:}$ are the examples of fibers. Similarly, *slices* are generated by fixing all but two indices constant, e.g., $\mathcal{X}_{:,:,k}$, $\mathcal{X}_{:,j,:}$, and $\mathcal{X}_{i,:,:}$. Throughout the paper, we append '-ONE' (or '-ALL') to the names of current state-of-the-art tensor factorization frameworks/implementations that maintain one (or d) CSF representation(s) of the input tensor. Table 1 summarizes the rest of the terminologies used throughout the paper.

Algorithm 1: CPD-ALS for third-order tensors

Input : $X \in \mathbb{R}^{I \times J \times K}$
Output: A $\in \mathbb{R}^{I \times R}$, B $\in \mathbb{R}^{J \times R}$, C $\in \mathbb{R}^{K \times R}$

1 **for** *iter = 1* **to** *outer iters or convergence* **do**
2 ▷ MTTKRP on mode-0
3 $Y_1 \leftarrow X_{(1)}$ (C $\odot$ B)
4 A $\leftarrow Y_1$ (B^{T}B * C^{T}C)†
5 normalize columns of A
6 ▷ MTTKRP on mode-1
7 $Y_2 \leftarrow X_{(2)}$ (C $\odot$ A)
8 B $\leftarrow Y_2$ (A^{T}A * C^{T}C)†
9 normalize columns of B
10 ▷ MTTKRP on mode-2
11 $Y_3 \leftarrow X_{(3)}$ (B $\odot$ A)
12 C $\leftarrow Y_2$ (A^{T}A * B^{T}B)†
13 normalize columns of C
14 **return** A, B, C

2.2 CANDECOMP/PARAFAC Decomposition and MTTKRP

CPD is the higher level generalization of Singular Value Decomposition (SVD), a popular matrix decomposition technique. CPD decomposes a tensor into a sum of component rank-one tensors. For example, the third-order tensor $\mathcal{X}$ is decomposed to $a_r \in \mathbb{R}^I$, $b_r \in \mathbb{R}^J$ and $c_r \in \mathbb{R}^K$, where a_r, b_r, b_r are the rank-one components of the factor matrices A, B and C respectively [21]. We can express the decomposition as:

$$\mathcal{X} \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r \tag{1}$$

We implement the alternating least squares (ALS) algorithm [11], [15] to perform CPD in this work, which is also known as the *workhorse* algorithm of it [21]. ALS aims to approximate $\mathcal{X}$ and minimize $||\mathcal{X} - \widetilde{\mathcal{X}}||$. $\widetilde{\mathcal{X}}$ is the approximated tensor generated from the factor matrices (refer to Equation (1)). ALS solves one matrix at a time while holding the others constant. For example, while updating A, matrices B and C will be fixed. The update computation can be described as:

$$A = X_{(1)}(C \odot B)(B^T B * C^T C)^\dagger. \tag{2}$$

The term $X_{(1)}(C \odot B)$ represents Khatri-Rao product between $X_{(1)}$, B, and C; the algorithm computing it is known as the MTTKRP (Matricized Tensor Times Khatri-Rao Product). $X_{(1)}$ is the mode-1 matricization of $\mathcal{X}$. The output of the Khatri-Rao product is then multiplied with $(B^T B * C^T C)^\dagger$, which is the pseudo-inverse of the $R \times R$ matrix generated by $B^T B$ and $C^T C$. Algorithm 1 demonstrates the steps to update each matrix using the ALS algorithm. Line 3, Line 7, and Line 11 show the MTTKRP operations to update A, B, and C respectively. MTTKRP is the performance bottleneck in CPD, primarily due to the access to the large, sparse tensor $\mathcal{X}$, and the scattered access to the factor matrices directed from $\mathcal{X}$.

2.3 The Optimized MTTKRP Algorithm

In the MTTKRP computation, if $(C \odot B)$ is computed separately, then the resulting dense $JK \times R$ matrix could cause memory overflow.

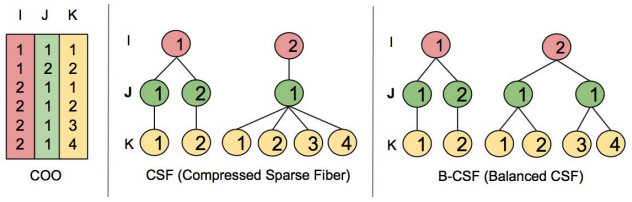


Figure 2: Storage formats of an illustrative sparse tensor

One way to avoid this issue is proposed by Kang et al. [19], which uses operator distributivity to compute the Hadamard products between B and $\mathcal{X}$, and C and $\mathcal{X}$. Based on this algorithm, Smith et al. [31] provide an optimized formulation to save flops and memory accesses, as shown below:

$$Y_1(i,:) = \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} X(i,j,k)(B(j,:) * C(k,:)) \tag{3}$$

$$= \sum_{k=0}^{K-1} C(k,:) * \sum_{j=0}^{J-1} X(i,j,k) * (B(j,:)) \tag{4}$$

In Equation (4), each $\mathcal{X}_{(i,:,k)}$ fiber has a scope to save $R(J-1)$ multiplications.

3 SPARSE TENSOR FORMATS

There are two primary families of sparse tensor formats: coordinate-based and tree-based. A straightforward approach to represent a sparse tensor is to store, for each nonzero, its indices along each dimension and the value. This storage format is called COO (Coordinate) format. Recently, Li et al. [23] proposed Hierarchical COO (HiCOO) format based on COO, which further compresses indices by multi-level blocking. Flagged-COO format [25] also belongs to the coordinate family. The tree-based format family compresses the sparse tensor indices into a "tree structure". CSF, proposed by Smith et al. [35], and BCSF, proposed by Nisa et al. [26], belong to this family. Since the tree-based formats are the focus of this work, we now analyze CSF and BCSF formats in terms of storage, floating point operations, memory access, and the number of required representations.

3.1 Storage and Floating Point Operations

Figure 2 shows the data structures of COO, CSF, and BCSF for a third-order tensor. COO requires $3 \times 4 \times M$ bytes to store the indices, where M is the total number of nonzeros, and each index is a 4-byte integer. CSF organizes the dimension of the tensor in a hierarchical manner and compresses repetitive indices. As shown in the CSF tree in Figure 2(b), the leaves at the lowest level store the indices of the M nonzeros at K dimension. The nonzeros sharing the same indices at J dimension are compressed to F fibers, and the fibers sharing the same indices at I dimension are grouped into S slices/roots. This particular hierarchical organization can be represented as $I \rightarrow J \rightarrow K$. In general, such hierarchical organizations are called *mode orientation*. Thus, a tensor with S slices, F fibers, and M nonzeros requires $4 \times (2S + 2F + M)$ bytes to represent its indices. For slices and fibers, two arrays are maintained to store

Algorithm 2: slice-alg: MTTKRP using BCSF for third-order tensors on GPUs [26]

Input : $slicePtr[S]$, $sliceInds[S]$, $fiberPtr[F_B]$, $fiberInds[F_B]$, $indK[M]$, $vals[M]$, dense matrices $B[J][R]$, $C[K][R]$
Output: dense matrix $\widetilde{A}[I][R]$

1 ▷ slices-parallel across thread-blocks
2 **for** *slice = 0* **to** S **do**
3 i = $sliceInds$[slice]
4 ▷ fibers-parallel across warps
5 **for** *fiber = slicePtr[slice]* **to** *slicePtr[slice + 1]* **do**
6 j = $fiberInds$[fiber]
7 ▷ nonzeros-cyclically processed by warps
8 **for** *z = fiberPtr[fiber]* **to** $fiberPtr_B$*[fiber + 1]* **do**
9 k = $indK[z]$
10 ▷ rank-parallel across threads
11 **for** *r = 0* **to** R **do**
12 $tmp[r] += vals[z] * C[k][r]$ ▷ register accumulation
13 **for** *r = 0* **to** R **do**
14 $tmp\_2[r] += tmp[r] * B[j][r]$ ▷ register accumulation
15 **for** *r = 0* **to** R **do**
16 $\widetilde{A}[i][r] += tmp\_2[r]$ ▷ Atomic writes
17 **return** $\widetilde{A}$

Algorithm 3: MTTKRP using d representations on d modes for an order-d tensor

Input : $indI[M]$, $indJ[M]$, $indK[M]$, $vals[M]$
Output: $\widetilde{A}[I][R]$, $\widetilde{B}[J][R]$, $\widetilde{C}[K][R]$

1 ▷ create CSFs
2 **for** *mode = 0* **to** $d-1$ **do**
3 CSF[mode] = create_CSF (mode)
4 ▷ execute MTTKRP
5 **for** *mode = 0* **to** $d-1$ **do**
6 slice-alg (CSF[mode])

their pointers and indices. SPLATT [35] library provides a highly optimized MTTKRP implementation using CSF data structures on the CPU. Figure 2(c) shows the balanced CSF structure. BCSF [26] extends CSF to GPU platforms, and provides a balanced data structure to store the CSF. The nonzero elements per fiber and per slice might vary significantly across a tensor. BCSF splits the heavy fibers into sub-fibers and creates F_B fibers, where $F_B \geq F$. The slices are grouped into multiple bins based on their length; each bin will have a different number of thread blocks assigned to one slice. BCSF increases GPU occupancy significantly with load balancing and other relevant optimizations [26]. Algorithm 2 shows the computation of MTTKRP on mode-0 to update A using BCSF. The first *for* loop as shown in Line 2 will iterate over all the slices; each slice will in turn iterate over its fibers in Line 2. Each nonzero element of the fibers will perform a multiplication with the corresponding row of C. The product will then be multiplied with the corresponding row of B (Line 10) and written back to the row of A (Line 16).

In real-world tensors, $S \ll M$ and $F \ll M$. Hence, CSF-based structures have a potential to reduce the floating point operations as well. As shown in Equation (4), and also in Algorithm 2, $R(J-1)$ flops can be saved by factoring C out; doing so reduces the required flops to $R(S + 2 \times (F + M))$. In contrast, the flops requirement of COO-based formats is $3 \times M \times R$.

	fiber length of the selected mode				fiber length of a candidate mode			
	0 ∼100	100∼5K	5K∼10K	>10K	0 ∼100	100∼5K	5K∼10K	>10K
deli	37M	76K	17	6	47M	3K	5	0
nell1	17M	136K	233	77	113M	11K	4	0
nell2	46K	291K	5	0	16M	66K	0	0
flick	13M	166K	70	11	28M	1K	1	0
fr_m	**61M**	**19.4K**	**42**	**32**	**60M**	**19.6K**	**48**	**50**
fr_s	**91M**	**28K**	**90**	**59**	**86M**	**30K**	**119**	**105**
darpa	54K	22K	164	277	28M	0	0	0

Table 2: Comparison of fiber lengths (nonzeros per fiber) between the selected mode by SPLATT-ONE and a candidate/non-selected mode

3.2 Number of Representations

CSF based representations are compressed with respect to a certain mode. Therefore, one may need to maintain d distinct representations to exploit the compression at each mode. For example, the third-order tensor shown in Figure 3(a) would require three CSF representations: CSF-0, CSF-1, and CSF-2; they are shown in Figure 3(b). Algorithm 2 (*slice-alg*) shows the computation of MTTKRP on mode-0 to update A using CSF-0, where i indices are at the slice mode. The intermediate modes in this case can either be mode-1 or mode-2. Similarly, to perform MTTKRP on mode-1 to update B, CSF-1 will be used and the j indices will be at the slice mode. A similar analogy can be drawn for mode-2. Both SPLATT and BCSF by default use d representations for an order-d tensor. In both the frameworks, the mode orientation are such that, at mode-n, dimension n will be at the root, and the rest of the modes will be sorted according to their dimension length to achieve the most compressed representation. Algorithm 3 shows the steps to perform MTTKRP at d modes. At Line 3, d number of CSFs are created, and at Line 6, MTTKRP operations are performed at each mode.

COO on the other hand is invariable to mode orientation, and uses a single representation to compute MTTKRP at all d modes. SPLATT library also supports SPLATT-ONE, i.e., using a single representation to perform MTTKRP on d modes [31]. For example, since the tensor in Figure 3(a) is of dimension $3 \times 5 \times 6$, CSF-0 will be selected as the SPLATT-ONE representation – the shortest mode, I, is slice mode, J is the fiber mode, and the longest mode, K, is the nonzero mode. Although in this case, MTTKRP computation at only one mode (e.g., mode-0 in CSF-0) will benefit from the compression at both fiber- and slice-level. At the other modes (e.g., mode-1 and mode-2 in CSF-0), MTTKRP will be computed using *fiber-alg* (Algorithm 7) exploiting compression only at the fiber-level, and using *nonzero-alg* at the nonzero-level (Algorithm 8) with no possible compression. We describe these algorithms in details in the next section. Algorithm 4 shows MTTKRP computation on d modes using a single representation. At Line 1, the dimension for the slice-, fiber-, and nonzero-level is chosen according to the length of the dimensions. MTTKRP on mode-0/mode-1/mode-2 is performed at Line 6/Line 8/Line 10. Using the 2D matrix analogy, computing MTTKRP at mode 1 using CSF-0 is similar to performing SpMTV using a CSR representation instead of CSC.

4 MM-CSF: A MIXED-MODE CSF

SPLATT-ALL and BCSF-ALL iteratively update the matrices which correspond to the mode at its root/slice level. For a third-order

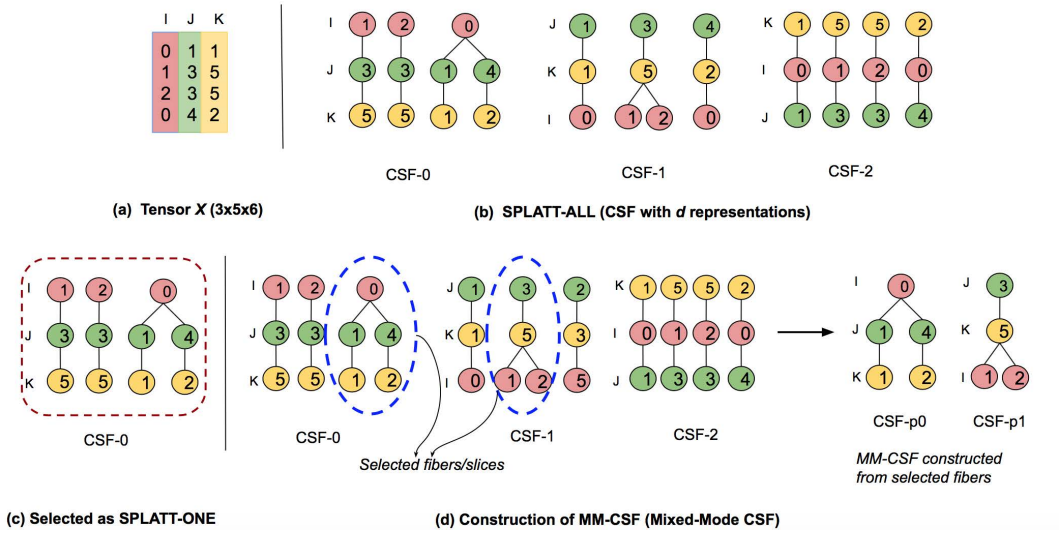


Figure 3: Construction of MM-CSF

Algorithm 4: MTTKRP using one CSF representation on d modes for a third-order tensor ($d = 3$)

Input : $indI[M]$, $indJ[M]$, $indK[M]$, $vals[M]$, $dims[d]$
Output: $\widetilde{A}[I][R]$, $\widetilde{B}[J][R]$, $\widetilde{C}[K][R]$

```
sMode = argmin(dims[0],dims[1],dims[2])   ▷ find the mode to create CSF
CSF = create_CSF (sMode);                 ▷ create CSF
                                          ▷ execute MTTKRP
for mode = 0 to d − 1 do
    if mode == sMode then
        slice-alg (CSF)
    else if mode == fMode then
        fiber-alg (CSF)                   ▷ Refer to Algorithm 7
    else if mode == nMode then
        nonzero-alg (CSF)                 ▷ Refer to Algorithm 8
```

tensor $\mathcal{X}$, matrix A, B and C are updated sequentially using CSF-0, CSF-1, and CSF-2 respectively, when mode-0, mode-1, and mode-2 are at the slice level. In the case of SPLATT-ONE representation, one of these three matrices will use the mode at slice-level to update itself, and the remaining two matrices will update the matrix at fiber-level and nonzero-level respectively. For SPLATT-ONE, the shortest dimension is selected as the slice (root) nodes, and the longest as the nonzero (leaf) nodes; the intermediate modes are sorted accordingly. The selected CSF thus has the highest average slice length, which leads to a high slice compression compared to other CSFs. However, the compression in the next fiber level is not always guaranteed. As shown in Table 2, for tensors fr_s and fr_m, instead of the CSF representation selected as SPLATT-ONE, other CSF representations can have higher average fiber length, and a higher number of fibers with length >10K. This phenomenon can be more severe for higher order tensors. To incorporate multi-level compression, we propose MM-CSF, a Mixed-Mode CSF representation, where heavy fibers and slices are stored at their most suitable modes. For a 2D matrix, this would imply storing the dense rows in CSR format, and the dense columns in CSC format. While performing $y = Ax$, the CSR part can efficiently parallelize across rows. Similarly, while performing $y = A^T x$, the CSC part can efficiently parallelize across columns.

4.1 Partitioning of Nonzeros

The first step in forming MM-CSF is to create d disjoint partitions of the nonzero elements of an order-d tensor $\mathcal{X}$, where each partition will have a different mode orientation. This aims to ensure that each nonzero goes to a partition where it has the potential to achieve the maximum compression. Thus, the resulting tensor partitions should jointly be more compressed, a.k.a. have fewer, but longer fibers. Figure 3(d) shows the construction of MM-CSF.

A nonzero element $\mathcal{X}_{i,j,k}$ of a 3D tensor belongs to three[2] possible fibers: fiber-0 of CSF-0, fiber-1 of CSF-1, and fiber-2 of CSF-2. Let us assume that the lengths of fiber-0, fiber-1, and fiber-2 are l_0, l_1, and l_2, respectively, and $l_0 > l_1 > l_2$. As fiber-0 is the longest fiber, $\mathcal{X}_{i,j,k}$ will be assigned to the partition P_0 orientated in mode-0. When multiple fibers that the nonzero belongs to have same length, to break the tie, we assign the nonzero to the partition where average fiber length is higher. Partitions P_1 and P_2 will be similarly constructed.

In Figure 4, we demonstrate this strategy using fibers $\mathcal{X}_{:,j,k}$ and $\mathcal{X}_{i,j,:}$ in the columns under *static strategy*. All the nonzeros except $\mathcal{X}_{0,1,2}$ are assigned to partition P_0. However, this partitioning scheme creates an additional fiber by splitting fiber $\mathcal{X}_{0,1,:}$ into two partitions. This shortens the length of fiber $\mathcal{X}_{0,1,:}$ but increases the total number of fibers, leading to inefficient compression. The main reason is that the partitioning decision is based on statically pre-calculated fiber lengths at each mode orientation. We propose a dynamic strategy by incrementally updating the fiber length. Continuing with the aforementioned example, once the nonzero $\mathcal{X}_{i,j,k}$ is assigned to partition P_0 since the length of fiber-0 (l_0) is the largest, the lengths of fiber-1 and fiber-2 will be reduced by

[2]For ease of explanation, we use natural mode orientations, i.e., I→J→K for CSF-0, J→K→I for CSF-1, and K→I→J for CSF-2. In reality, $d!$ mode orientations are possible, and the actual mode orientations of each CSF may differ from natural ones.

Algorithm 5: MTTKRP using MM-CSF representation on d modes for a third-order tensor ($d = 3$).

```
   Input : indI[M], indJ[M], indK[M], vals[M], A[I][R], B[J][R], C[K][R]
   Output: Ã[I][R], B̃[J][R], C̃[K][R]
1                                              ▷ select partition for each nonzero
2  for z = 0 to M do
3      i = indI[z]; j = indJ[z]; k = indK[z];
4      if fiberLen(i,j,:) >= max(fiberLen(i,:,k), fiberLen(:,j,k)) then
5          partition = 0
6          fiberLen(i,:,k) -= 1, fiberLen(:,j,k) -= 1
7      else if fiberLen(i,:,k) >= max(fiberLen(i,j,:), fiberLen(:,j,k)) then
8          partition = 1
9          fiberLen(i,j,:) -= 1, fiberLen(:,j,k) -= 1
10     else if fiberLen(:,j,k) >= max(fiberLen(i,:,k), fiberLen(i,j,:)) then
11         partition = 2
12         fiberLen(i,:,k) -= 1, fiberLen(i,j,:) -= 1
13     MM-COO[partition] ∪ = z                 ▷ Add z to the selected partition
14                                             ▷ create CSF for partitions
15 for partition = 0 to noOfPartition do
16     MM-CSF[partition]=create_CSF(MM-COO[partition])
17                                             ▷ execute MTTKRP using MM-CSF
18 for mode = 0 to nModes do
19     for partition = 0 to numPartitions do
20         sMode = MM-CSF[partition].modeOrder[0]
21         fMode = MM-CSF[partition].modeOrder[1]
22         nMode = MM-CSF[partition].modeOrder[2]
23         if mode == sMode then
24             slice-alg (MM-CSF[partition])
25         else if mode == fMode then
26             fiber-alg (MM-CSF[partition])
27         else if mode == nMode then
28             nonzero-alg (MM-CSF[partition])
```

1 to record the processed nonzero. This is illustrated under the Figure 4 *dynamic strategy* columns. The pre-calculated length of fiber $\mathcal{X}_{1,2,:}$ is 3. Under static partitioning strategy, nonzero $\mathcal{X}_{0,1,2}$ will be assigned to partition 1, which creates a fiber in partition 1 with only one nonzero. This is not an efficient compression. On the other hand, fiber $\mathcal{X}_{0,1,:}$ with its length 2 can provide a compression of 2 nonzeros. Compared to static partitioning, we have one less fiber and better compression after dynamic partitioning. Therefore, dynamic partitioning strategy reduces storage requirement, provides better compression, and consequently improves performance.

Algorithm 5 demonstrates the construction of MM-CSF partitions. The construction scheme and the algorithms shown in this paper are for 3D tensors. The extension to arbitrary dimensions is straightforward. The *for* loop (Line 2) determines the partitions of all nonzeros. A nonzero is assigned to the one with the longest fiber length among fiber-0, fiber-1 and fiber-2, shown in Line 4 to 13. After all the nonzeros have been processed, MM-CSF is constructed by creating one CSF representation for each partition, like CSF-p_0, CSF-p_1 in Figure 3(d). To perform MTTKRP on a particular mode, each partition needs to perform its role by using either slice-, fiber-, or nonzero-centric algorithms. For example, to perform MTTKRP on mode-0, CSF-p_0 will use the *(optimized) slice-alg* (Algorithm 6), CSF-p_1 will use the *fiber-alg* (Algorithm 7) and finally, CSF−p_2 will use the *nonzero-alg* (Algorithm 8). These algorithms will be described in Section 5.

nonzero i,j,k	static strategy						dynamic strategy			
	fibers <i,j>		fibers <j,k>		Assig. parti.	tot fbr	fibers <i,j>	fibers <j,k>	Assig. parti.	tot fbr
	pair	len	pair	len			len	len		
5 1 1	<5,1>	3	<1,1>	2	0	1	3	2	0	1
5 1 2	<5,1>	3	<1,2>	3	0	1	3	3	0	1
5 1 3	<5,1>	3	<1,3>	1	0	1	3	1	0	1
4 1 2	<4,1>	3	<1,2>	3	0	2	3	2	0	2
4 1 1	<4,1>	3	<1,1>	2	0	2	3	1	0	2
4 1 5	<4,1>	3	<1,5>	1	0	2	3	1	0	2
0 1 2	<0,1>	2	<1,2>	3	1	3	2	1	0	3
0 1 4	<0,1>	2	<1,4>	1	0	4	2	1	0	3

Figure 4: Adjusting fiber lengths during partitioning

Formats	FLOPS	Reads on matrices	Writes on matrices	Storage of $\mathcal{X}$ in words
COO	3*3MR	3*2MR	3*MR	3*M
SPLATT-ALL	3*(S+2F+2M)R	3*(F+M)R	3*SR	3*2(S+F)+M
SPLATT-ONE	(S+4F+5M)R[1]	3*(F+M)R	(S+F+M)R[2]	2(S+F)+M
MM-CSF	(S_{P0}+4F_{P0}+5M_{P0})R +(S_{P1}+4F_{P1}+5M_{P1})R +(S_{P2}+4F_{P2}+5M_{P2})R [3]	(F_{P0}+M_{P0})R +(F_{P1}+M_{P1})R +(F_{P2}+M_{P2})R	(2F_{P0}+M_{P0})R +(2F_{P1}+M_{P1})R +(2F_{P2}+M_{P2})R	3(F_{P0}+F_{P1}+F_{P2}) +M_{P0}+M_{P1}+M_{P2}

[1] (S+4F+5M)R = slice-mode: (S+2F+2M)R, fiber-mode: 2(F+M)R, nonzero-mode: 3MR
[2] (S+F+M)R = slice-mode: SR, fiber-mode: FR, nonzero-mode: 3MR
[3] S_{PX}: S in partition x, F_{PX}: F in partition x, M_{PX}: M in partition x

Table 3: Theoretical comparison between formats in terms of storage, flop computation, read and write transactions.

5 BALANCED MTTKRP ALGORITHMS USING MM-CSF

The current state-of-the-art, BCSF-ALL [26], only performs slice-alg for MTTKRP on GPUs. We propose balanced fiber-alg and nonzero-alg for GPUs, and furthermore optimize the slice-alg of BCSF-ALL. BCSF-ALL can take advantage of our optimized slice-alg. However, the data structure to support these new algorithms consumes ($3F + M$) space rather than ($2S + 2F + M$) in other formats; usually $F \gg S$. For MM-CSF, the number of fibers, F, can be significantly reduced (e.g., a 2× reduction in tensors fr_s and fr_m) by applying the partitioning scheme described in Section 4. Reduced fiber count not only improves space efficiency, but also improves the performance, as less fibers lead to reduced memory accesses. Table 3 shows a theoretical comparison of the read transactions, write transactions, floating-point operations (flop) etc., among COO, SPLATT-ALL, SPLATT-ONE and MM-CSF.

5.1 MTTKRP on Slice mode

Details of BCSF-ALL. Updating the matrix corresponding to the slice-level incurs a minimum number of global write operations, since the number of slices is traditionally less than the number of fibers or non-zeros for tensors. The BCSF-ALL scheme shown in Algorithm 2 comprises two steps. In the first step, all the nonzeros are reduced to the corresponding fiber (Line 14). The second step involves a subsequent reduction across all the fibers to the parent slice (Line 12). These two steps are collectively termed as *slice-mode operation*. The reductions are performed in registers since they have the lowest access latency in the GPU memory hierarchy. However, the indices of the nonzeros, the fibers and the slices must be read from the global memory, so that corresponding rows can be fetched from the factor matrices (Lines 6,8).

We use an illustrative example to demonstrate the total read-/write computations involved in performing MTTKRP on mode-0

Algorithm 6: opt-slice-alg(): MTTKRP at slice level using MM-CSF for third-order tensors on GPUs ($d = 3$).

Input : *fiberPtr*[F], *sliceInds*[F], *fiberInds*[F], *indK*[M], *vals*[M], dense matrices $B[J][R]$, $C[K][R]$
Output : dense matrix $\widetilde{A}[I][R]$

```
1  fibersGrp = number of fibers in a group
2                                              ▷ parallel across thread-blocks
3  for fiber = 0 to F/fibersGrp do
4      for fiberInGrp = 0 to fibersGrp do
5          localFiber = fiber + fiberInGrp;
6          i = sliceInds[localFiber]
7          j = fiberInds[localFiber]
8                                              ▷ nonzeros-cyclically processed by warps
9          for z = fiberPtr[localFiber] to fiberPtr[localFiber + 1] do
10             k = indK[z]
11                                             ▷ rank-parallel across threads
12             for r = 0 to R do
13                 tmp[r]+ = vals[z] * C[k][r]          ▷ register accumulation
14         for r = 0 to R do
15             tmp_2[r]+ = tmp[r] * B[j][r]             ▷ register accumulation
16                                             ▷ fibers from different slices write back to DRAM
17         if sliceInds[localFiber] != sliceInds[localFiber+1] then
18             for r = 0 to R do
19                 Ã[i][r]+ = tmp_2[r]                  ▷ Atomic writes
20 return Ã
```

(i.e., MTTKRP at Line 2 of Algorithm 1) using the CSF-0 representation. Assume that the tensor $\mathcal{X}$ has only one slice, $\mathcal{X}_{i,:,:}$, and the slice has exactly two fibers, $\mathcal{X}_{i,x,:}$, and $\mathcal{X}_{i,y,:}$. Further, assume that each fiber has only three nonzeros, $\mathcal{X}_{i,x,p}$, $\mathcal{X}_{i,x,q}$ and $\mathcal{X}_{i,x,r}$ for fiber $\mathcal{X}_{i,x,:}$, and $\mathcal{X}_{i,y,p}$, $\mathcal{X}_{i,y,q}$ and $\mathcal{X}_{i,y,r}$ for fiber $\mathcal{X}_{i,y,:}$. In the slice-mode scheme, each nonzero will read the rows with indices p, q, and r from the dense matrix C, use these to perform the computation at Line 12 of Algorithm 2, and accumulate the result in registers (*tmp*). Then, each fiber will read the rows with indices x and y from the dense matrix B, use these to perform the computation at Line 14 of Algorithm 2, and accumulate the result in registers (*tmp_2*). Finally, the slice will perform a read-modify-write to the row i of A. In general, the total number of reads is F (number of fibers) and M (number of nonzeros); the number of read-modify-write is S (number of slices) as shown in Table 3. The flop count varies from $2MR$ to $5MR$.

Improvement over BCSF-ALL. BCSF-ALL splits the exceptionally large slices into sub-slices, and assigns multiple thread-blocks to process each slice. In a similar spirit, multiple smaller slices can be assigned to the same thread-block. To select these assignments, extra pre-processing time and a separate data structure are maintained. The warps within a thread-block process the fibers (sub-fiber) in the respective slice. Each warp reduces the nonzeros of its fiber, and stores the accumulated result in a register. Figure 5(a) pictorially represents the slice-mode algorithm using BCSF-ALL. The scope of parallelism can be increased significantly by offering a finer-grained parallelism. In the proposed format (Figure 5(b)), we assign thread-blocks to fibers instead of slices and warps to nonzeros instead of fibers. One limitation of this scheme is the increased number of global writes. Previously, fibers from the same slice would have accumulated the results in the register, and write back to global memory at the end. To incorporate this compression benefit, we group fibers into smaller chunks and assign one thread-block to

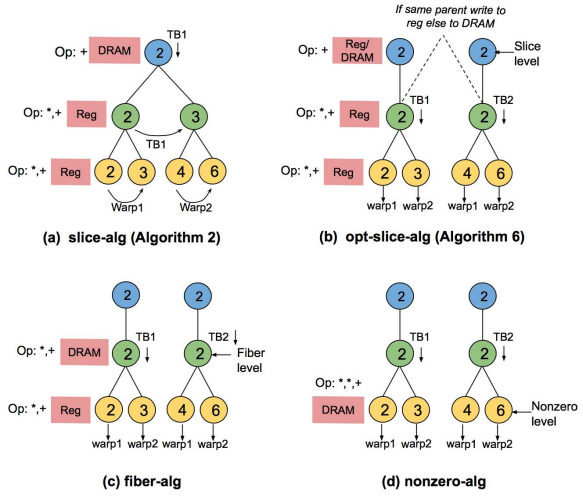


Figure 5: MTTKRP algorithm variants

process a chunk. Fibers at each chunk check whether their parent slice is the same one or not. As long as they share the same slice, it keeps accumulating in the register, otherwise, writes back to global memory. Algorithm 6 demonstrates this optimized version of slice-alg. Line 3 iterates over the chunks and Line 4 iterates over the fibers at each chunk. Line 17 shows the comparison between the parent slices before writing back to global memory. Note that, this scheme might increase the number of atomic writes. Multiple chunks of fibers can share the same slice, and the number of atomics would increase with the number of chunks. To avoid race condition, atomic operations are used to accumulate the values from sub-slices of a slice.

In the original BCSF-ALL data structure, a fiber location was accessed via slice pointers. In matrix terminology, it would imply accessing the start point of the column indices using the row pointer. If we parallelize across the fibers, we need to directly access the fiber indices without fetching the start location from the slice pointers. One expensive way to achieve this is to perform a search to find parent slice of the fiber. Instead, we maintain an array of size F to store the corresponding slice indices in lieu of two arrays (pointers and indices) of size S. Now, each thread-block can directly access the slice and fiber indices. This scheme outperforms BCSF-ALL by increasing parallelism while preserving the compression, as we demonstrate through evaluation in Section 6.

5.2 MTTKRP on Fiber Mode

We now describe the algorithm to compute MTTKRP on mode-1 (i.e., MTTKRP at Line 3 of Algorithm 1) to update matrix B using CSF-0 representation. Algorithm 7 demonstrates the steps of the algorithm. We continue with the illustrative example of Section 5.1. This time, instead of updating matrix A via the slice indices, we will update B via the fiber indices. The indices of interest for B are x and y. Recall that the fibers are $\mathcal{X}_{i,x,:}$ and $\mathcal{X}_{i,y,:}$; these fibers will read the row i from A, and rows p, q, and r from C. Therefore, the number of read-modify-writes (i.e., atomics) is F (number of fibers), instead of S (number of slices) in the previous slice-mode

Algorithm 7: fiber-alg(): MTTKRP at fiber level using MM-CSF for third-order tensors on GPUs ($d = 3$)

Input : $fiberPtr[F]$, $sliceInds[F]$, $fiberInds[F]$, $indK[M]$, $vals[M]$, dense matrices $A[I][R]$, $C[K][R]$
Output: dense matrix $\widetilde{B}[J][R]$

▷ fibers-parallel across thread-blocks
for *fiber = 0* **to** F **do**
  i = $sliceInds$[fiber]
  j = $fiberInds$[fiber]
  ▷ nonzeros-parallel across warps
  for z = $fiberPtr[fiber]$ **to** $fiberPtr[fiber + 1]$ **do**
    k = $indK[z]$
    ▷ rank-parallel across threads
    for $r = 0$ **to** R **do**
      $tmp[r] += vals[z] * C[k][r]$ ▷ accumulation in registers
  for $r = 0$ **to** R **do**
    $\widetilde{B}[j][r] += tmp[r] * A[i][r]$
return $\widetilde{A}$

Algorithm 8: nonzero-alg(): MTTKRP at nonzero level using MM-CSF for third-order tensors on GPUs ($d = 3$)

Input : $fiberPtr[F]$, $sliceInds[F]$, $fiberInds[F]$, $indK[M]$, $vals[M]$, dense matrices $A[I][R]$, $B[J][R]$
Output: dense matrix $\widetilde{C}[K][R]$

▷ fibers-parallel across thread-blocks
for *fiber = 0* **to** F **do**
  i = $sliceInds$[fiber]
  j = $fiberInds$[fiber]
  ▷ nonzeros-parallel across warps
  for z = $fiberPtr[fiber]$ **to** $fiberPtr[fiber + 1]$ **do**
    k = $indK[z]$
    ▷ rank-parallel across threads
    for $r = 0$ **to** R **do**
      $\widetilde{C}[k][r] += vals[z] * B[j][r] * A[i][r]$
return $\widetilde{A}$

algorithm. The total number of reads decreases from $(F + M)R$ of the slice-mode algorithm to $(S + M)R$.

Some tensors inherently show good sparsity structure, for example, a low standard deviation in fiber length and slice length, clustered nonzeros in one mode, etc.; and a totally different structure in another mode, like power law structure. A mode offering a low number of writes with an imbalanced structure might underperform compared to a mode with a higher number of writes and better workload balance. A good example of such case is tensor Darpa. Darpa has 28 million nonzeros with the density of $2.37E-9$. Both CSF-0 and CSF-1 have 22K slices and 281K fibers after splitting the long fibers. But the standard deviation of nonzero per slice is 60K for mode 1 and 26K for mode0. So, in terms of sparsity structure, CSF-0 is more balanced. If we use slice-mode on CSF-1 to compute MTTKRP on mode-1, the total number of reads and writes are 28.7M and 22K respectively. Applying fiber-mode on CSF-0 will result in 28.5M reads and 281K writes. Interestingly, 70% improvement is achieved by using fiber-mode than slice-mode on mode1. We verified our intuition by collecting metrics from NVPROF [1] profiler provided by NVIDIA. The metric *achieved_occupancy*, defined as the ratio of the average active warps per active cycle to the maximum number of warps supported on an SM, increase to 60% with fiber-mode from 40% for slice-mode on NVIDIA P100.

This algorithm exposes an opportunity for finer grained parallelism and reduction in memory latency by allowing similar parallelization strategy like the slice-mode algorithm. Figure 5(c) demonstrates the parallelization techniques. Here, warps can still use registers to reduce the nonzeros, but not to accumulate the sum from fibers. This is because the write locations are now the fiber indices, and fibers from other slices might write to the same location. Hence, we need to use atomic operations to guarantee correctness. But increased parallelism often mitigates the shortcoming of having high atomic operations and achieves comparable performance.

5.3 MTTKRP on nonzero mode

In the nonzero-mode algorithm presented in Algorithm 8, the write locations are fetched from the nonzero locations to update matrix C. Referring back to the illustrative example of Section 5.1, the three nonzeros are $\mathcal{X}_{i,x,p}$, $\mathcal{X}_{i,x,q}$ and $\mathcal{X}_{i,x,r}$ and the update locations are p, q and r. Each nonzero reads row i row of A and, rows x and y of B. We adopt a similar parallelization strategy as fiber-mode. The total number of reads are further reduced to $(S + F)R$, and the number of writes increases to M. Figure 5(d) demonstrates the parallelization of this technique. Just like fiber-mode algorithm, the performance of this algorithm also depends on the sparsity structure of the tensor and the ratio between read and write. For example, Nell-1 dataset shows 11% improvement using nonzero-mode algorithm on mode-2 with CSF-0. Nell-1 has 140M nonzeros with a density of $9.05E-13$. CSF-0 representation for Nell-1 has 2M slices and 17M fibers, and CSF-2 has 25M slices and 113M fibers. If we use CSF-2 to compute mode-2 using slice-mode, the number of reads and writes are 253M and 25M respectively. On the other hand, if we use CSF-0, the number of reads and writes are 157M and 140M.

6 EXPERIMENTAL EVALUATION

6.1 Evaluation Setup

We evaluate the performance of MM-CSF[3] in computing MTTKRP, the computational kernel of a popular CANDECOMP/PARAFAC decomposition (CPD), against five publicly available state-of-the-art frameworks: SPLATT[4] [35], BCSF[5] [26], F-COO[6] [25], and ParTI[7] [22] which provides HiCOO [23] and COO implementations. We used the latest updated code in the SPLATT git repository instead of the release version, as suggested by the authors. Of these frameworks, HiCOO and SPLATT are CPU-based implementations; ParTI-COO[8], BCSF [26], and F-COO [25] are GPU-based frameworks. SPLATT, BCSF, and F-COO each create d representations for an order-d tensor by default. Additionally, SPLATT provides extensions to select the number of representations for a tensor [31]. Therefore, we present comparisons against both d representations (SPLATT-ALL), and single representation (SPLATT-ONE) for SPLATT. Tiling is enabled for SPLATT while collecting the performance data. For a fair comparison, we modify the default

[3] https://github.com/isratnisa/MM-CSF
[4] https://github.com/ShadenSmith/splatt
[5] https://github.com/isratnisa/B-CSF
[6] https://github.com/kobeliu85/mttkrp-gpu
[7] https://github.com/hpcgarage/ParTI
[8] Since the COO CPU of ParTI is significantly outperformed by HiCOO, we only evaluate HiCOO for CPU. All references to ParTI-COO refer to the GPU implementation.

Tensors	order	Dimensions	#Nonzeros	Density
deli	3	$533K \times 17M \times 2M$	140M	6.14E-12
nell1	3	$3M \times 2M \times 25M$	144M	9.05E-13
nell2	3	$12K \times 9K \times 29K$	77M	9.05E-13
flick	3	$320K \times 28M \times 2M$	113M	7.80E-12
fr_m	3	$23M \times 23M \times 166$	99M	1.10E-09
fr_s	3	$39M \times 39M \times 532$	140M	1.73E-10
darpa	3	$22K \times 22K \times 23M$	28M	2.37E-09
nips	4	$2K \times 3K \times 14K \times 17$	3M	3.85E-04
enron	4	$6K \times 6K \times 244K \times 1K$	5M	1.83E-06
ch-cr	4	$6K \times 24 \times 77 \times 32$	54M	1.48E-01
flick	4	$320K \times 28M \times 2M \times 731$	113M	1.07E-14
uber	4	$183 \times 24 \times 1K \times 2K$	3M	5.37E-10

Table 4: Sparse tensor datasets

BCSF-ALL implementation of [26] to support BCSF-ONE (i.e., use a single BCSF representation for all modes). We extend the fiber splitting and binning concept used in BCSF-ALL to implement well-optimized fiber- and nonzero-mode algorithms for BCSF-ONE. HiCOO and ParTI-COO use a single representation HiCOO and COO respectively.

The GPU data is collected on an NVIDIA Volta V100 GPU with 16GB memory. It has 80 SMs and a 6144 KB L2 cache. The CPU data is collected on a Dell PowerEdge R740: a two-socket server with 40-core Intel Xeon 6148. It has 384GB memory with 2.40GHz clock frequency. The CUDA code is compiled with NVCC-9.2, and the CPU code is compiled with GCC-7.3.0. The execution on CPU is parallelized over 40 threads. The results are collected using single-precision data type and tensor rank, R, is set to 32.

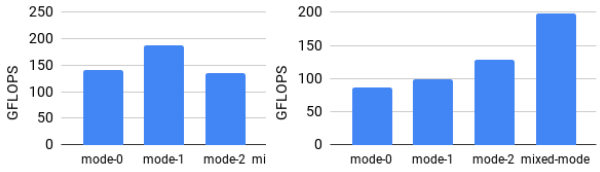
The benchmarks comprise 3D and 4D sparse tensors collected from real-world applications. Datasets like deli (delicious), nell1 and nell2 (Never Ending Language Learner knowledge), flick (Flickr) are from The Formidable Repository of Open Sparse Tensors and Tools, FROSTT [30]. Darpa, fr_m (freebase-music) and fr_s (freebase-sampled) are from the dataset used in HaTen2 [18]. Table 4 lists the tensor order, dimensions, number of nonzeros (#Nonzeros), and the density of these tensors.

6.2 Reduction in Fibers Using MM-CSF

Table 5 shows the reduction in fibers with MM-CSF for 3D tensors, compared to BCSF-ALL and BCSF-ONE representation. To provide better work balance on GPU, long fibers are split into sub-fibers, which increases the number of fibers when compared to SPLATT-ALL. This trend can be observed for some tensors in Table 5, e.g., a 5% increase in fiber count for nell-2. However, compared to BCSF-ALL, MM-CSF achieves an average of 80% reduction in the total fiber count. For most of the benchmarks, we observe a reduction in fiber count with MM-CSF compared to BCSF-ONE as well. For fr_m and fr_s dataset, a reduction of 55% (61M to 27M) and 50% (91M to 45M) respectively in fiber count is observed. The primary reason behind such drastic reduction in fiber count for fr_m and fr_s is the presence of long fibers in mode-2 and mode-0, as noted in Table 2 of Section 4.

	#Fibers (millions)			Reduction %		
	BCSF-ALL	BCSF-ONE	MM-CSF	BCSF-ALL	BCSF-ONE	MM-CSF
deli	122	38	26	0	69	78
nell1	149	18	18	0	88	88
nell2	18	1	1	-5	96	95
flick	55	14	9	-1	75	83
fr_m	183	62	28	0	66	85
fr_s	269	92	45	0	66	83
darpa	29	0.28	0.28	-1	99	99

Table 5: Reduction in number of fibers using MM-CSF compared to other GPU based CSF formats. Reduction (%) is shown compared to SPLATT-ALL.



(a) fr_m **(b) fr_s**

Figure 6: Achieved GFLOPS by assigning all elements to any single-mode vs. using a mixed-mode (MM).

6.3 Impact of Partitioning

We use the partitioning described in Section 4.1 for efficient compression. To evaluate the impact of the implemented partitioning scheme, we compare the results by assigning all elements of the tensor to a single partition against the mixed-mode partitioning, where nonzeros are assigned to multiple partitions. Both variations use the same underlying data structure for a fair comparison. Figure 6a and Figure 6b demonstrate the benefit of partitioning on two representative tensors, fr_s, and fr_m. In both cases, the mixed mode partitioning creates two partitions and assigns nearly 50% nonzeros to each one. In both the tensors, we observe that performing mixed mode partitioning provides a significant performance improvement. On our evaluation over all 3D and 4D tensors, we consistently observe an improvement with partitioning over using an arbitrarily selected single representation.

6.4 Improvement in GPU Occupancy and DRAM Transactions

The kernels for MM-CSF increase GPU occupancy by applying fine-grained parallelism. Table 6 documents the achieved occupancy for 3D tensors, measured via NVPROF. As evident, MM-CSF improves the device occupancy by 45% on average compared to BCSF-ALL. For fr_s dataset, the achieved occupancy improves by almost 2×. Additionally, MM-CSF consistently reduces the global load transactions for all, and DRAM read transactions for majority of the tensors. In cases like deli, where the occupancy improvement is insignificant, the performance improvement can be attributed to a reduction in DRAM reads. However, MM-CSF incurs more DRAM

	GFLOPS		occup. in %		glb. loads in GiB		DRAM in GiB	
	BCSF -ALL	MM-CSF	BCSF -ALL	MM-CSF	BCSF -ALL	MM-CSF	BCSF -ALL	MM-CSF
deli	333	382	73	80	104	86	43	34
nell1	270	285	68	77	112	80	55	55
nell2	607	763	58	76	45	35	4	4
flick	327	435	50	79	76	59	33	27
fr_m	194	235	42	83	97	69	48	50
fr_s	203	228	53	84	140	102	70	73
darpa	209	327	35	52	28	13	12	11

Table 6: Improved occupancy, global loads, and DRAM read transactions using MM-CSF compared to BCSF-ALL (Data collected using NVPROF profiler on V100).

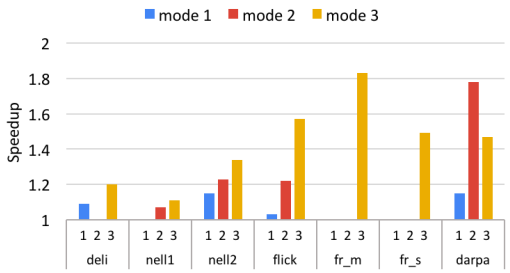


Figure 7: Speedup using MM-CSF compared to BCSF-ALL at d modes on Tesla V100 GPU

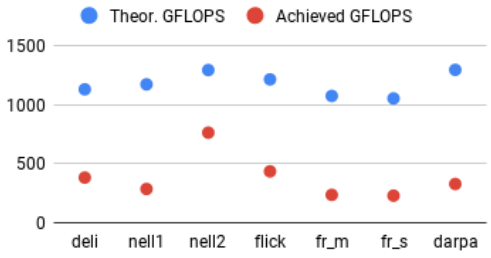


Figure 8: Achieved GFLOPS by MM-CSF compared to theoretically achievable GFLOPS in V100

	MM-CSF	BCSF-ALL	BCSF-ONE	PARTI COO	Hi-COO	SPLATT ALL	SPLATT ONE
deli	106	121	125	149	5,403	5,342	3,284
nell1	145	153	152	235	8,683	2,184	1,969
nell2	29	37	37	71	262	140	94
flick	75	99	92	110	8,374	3,753	1,175
fr_m	122	148	208	225	5,136	5,021	6,897
fr_s	177	199	259	-	7,853	9,344	9,591
darpa	25	39	29	82	1,124	1,078	705
uber	3.72	2.6	4.05	-	298	93	109
nips	2.09	3.2	3.27	-	64	32	18
chicago	3.26	6.4	7.98	-	38	49	10
flickr-4d	130	176	183	-	5,632	9,392	2,076
enron	29	57	38	-	1,085	1,101	1,393

Table 7: Time (ms) to run MTTKRP using MM-CSF and state-of-the-art benchmarks

transactions for fr_m and fr_s. This can be explained by the dimensionalities of these two tensors. For example, fr_m has dimensions $23M \times 23M \times 166$, which implies that A and B matrices are significantly large, C matrix is small enough to be cached entirely in L2 cache of the Volta GPU. We compute the volume of data (in GiB) read from DRAM from the metrics collected by NVPROF. MM-CSF reads 17, 16 and **16** GiB data from DRAM in mode-0, mode-1 and mode-2 respectively, whereas BCSF-ALL reads 14, 14, and **20** GiB data from DRAM. BCSF-ALL for fr_m has 60M fibers in each mode (refer to Table 5). Therefore, while updating A and B in mode-0 and mode-1, the 60M accesses to the fibers come from C, which will likely to be cached in L2. However, while updating C in mode-2, the fiber accesses come from A, resulting in a dramatic increase in DRAM reads. In contrast, with MM-CSF, all A, B, C matrices will potentially be accessed at each mode due to the mixed-mode representation. This results in a consistent DRAM read in all modes, but slightly elevated DRAM reads in the first two modes compared to BCSF-ALL. However, the increase in DRAM transactions with MM-CSF in these two cases is compensated by an overall reduction in global memory transactions and the improved occupancy, resulting in performance enhancement over BCSF-ALL.

6.5 Performance Comparison with BCSF-ALL

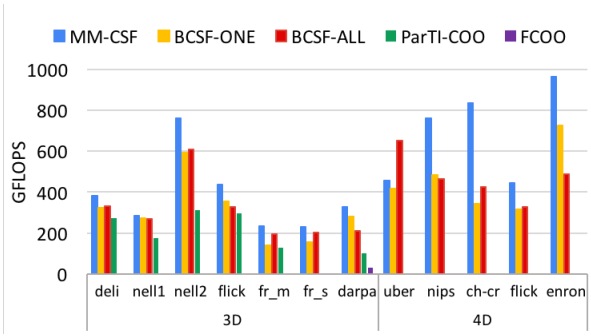
An important metric to demonstrate the utility of MM-CSF is to show that by using it, one can match the performance achieved by the current state-of-the-art frameworks in computing MTTKRP, while simultaneously reducing the space requirement. To the best of our knowledge, BCSF-ALL on GPU offers the maximum performance compared to the other existing frameworks. Figure 7 presents the speedup achieved by MM-CSF compared to BCSF-ALL for 3D tensors. On darpa and fr_m dataset, we outperform BCSF-ALL by a factor of 1.8×. Consistent speedup is observed for the rest of the tensors. For the cases where BCSF-ALL already provides high occupancy, we do not observe any further speedup.

6.6 Performance Model

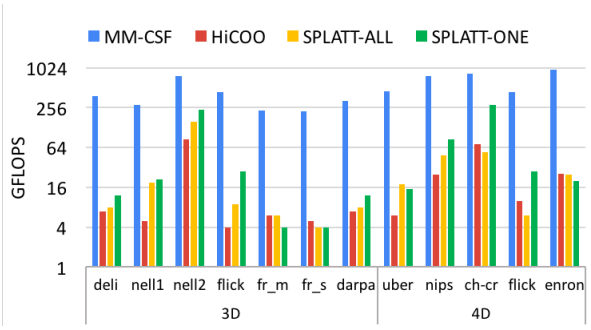
Figure 8 plots the achieved performance versus the theoretically achievable performance in computing MTTKRP on 3D tensors using MM-CSF representation. The theoretically achievable GFLOPS is computed by multiplying the operational intensity (OI) of MTTKRP kernel with the peak bandwidth of V100 GPU device. The gap between realized and theoretical peak performance of GPUs is challenging to bridge, even for compute-bound GEMM kernels. For MTTKRP, the significant gap can primarily be attributed to the poor data locality due to the sparsity of the input tensors. The performance gap is less pronounced for nell-2 dataset. This can be explained by the fact that it is the smallest among all evaluated tensors, with a dimension of $12K \times 9K \times 29K$, and consequently has the highest L2 hit rate (82%).

6.7 Overall Performance

Figure 9 shows the performance achieved by using MM-CSF as the representation to compute MTTKRP, against other state-of-the-art representations/frameworks on both CPU and GPU platforms.



(a) GFLOPS comparison on an NVIDIA V100 GPU



(b) GFLOPS comparison between MM-CSF (on GPU) and CPU-based framework on an Intel 40-core CPU.

Figure 9: Achieved performance of MM-CSF compared to other frameworks.

	SPLATT -ALL	SPLATT -ONE	BCSF -ALL	ParTI -COO	Hi COO	F-COO	MM-CSF
deli	2690	824	2691	1604	2955	3260	838
nell1	3006	697	3010	1643	3062	3341	759
nell2	1012	296	1019	880	250	1789	303
flick	1940	535	1943	1292	1309	2627	539
fr_m	2893	849	2894	1139	1040	2316	700
fr_s	4249	1232	4250	1601	1566	3256	1051
darpa	723	109	726	325	200	662	112

Table 8: Storage comparison in MiB

For a uniform comparison, the floating-point operations of COO-MTTKRP are used as a baseline in computing the GFLOPS for all the frameworks. MM-CSF achieves 510 GFLOPS on average, outperforming BCSF-ONE by a factor of 1.4×, and ParTI-COO by a factor of 2× (Figure 9a). Note that the missing data for F-COO in Figure 9a is due to the failure of successful completion of MTTKRP computation at all modes. For nell2 dataset, MM-CSF achieves the highest performance of 966 GFLOPS.

Figure 9b presents the performance comparison of MM-CSF with CPU-based formats. MM-CSF outperforms SPLATT-ALL by 35× on average. Recently published state-of-the-art COO-based format, HiCOO, is 47× slower than MM-CSF. We also present the execution time of the CPU- and GPU-based benchmarks in Table 7.

6.8 Overall Storage

We present a comparison in space requirements of MM-CSF against state-of-the-art frameworks based on both CSR and COO format families in Table 8. We only use the indices to compute storage of the tensors, as storing the values of each nonzero needs the same space regardless of formats. MM-CSF significantly reduces the space requirement compared to SPLATT-ALL and BCSF-ALL. We now explain the slight increase observed in the storage requirement for MM-CSF compared to SPLATT-ONE. Apart from the fiber splitting for load balancing, MM-CSF also creates an extra data structure of size of F to trace the slice indices along with the fiber indices. Despite these factors that can cause an increase in MM-CSF storage when compared to SPLATT-ONE, we observe an improvement in storage for fr_m and fr_s dataset with MM-CSF in Table 8.

GPU-based F-COO stores d representations of the tensor in COO format. MM-CSF consumes 50% lower space than COO-based frameworks, and 40% lower space than HiCOO. A 3× space reduction is achieved for nell2 and darpa. This is expected as both of these tensors have long fibers and slices, providing good compression that only a CSF based format can exploit.

6.9 Format Conversion to MM-CSF

We compare the pre-processing time involved in constructing MM-CSF vs. BCSF-ALL. While constructing BCSF-ALL, sorting is performed at each mode to identify the nonzeros belonging to the same fiber and same slice. CSF is then constructed on the sorted tensor. Load balancing is achieved via binning [3], where slices with similar lengths are binned together. To construct MM-CSF, we first collect the fiber lengths of $\geq d$ modes, then create p disjoint partitions of nonzeros, and finally, construct CSF for each partition. There is no binning required for MM-CSF. The available BCSF-ALL implementation of [26] uses an unoptimized sort in its preprocessing step. For an unbiased comparison, we replaced it with an optimized version that is used in MM-CSF preprocessing step. Figure 10 presents the normalized time to construct BCSF-ALL and MM-CSF for 3D tensors, including memory copy time for one iteration (i.e., time taken to copy the data from host to GPU device). We observe that on average, MM-CSF incurs merely 15% extra preprocessing overhead over BCSF-ALL. Additionally, MM-CSF consumes significantly less space than BCSF-ALL to store the tensor. Since one might need to perform memory copy with each CPD iteration depending on the size of the tensor, MM-CSF will have a significant advantage over BCSF-ALL in such cases.

6.10 Application speedup

Figure 11 demonstrates the speedup achieved in CPD computation of 3D tensors by using MM-CSF as the storage format in conjunction with the optimized MTTKRP kernels. The reported time is an average of ten iterations. Apart from MTTKRP, all the remaining kernels in the application are invocations of CPU BLAS functions. After each MTTKRP iteration, the updated matrix is copied back to the CPU, where it is used as an input by the BLAS kernels,

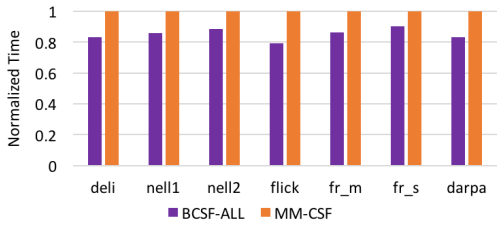


Figure 10: Pre-processing time of BCSF-ALL and MM-CSF

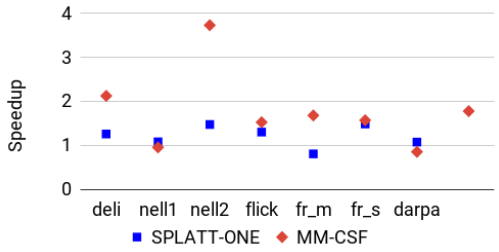


Figure 11: Speedup in CP decomposition using MM-CSF and SPLATT-ONE compared to SPLATT-ALL

followed by a normalization on the column vectors (Line 5 in Algorithm 1). This normalized matrix needs to copied back to GPU to be used in the next MTTKRP computation. Despite the GPU memory copy overhead at each iteration, we outperform SPLATT-ALL from SPLATT by a factor of 1.8× on average. One of our future endeavor involves replacing the CPU BLAS functions with cuBLAS routines to avoid the back-and-forth memory copy time.

7 RELATED WORK

Sparse tensor decompositions and their related operations have attracted attention of researchers to improve their performance and storage. Like matrix factorization [40], [39], [27], tensor factorization is also gaining significant popularity. We briefly discuss prior performance optimization work of MTTKRP operation and CANDECOMP/PARAFAC decomposition (CPD) for sparse tensors.

Tensor Toolbox [5] and Tensorlab [38] packages implement CPD and MTTKRP based on COO format, where an MTTKRP operation is computed as a series of sparse tensor-times-vector. DFacTo [13] performs an MTTKRP by computing multiple sparse matrix-vector multiplication (SpMV) routines which can be computed efficiently through existing high performance libraries. However, the intermediate storage of it could be very large by saving the outputs of SpMV. Smith et al. [29, 35] proposed the CSF storage format, an extension of Compressed Sparse Row (CSR) format for sparse matrices, and optimized the performance and memory access of MTTKRP in the SPLATT library along with the support of different tensor decompositions and completion algorithms [33]. Choi et al. [12] employed two blocking strategies to further optimize MTTKRP using the CSF format. A new Hierarchical COOrdinate (HiCOO) format, derived from the COO format, was recently proposed by Li et al. [23]. HiCOO compresses tensor indices as units of sparse blocks, to save storage and to reduce a sparse tensor algorithm's memory footprint. However, HiCOO does not work well for hypersparse tensors, a.k.a. tensors with extreme low density, sometimes even after reordering [24], thus the other formats like CSF and COO still play important roles. Baskaran et al. proposed multiple optimization techniques to address load imbalance, sparsity, etc. of sparse tensor computation in [6–8].

Some research targeted on other platforms. GigaTensor [19] targets on large-scale sparse tensors by providing a scalable framework using the MapReduce paradigm. Blanco et al. [9] accelerated tensor decompositions using a queuing strategy to exploit the dependency and data reuse using Spark engine on distributed platforms. Kaya et al. [20] scaled CPD on distributed memory systems using message passing interface (MPI), the implementation of which is also based on COO format. Smith et al. [34] improved MTTKRP performance on Intel Xeon Phi Knights Landing manycore processor. A Parallel Tensor Infrastructure (ParTI!) supports COO stored tensors to do MTTKRP on NVIDIA GPUs by parallelizing nonzeros and using atomic operations. Liu et al. [25] proposed a more compressed Flagged COO (F-COO) format uses a fast parallel scan routine on GPUs to reduce write conflicts. However, F-COO closely depends on a particular MTTKRP operation, which affects its flexibility. Nisa et al. [26] optimized MTTKRP performance by proposing load-balanced data structure (BCSF) and parallel strategies, which makes CSF variant MTTKRPP being efficient on GPUs. Phipps et al. [28] leverages the Kokkos framework [14] to optimize MTTKRP on CPUs and GPUs using a single code implementation. Our work further improves MTTKRP by making CSF and BCSF formats more adaptable and efficient to MTTKRP and CPD.

8 CONCLUSION

In recent years, tensors have become mainstream in high-performance computing. Several frameworks and libraries are being developed to optimize operations on sparse tensors. Efficient and compact representations of high-order sparse tensors are crucial on architectures with limited global memory and low energy footprint, like GPUs. In this paper, we devise MM-CSF, a *mixed-mode* storage format for sparse tensors of arbitrary dimensions. Through extensive evaluation on an NVIDIA Volta GPU, we demonstrate the efficacy of MM-CSF in (a) reducing the storage requirement for sparse tensors, and (b) improving the performance of computations like tensor factorizations.

ACKNOWLEDGMENTS

We thank the reviewers for the valuable feedback and the Ohio Supercomputer Center for use of their GPU resources. This material is based upon work supported by the National Science Foundation under Grant No. 1816793 and 1513120. This research is also partially funded by the US Department of Energy, Office for Advanced Scientific Computing (ASCR) under Award No. 66150: "CENATE: The Center for Advanced Technology Evaluation" and the Laboratory Directed Research and Development program at PNNL under contract No. ND8577. Pacific Northwest National Laboratory (PNNL) is a multiprogram national laboratory operated for DOE by Battelle Memorial Institute under Contract DE-AC05-76RL01830. This research was partially supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations âĂŞ the Office of Science and the National Nuclear Security Administration.

REFERENCES

[1] [n. d.]. nvprof-metrics. https://docs.nvidia.com/cuda/profiler-users-guide/index.html. Accessed: 2018-09-30.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[3] Arash Ashari, Naser Sedaghati, John Eisenlohr, Srinivasan Parthasarathy, and P Sadayappan. 2014. Fast sparse matrix-vector multiplication on GPUs for graph applications. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE Press, 781–792.

[4] Brett W Bader, Michael W Berry, and Murray Browne. 2008. Discussion tracking in Enron email using PARAFAC. In *Survey of Text Mining II*. Springer, 147–163.

[5] Brett W. Bader, Tamara G. Kolda, et al. 2015. MATLAB Tensor Toolbox Version 2.6. Available online. http://www.sandia.gov/~tgkolda/TensorToolbox/

[6] M. Baskaran, T. Henretty, B. Pradelle, M. H. Langston, D. Bruns-Smith, J. Ezick, and R. Lethin. 2017. Memory-efficient parallel tensor decompositions. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7. https://doi.org/10.1109/HPEC.2017.8091026

[7] Muthu Baskaran, Benoit Meister, and Richard Lethin. 2014. Low-overhead Load-balanced Scheduling for Sparse Tensor Computations. In *IEEE High Performance Extreme Computing Conference*. Waltham, MA.

[8] Muthu Baskaran, Benoit Meister, Nicolas Vasilache, and Richard Lethin. 2012. Efficient and Scalable Computations with Sparse Tensors. In *IEEE High Performance Extreme Computing Conference*. Waltham, MA.

[9] Zachary Blanco, Bangtian Liu, and Maryam Mehri Dehnavi. 2018. CSTF: Large-Scale Sparse Tensor Factorizations on Distributed Platforms. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP 2018)*. ACM, New York, NY, USA, Article 21, 10 pages. https://doi.org/10.1145/3225058.3225133

[10] Aydin Buluc and John R Gilbert. 2008. On the representation and multiplication of hypersparse matrices. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 1–11.

[11] J Douglas Carroll and Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of âĂIJEckart-YoungâĂİ decomposition. *Psychometrika* 35, 3 (1970), 283–319.

[12] Jee Choi, Xing Liu, Shaden Smith, and Tyler Simon. 2018. Blocking Optimization Techniques for Sparse Tensor Computation. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 568–577.

[13] Joon Hee Choi and S Vishwanathan. 2014. DFacTo: Distributed factorization of tensors. In *Advances in Neural Information Processing Systems*. 1296–1304.

[14] H Carter Edwards, Christian R Trott, and Daniel Sunderland. 2014. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel and Distrib. Comput.* 74, 12 (2014), 3202–3216.

[15] Richard A Harshman. 1970. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis. (1970).

[16] Joyce C Ho, Joydeep Ghosh, Steve R Steinhubl, Walter F Stewart, Joshua C Denny, Bradley A Malin, and Jimeng Sun. 2014. Limestone: High-throughput candidate phenotype generation via tensor factorization. *Journal of biomedical informatics* 52 (2014), 199–211.

[17] Joyce C Ho, Joydeep Ghosh, and Jimeng Sun. 2014. Marble: high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 115–124.

[18] Inah Jeon, Evangelos E Papalexakis, U Kang, and Christos Faloutsos. 2015. Haten2: Billion-scale tensor decompositions. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 1047–1058.

[19] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. 2012. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 316–324.

[20] O. Kaya and B. UÃğar. 2015. Scalable Sparse Tensor Decompositions in Distributed Memory Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 77:1–77:11. https://doi.org/10.1145/2807591.2807624

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

[22] Jiajia Li, Yuchen Ma, and Richard Vuduc. 2017. ParTI!: A Parallel Tensor Infrastructure for Multicore CPU and GPUs. Available from https://github.com/hpcgarage/ParTI.

[23] Jiajia Li, Jimeng Sun, and Richard Vuduc. 2018. HiCOO: Hierarchical Storage of Sparse Tensors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '18)*. ACM, New York, NY, USA.

[24] Jiajia Li, Bora Uçar, Ümit V. Çatalyürek, Jimeng Sun, Kevin Barker, and Richard Vuduc. 2019. Efficient and Effective Sparse Tensor Reordering. In *Proceedings of the ACM International Conference on Supercomputing (ICS '19)*. ACM, New York, NY, USA, 227–237. https://doi.org/10.1145/3330345.3330366

[25] Bangtian Liu, Chengyao Wen, Anand D Sarwate, and Maryam Mehri Dehnavi. 2017. A Unified Optimization Approach for Sparse Tensor Operations on GPUs. In *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 47–57.

[26] Israt Nisa, Jiajia Li, Aravind Sukumaran-Rajam, Richard W. Vuduc, and P. Sadayappan. 2019. Load-Balanced Sparse MTTKRP on GPUs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Vol. abs/1904.03329.

[27] Israt Nisa, Aravind Sukumaran-Rajam, Rakshith Kunchum, and P Sadayappan. 2017. Parallel ccd++ on gpu for matrix factorization. In *Proceedings of the General Purpose GPUs*. ACM, 73–83.

[28] Eric T Phipps and Tamara G Kolda. 2019. Software for Sparse Tensor Decomposition on Emerging Computing Architectures. *SIAM Journal on Scientific Computing* 41, 3 (2019), C269–C290.

[29] Shaden Smith. 2019. *Algorithms for Large-Scale Sparse Tensor Factorization*. Ph.D. Dissertation. University of Minnesota, Minneapolis, MN, USA. http://hdl.handle.net/11299/206375

[30] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. 2017. FROSTT: The Formidable Repository of Open Sparse Tensors and Tools. http://frostt.io/

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

[32] Shaden Smith and George Karypis. 2016. A medium-grained algorithm for sparse tensor factorization. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 902–911.

[33] Shaden Smith, Jongsoo Park, and George Karypis. 2016. An Exploration of Optimization Algorithms for High Performance Tensor Completion. *Proceedings of the 2016 ACM/IEEE conference on Supercomputing* (2016).

[34] Shaden Smith, Jongsoo Park, and George Karypis. 2017. Sparse Tensor Factorization on Many-Core Processors with High-Bandwidth Memory. *31st IEEE International Parallel & Distributed Processing Symposium (IPDPS'17)* (2017).

[35] Shaden Smith, Niranjay Ravindran, Nicholas D Sidiropoulos, and George Karypis. 2015. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 61–70.

[36] Jimeng Sun, Spiros Papadimitriou, and S Yu Philip. 2006. Window-based Tensor Analysis on High-dimensional and Multi-aspect Streams.. In *ICDM*, Vol. 6. 1076–1080.

[37] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 374–383.

[38] Nico Vervliet, Otto Debals, and Lieven De Lathauwer. 2016. Tensorlab 3.0âĂŤNumerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization. In *Signals, Systems and Computers, 2016 50th Asilomar Conference on*. IEEE, 1733–1738.

[39] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. 2014. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems* 41, 3 (2014), 793–819.

[40] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *International conference on algorithmic applications in management*. Springer, 337–348.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

The results for GPU are collected on an NVIDIA Volta V100 GPU with 16GB memory. The results for CPU are collected using a Dell PowerEdge R740 two-socket servers with Intel Xeon 6148. For the GPU codes, NVCC-9.2 compiler is used, and for the CPU code gcc (GCC) 7.3.0 is used with the OpenMP flag. Number of threads is set to 40.

ARTIFACT AVAILABILITY

Software Artifact Availability: All author-created software artifacts are maintained in a public repository under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

List of URLs and/or DOIs where artifacts are available:

`10.5281/zenodo.3379102`
`https://github.com/isratnisa/MM-CSF`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Volta V100 GPU, Intel(R) Xeon(R) Gold 6148 CPU

Operating systems and versions: Red Hat Enterprise Linux Server VERSION=7.5

Compilers and versions: NVCC 9.2, gcc (GCC) 7.3.0

Libraries and versions: boost 1.67, OpenBLAS-3.6, openMP

Key algorithms: MTTKRP, CPD

Input datasets and versions: http://frostt.io

Output from scripts that gathers execution environment information.

```
LMOD_FAMILY_COMPILER_VERSION=18.0.3
MKLROOT=/opt/intel/18.0.3/compilers_and_libraries_20⌋
↪ 18/linux/mkl
MANPATH=/opt/mvapich2/intel/18.0/2.3/share/man:/opt/⌋
↪ intel/18.0.3/itac_latest/man:/opt/intel/18.0.3/d⌋
↪ ocumentation_2018/en/debugger/gdb-igfx/man:/opt/⌋
↪ intel/18.0.3/inspector_2018/man:/opt/intel/18.0.⌋
↪ 3/compilers_and_libraries_2018/linux/man/common:⌋
↪ /opt/torque/share/man:/opt/moab/man:/apps/lmod/l⌋
↪ mod/share/man:/usr/share/man/overrides:/usr/shar⌋
↪ e/man:/usr/local/share/man:/opt/ibutils/share/ma⌋
↪ n:/opt/ddn/ime/share/man:/opt/puppetlabs/puppet/⌋
↪ share/man:/opt/intel/18.0.3/vtune_amplifier/man:⌋
↪ /opt/intel/18.0.3/advisor/man
__LMOD_REF_COUNT_FPATH=/opt/intel/18.0.3/compilers_a⌋
↪ nd_libraries_2018/linux/mkl/include:1
XALT_ETC_DIR=/apps/xalt/etc
_ModuleTable003_=ZE9yZGVyIl09NCxwcm9wVD17fSxbInN0YWN⌋
↪ rRGVwdGgiXT0xLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2V⌋
↪ yTmFtZSJdPSJtdmFwaWNoMiIsfSx4YWx0PXtbImZuIl09Ii9⌋
↪ hcHBzL2xtb2RmaWxlcy9Db3JlL3hhbHQvbGF0ZXN0Lmx1YSI⌋
↪ sWyJmdWxsTmFtZSJdPSJ4YWx0L2xhdGVzdCIsWyJsb2FkT3J⌋
↪ kZXIiXT0xLHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTEsWyJ⌋
↪ zdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09InhhbHQ⌋
↪ iLH0sfSxtcGF0aEE9eyIvYXBwcy9sbW9kZmlsZXMvTVBJL2l⌋
↪ udGVsLzE4LjAvbXZhcGljaDIvMi4zIiwiL2FwcHMvbG1vZGZ⌋
↪ pbGVzL0NvbXBpbGVyL2ludGVsLzE4LjAiLCIvYXBwcy9sbW9⌋
↪ kZmlsZXMvTGludXgiLCIvYXBwcy9sbW9kZmlsZXMvQ29y
PBS_VERSION=TORQUE-6.1.2
MPI_FFLAGS=-I/opt/mvapich2/intel/18.0/2.3/include
IPPROOT=/opt/intel/18.0.3/compilers_and_libraries_20⌋
↪ 18/linux/ipp
MPICH_HOME=/opt/mvapich2/intel/18.0/2.3
MPI_F90FLAGS=-I/opt/mvapich2/intel/18.0/2.3/include
F90=ifort
SHELL=/bin/bash
TERM=xterm-256color
__LMOD_REF_COUNT_MODULEPATH=/apps/lmodfiles/MPI/inte⌋
↪ l/18.0/mvapich2/2.3:1;/apps/lmodfiles/Compiler/i⌋
↪ ntel/18.0:1;/apps/lmodfiles/Linux:1;/apps/lmodfi⌋
↪ les/Core:1;/apps/lmod/lmod/modulefiles/Core:1
HISTSIZE=1000
PBS_JOBNAME=STDIN
MODULEPATH_ROOT=/apps/lmodfiles
TMPDIR=/tmp/pbstmp.472725
LMOD_SYSTEM_DEFAULT_MODULES=modules
XALT_EXECUTABLE_TRACKING=yes
LMOD_PACKAGE_PATH=/apps/lmodfiles/site
```

```
LIBRARY_PATH=/opt/intel/18.0.3/compilers_and_librari
↪  es_2018/linux/linux/lib/intel64_lin:/opt/intel/1
↪  8.0.3/compilers_and_libraries_2018/linux/daal/li
↪  b/intel64_lin:/opt/intel/18.0.3/compilers_and_li
↪  braries_2018/linux/ipp/lib/intel64_lin:/opt/inte
↪  l/18.0.3/compilers_and_libraries_2018/linux/mkl/
↪  lib/intel64_lin:/opt/intel/18.0.3/compilers_and_
↪  libraries_2018/linux/tbb/lib/intel64_lin/gcc4.4
__LMOD_REF_COUNT_COMPILER_PATH=/apps/xalt/xalt/bin:1
LD_PRELOAD=/apps/xalt/xalt/lib64/libxalt_init.so
PBS_ENVIRONMENT=PBS_INTERACTIVE
LMOD_PKG=/apps/lmod/lmod
FPATH=/opt/intel/18.0.3/compilers_and_libraries_2018
↪  /linux/mkl/include
COMPILER=intel
QTDIR=/usr/lib64/qt-3.3
IMEDIR=
LMOD_VERSION=7.8
QTINC=/usr/lib64/qt-3.3/include
PBS_HOME=/var/spool/torque

CC=icc
__LMOD_REF_COUNT_LOADEDMODULES=xalt/latest:1;cxx17/7
↪  .3.0:1;intel/18.0.3:1;mvapich2/2.3:1;modules/au2
↪  018:1
QT_GRAPHICSSYSTEM_CHECKED=1
INTEL_DIR=/opt/intel/18.0.3
USER=USER
PBS_TASKNUM=1
MV2_CPU_BINDING_POLICY=hybrid
COMPILER_MINOR=0
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:p
↪  i=40;38;5;11:so=38;5;13:do=38;5;5:bd=48;5;232;38
↪  ;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=0
↪  5;48;5;232;38;5;15:su=48;5;196;38;5;15:sg=48;5;1
↪  1;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38;5;1
↪  6:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;
↪  34:*.tar=38;5;9:*.tgz=38;5;9:*.arc=38;5;9:*.arj=
↪  38;5;9:*.taz=38;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.
↪  lzh=38;5;9:*.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5
↪  ;9:*.tzo=38;5;9:*.t7z=38;5;9:*.zip=38;5;9:*.z=38
↪  ;5;9:*.Z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38
↪  ;5;9:*.lz=38;5;9:*.lzo=38;5;9:*.xz=38;5;9:*.bz2=
↪  38;5;9:*.bz=38;5;9:*.tbz=38;5;9:*.tbz2=38;5;9:*.
↪  tz=38;5;9:*.deb=38;5;9:*.rpm=38;5;9:*.jar=38;5;9
↪  :*.war=38;5;9:*.ear=38;5;9:*.sar=38;5;9:*.rar=38
↪  ;5;9:*.alz=38;5;9:*.ace=38;5;9:*.zoo=38;5;9:*.cp
↪  io=38;5;9:*.7z=38;5;9:*.rz=38;5;9:*.cab=38;5;9:*
↪  .jpg=38;5;13:*.jpeg=38;5;13:*.gif=38;5;13:*.bmp=
↪  38;5;13:*.pbm=38;5;13:*.pgm=38;5;13:*.ppm=38;5;1
↪  3:*.tga=38;5;13:*.xbm=38;5;13:*.xpm=38;5;13:*.ti
↪  f=38;5;13:*.tiff=38;5;13:*.png=38;5;13:*.svg=38;
↪  5;13:*.svgz=38;5;13:*.mng=38;5;13:*.pcx=38;5;13:
↪  *.mov=38;5;13:*.mpg=38;5;13:*.mpeg=38;5;13:*.m2v
↪  =38;5;13:*.mkv=38;5;13:*.webm=38;5;13:*.ogm=38;5
↪  ;13:*.mp4=38;5;13:*.m4v=38;5;13:*.mp4v=38;5;13:*
↪  .vob=38;5;13:*.qt=38;5;13:*.nuv=38;5;13:*.wmv=38
↪  ;5;13:*.asf=38;5;13:*.rm=38;5;13:*.rmvb=38;5;13:
↪  *.flc=38;5;13:*.avi=38;5;13:*.fli=38;5;13:*.flv=
↪  38;5;13:*.gl=38;5;13:*.dl=38;5;13:*.xcf=38;5;13:
↪  *.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=
↪  38;5;13:*.axv=38;5;13:*.anx=38;5;13:*.ogv=38;5;1
↪  3:*.ogx=38;5;13:*.aac=38;5;45:*.au=38;5;45:*.fla
↪  c=38;5;45:*.mid=38;5;45:*.midi=38;5;45:*.mka=38;
↪  5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg=38;5;45:*
↪  .ra=38;5;45:*.wav=38;5;45:*.axa=38;5;45:*.oga=38
↪  ;5;45:*.spx=38;5;45:*.xspf=38;5;45:
LMOD_sys=Linux
LD_LIBRARY_PATH=/opt/mvapich2/intel/18.0/2.3/lib:/ap
↪  ps/gnu/7.3.0/lib64:/apps/gnu/7.3.0/lib:/opt/inte
↪  l/18.0.3/debugger_2018/libipt/intel64/lib:/opt/i
↪  ntel/18.0.3/compilers_and_libraries_2018/linux/l
↪  ib/intel64_lin:/opt/intel/18.0.3/compilers_and_l
↪  ibraries_2018/linux/daal/lib/intel64_lin:/opt/in
↪  tel/18.0.3/compilers_and_libraries_2018/linux/ip
↪  p/lib/intel64_lin:/opt/intel/18.0.3/compilers_an
↪  d_libraries_2018/linux/mkl/lib/intel64_lin:/opt/
↪  intel/18.0.3/compilers_and_libraries_2018/linux/
↪  tbb/lib/intel64_lin/gcc4.4:/opt/torque/lib64:/op
↪  t/torque/lib::

XXX_COMPILER_MAJOR=18
XXX_FAMILY_MPI=mvapich2
PBSCOREDUMP=""
CPATH=/opt/intel/18.0.3/compilers_and_libraries_2018
↪  /linux/mkl/include:/opt/intel/18.0.3/compilers_a
↪  nd_libraries_2018/linux/tbb/include
```

```
_ModuleTable004_=ZSIsIi9hcHBzL2xtb2QvbG1vZC9tb2R1bGV
↪ maWxlcy9Db3JlIix9LFsic3lzdGVtQmFzZU1QQVRIIl09Ii9
↪ hcHBzL2xtb2RmaWxlcy9MaW51eDovYXBwcy9sbW9kZmlsZXM
↪ vQ29yZTovYXBwcy9sbW9kL2xtb2QvbW9kdWxlZmlsZXMvQ29
↪ yZSIsfQ==
PBS_WALLTIME=10740
XXX_CXX=icpc
__LMOD_REF_COUNT__LMFILES_=/apps/lmodfiles/Core/xalt
↪ /latest.lua:1;/apps/lmodfiles/Compiler/intel/18.
↪ 0/cxx17/7.3.0.lua:1;/apps/lmodfiles/Core/intel/1
↪ 8.0.3.lua:1;/apps/lmodfiles/Compiler/intel/18.0/
↪ mvapich2/2.3.lua:1;/apps/lmodfiles/Core/modules/
↪ au2018.lua:1
PBS_MOMPORT=15003
PBS_GPUFILE=/var/spool/torque/aux//472725. gpu
LMOD_SITE_NAME=XXX
MPIEXEC_COMM=pmi
PBS_O_QUEUE=batch
LMOD_PREPEND_BLOCK=normal
LMOD_FAMILY_MPI_VERSION=2.3
MPI_CFLAGS=-I/opt/mvapich2/intel/18.0/2.3/include
MPI_CXXFLAGS=-I/opt/mvapich2/intel/18.0/2.3/include
MPI_LIBS=-L/opt/mvapich2/intel/18.0/2.3/lib -lmpich
↪ -libverbs -lpthread
NLSPATH=/opt/intel/18.0.3/debugger_2018/gdb/intel64/
↪ share/locale/%l_%t/%N:/opt/intel/18.0.3/compiler
↪ s_and_libraries_2018/linux/lib/intel64_lin/local
↪ e/%l_%t/%N:/opt/intel/18.0.3/compilers_and_libra
↪ ries_2018/linux/mkl/lib/intel64_lin/locale/%l_%t
↪ /%N
PATH=/apps/xalt/xalt/bin:/opt/mvapich2/intel/18.0/2.
↪ 3/bin:/apps/gnu/7.3.0/bin:/opt/intel/18.0.3/itac
↪ _latest/bin:/opt/intel/18.0.3/advisor/bin64:/opt
↪ /intel/18.0.3/vtune_amplifier/bin64:/opt/intel/1
↪ 8.0.3/inspector_2018/bin64:/opt/intel/18.0.3/com
↪ pilers_and_libraries_2018/linux/bin/intel64:/app
↪ s/software_usage:/opt/torque/bin:/usr/lib64/qt-3
↪ .3/bin:/opt/XXX/bin:/opt/moab/bin:/bin:/usr/bin:
↪ /usr/local/bin:/usr/local/sbin:/usr/sbin:/opt/ib
↪ utils/bin:/opt/ddn/ime/bin:/opt/puppetlabs/bin
PBS_O_LOGNAME=USER
MAIL=/var/spool/mail/USER
__LMOD_REF_COUNT_NLSPATH=/opt/intel/18.0.3/debugger_
↪ 2018/gdb/intel64/share/locale/%l_%t/%N:1;/opt/in
↪ tel/18.0.3/compilers_and_libraries_2018/linux/li
↪ b/intel64_lin/locale/%l_%t/%N:1;/opt/intel/18.0.
↪ 3/compilers_and_libraries_2018/linux/mkl/lib/int
↪ el64_lin/locale/%l_%t/%N:1
_ModuleTable001_=X01vZHVsZVRhYmxlXz17WyJNVHZlcnNpb24
↪ iXT0zLFsiY19yZWJ1aWxkVGltZSJdPTg2NDAwLFsiY19zaG9
↪ ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9e30sZmFtaWx5PXtbImN
↪ vbXBpbGVyIl09ImludGVsIixbIm1waSJdPSJtdmFwaWNoMiI
↪ sfSxtVD17Y3h4MTc9e1siZm4iXT0iL2FwcHMvbG1vZGZpbGV
↪ zL0NvbXBpbGVyL2ludGVsLzE4LjAvY3h4MTcvNy4zLjAubHV
↪ hIixbImZ1bGxOYW1lIl09ImN4eDE3LzcuMy4wIixbImxvYWR
↪ PcmRlciJdPTIscHJvcFQ9e30sWyJzdGFja0RlcHRoIl09Mix
↪ bInN0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5hbWUiXT0iY3h
↪ 4MTciLH0saW50ZWw9e1siZm4iXT0iL2FwcHMvbG1vZGZpbGV
↪ zL0NvcmUvaW50ZWwvMTguMC4zLmx1YSIsWyJmdWxsTmFt
PBS_O_LANG=en_US.UTF-8
PBS_JOBCOOKIE=8DD3E1D6A361CA28088AEFA814243CC6
LMOD_SETTARG_CMD=:
XXX_FAMILY_COMPILER=intel
XXX_MVAPICH2_DIR=/opt/mvapich2/intel/18.0/2.3
TBBROOT=/opt/intel/18.0.3/compilers_and_libraries_20
↪ 18/linux/tbb

PDSH_RCMD_TYPE=ssh
_LMFILES_=/apps/lmodfiles/Core/xalt/latest.lua:/apps
↪ /lmodfiles/Compiler/intel/18.0/cxx17/7.3.0.lua:/
↪ apps/lmodfiles/Core/intel/18.0.3.lua:/apps/lmodf
↪ iles/Compiler/intel/18.0/mvapich2/2.3.lua:/apps/
↪ lmodfiles/Core/modules/au2018.lua
LANG=en_US.UTF-8
PBS_NODENUM=0
MODULEPATH=/apps/lmodfiles/MPI/intel/18.0/mvapich2/2
↪ .3:/apps/lmodfiles/Compiler/intel/18.0:/apps/lmo
↪ dfiles/Linux:/apps/lmodfiles/Core:/apps/lmod/lmo
↪ d/modulefiles/Core
MOABHOMEDIR=/var/spool/moab
XXX_FAMILY_COMPILER_VERSION=18.0.3
PBS_NUM_NODES=1
KDEDIRS=/usr

LOADEDMODULES=xalt/latest:cxx17/7.3.0:intel/18.0.3:m
↪ vapich2/2.3:modules/au2018
_ModuleTable_Sz_=4
PBS_O_SHELL=/bin/bash
XXX_MVAPICH2_LIB=/opt/mvapich2/intel/18.0/2.3/lib
LMOD_CMD=/apps/lmod/lmod/libexec/lmod
XXX_MPI_CC=mpicc
PBS_JOBID=472725.
LMOD_AVAIL_STYLE=system
DAALROOT=/opt/intel/18.0.3/compilers_and_libraries_2
↪ 018/linux/daal
HISTCONTROL=ignoredups
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
ENVIRONMENT=BATCH
INTEL_PYTHONHOME=/opt/intel/18.0.3/debugger_2018/pyt
↪ hon/intel64
XXX_F77=ifort

SHLVL=2
XXX_FC=ifort
```

```
XXX_MPI_CXX=mpic++
__LMOD_REF_COUNT_PATH=/apps/xalt/xalt/bin:1;/opt/mva
↪  pich2/intel/18.0/2.3/bin:1;/apps/gnu/7.3.0/bin:1
↪  ;/opt/intel/18.0.3/itac_latest/bin:1;/opt/intel/
↪  18.0.3/advisor/bin64:1;/opt/intel/18.0.3/vtune_a
↪  mplifier/bin64:1;/opt/intel/18.0.3/inspector_201
↪  8/bin64:1;/opt/intel/18.0.3/compilers_and_librar
↪  ies_2018/linux/bin/intel64:1;/apps/software_usag
↪  e:1;/opt/torque/bin:1;/usr/lib64/qt-3.3/bin:1;/o
↪  pt/XXX/bin:1;/opt/moab/bin:1;/bin:1;/usr/bin:1;/
↪  usr/local/bin:1;/usr/local/sbin:1;/usr/sbin:1;/o
↪  pt/ibutils/bin:1;/opt/ddn/ime/bin:1;/opt/puppetl
↪  abs/bin:1

__LMOD_REF_COUNT_CPATH=/opt/intel/18.0.3/compilers_a
↪  nd_libraries_2018/linux/mkl/include:1;/opt/intel
↪  /18.0.3/compilers_and_libraries_2018/linux/tbb/i
↪  nclude:1

_ModuleTable002_=ZSJdPSJpbnRlbC8xOC4wLjMiLFsibG9hZE9
↪  yZGVyIl09Myxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0xLFs
↪  ic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJpbnR
↪  lbCIsfSxtb2R1bGVzPXtbImZuIl09Ii9hcHBzL2xtb2RmaWx
↪  lcy9Db3JlL21vZHVsZXMvYXUyMDE4Lmx1YSIsWyJmdWxsTmF
↪  tZSJdPSJtb2R1bGVzL2F1MjAxOCIsWyJsb2FkT3JkZXIiXT0
↪  1LHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTAsWyJzdGF0dXM
↪  iXT0iYWN0aXZlIixbInVzZXJOYW1lIl09Im1vZHVsZXMiLH0
↪  sbXZhcGljaDI9e1siZm4iXT0iL2FwcHMvbG1vZGZpbGVzL0N
↪  vbXBpbGVyL2ludGVsLzE4LjAvbXZhcGljaDIvMi4zLmx1YSI
↪  sWyJmdWxsTmFtZSJdPSJtdmFwaWNoMi8yLjMiLFsibG9h
__LMOD_REF_COUNT_INCLUDE=/opt/intel/18.0.3/compilers
↪  _and_libraries_2018/linux/daal/include:1;/opt/in
↪  tel/18.0.3/compilers_and_libraries_2018/linux/ip
↪  p/include:1;/opt/intel/18.0.3/compilers_and_libr
↪  aries_2018/linux/mkl/include:1
XALT_SCALAR_AND_SPSR_SAMPLING=yes
PBS_VNODENUM=0
BASH_ENV=/apps/lmod/lmod/init/bash
XXX_MPI_FC=mpifort
LOGNAME=USER
LMOD_arch=x86_64
MV2_IBA_HCA=mlx5_0
CVS_RSH=ssh
QTLIB=/usr/lib64/qt-3.3/lib
XXX_FAMILY_MPI_VERSION=2.3
PBS_QUEUE=serial
PDSH_SSH_ARGS_APPEND=-oStrictHostKeyChecking=no
↪  -oUserKnownHostsFile=/dev/null -oLogLevel=ERROR
MODULESHOME=/apps/lmod/lmod
__LMOD_REF_COUNT_LIBRARY_PATH=/opt/intel/18.0.3/comp
↪  ilers_and_libraries_2018/linux/linux/lib/intel64
↪  _lin:1;/opt/intel/18.0.3/compilers_and_libraries
↪  _2018/linux/daal/lib/intel64_lin:1;/opt/intel/18
↪  .0.3/compilers_and_libraries_2018/linux/ipp/lib/
↪  intel64_lin:1;/opt/intel/18.0.3/compilers_and_li
↪  braries_2018/linux/mkl/lib/intel64_lin:1;/opt/in
↪  tel/18.0.3/compilers_and_libraries_2018/linux/tb
↪  b/lib/intel64_lin/gcc4.4:1
PBS_O_MAIL=/var/spool/mail/USER
PBS_O_SUBMIT_FILTER=/usr/local/sbin/torque_submitfil
↪  ter

LESSOPEN=||/usr/bin/lesspipe.sh %s
LMOD_SETTARG_FULL_SUPPORT=no
COMPILER_PATH=/apps/xalt/xalt/bin
__LMOD_REF_COUNT_LD_LIBRARY_PATH=/opt/mvapich2/intel
↪  /18.0/2.3/lib:1;/apps/gnu/7.3.0/lib64:1;/apps/gn
↪  u/7.3.0/lib:1;/opt/intel/18.0.3/debugger_2018/li
↪  bipt/intel64/lib:1;/opt/intel/18.0.3/compilers_a
↪  nd_libraries_2018/linux/lib/intel64_lin:1;/opt/i
↪  ntel/18.0.3/compilers_and_libraries_2018/linux/d
↪  aal/lib/intel64_lin:1;/opt/intel/18.0.3/compiler
↪  s_and_libraries_2018/linux/ipp/lib/intel64_lin:1
↪  ;/opt/intel/18.0.3/compilers_and_libraries_2018/
↪  linux/mkl/lib/intel64_lin:1;/opt/intel/18.0.3/co
↪  mpilers_and_libraries_2018/linux/tbb/lib/intel64
↪  _lin/gcc4.4:1;/opt/torque/lib64:1;/opt/torque/li
↪  b:1
MV2_USE_RDMA_CM=0
__Init_Default_Modules=1
LMOD_FULL_SETTARG_SUPPORT=no
__LMOD_REF_COUNT_LD_PRELOAD=/apps/xalt/xalt/lib64/li
↪  bxalt_init.so:1
LMOD_FAMILY_COMPILER=intel
PBS_NP=28

PBS_NUM_PPN=28
QT_PLUGIN_PATH=/usr/lib64/kde4/plugins:/usr/lib/kde4
↪  /plugins
LMOD_CACHED_LOADS=yes
LMOD_DIR=/apps/lmod/lmod/libexec
INCLUDE=/opt/intel/18.0.3/compilers_and_libraries_20
↪  18/linux/daal/include:/opt/intel/18.0.3/compiler
↪  s_and_libraries_2018/linux/ipp/include:/opt/inte
↪  l/18.0.3/compilers_and_libraries_2018/linux/mkl/
↪  include
```

```
__LMOD_REF_COUNT_MANPATH=/opt/mvapich2/intel/18.0/2.
↪  3/share/man:1;/opt/intel/18.0.3/itac_latest/man:
↪  1;/opt/intel/18.0.3/documentation_2018/en/debugg
↪  er/gdb-igfx/man:1;/opt/intel/18.0.3/inspector_20
↪  18/man:1;/opt/intel/18.0.3/compilers_and_librari
↪  es_2018/linux/man/common:1;/opt/torque/share/man
↪  :1;/opt/moab/man:1;/apps/lmod/lmod/share/man:1;/
↪  usr/share/man/overrides:1;/usr/share/man:1;/usr/
↪  local/share/man:1;/opt/ibutils/share/man:1;/opt/
↪  ddn/ime/share/man:2;/opt/puppetlabs/puppet/share
↪  /man:1;/opt/intel/18.0.3/vtune_amplifier/man:1;/
↪  opt/intel/18.0.3/advisor/man:1
XXX_XALT_DIR=/apps/xalt/xalt
__LMOD_Priority_PATH=/apps/xalt/xalt/bin:-100
=
LMOD_COLORIZE=yes
LMOD_FAMILY_MPI=mvapich2
PBS_O_PATH=/apps/xalt/xalt/bin:/opt/mvapich2/intel/1
↪  8.0/2.3/bin:/apps/gnu/7.3.0/bin:/opt/intel/18.0.
↪  3/itac_latest/bin:/opt/intel/18.0.3/advisor/bin6
↪  4:/opt/intel/18.0.3/vtune_amplifier/bin64:/opt/i
↪  ntel/18.0.3/inspector_2018/bin64:/opt/intel/18.0
↪  .3/compilers_and_libraries_2018/linux/bin/intel6
↪  4:/apps/software_usage:/opt/torque/bin:/usr/lib6
↪  4/qt-3.3/bin:/opt/XXX/bin:/opt/moab/bin:/usr/loc
↪  al/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/i
↪  butils/bin:/opt/ddn/ime/bin:/opt/puppetlabs/bin
BASH_FUNC_module()=() {  eval $($LMOD_CMD bash "$@")
↪  && eval $(${LMOD_SETTARG_CMD:-:} -s sh)
}
BASH_FUNC_ml()=() {  eval $($LMOD_DIR/ml_cmd "$@")
}
_=/bin/env
++ lsb_release -a
LSB Version:        :core-4.1-amd64:core-4.1-noarch:
↪  cxx-4.1-amd64:cxx-4.1-noarch:desktop-4.1-amd64:d
↪  esktop-4.1-noarch:languages-4.1-amd64:languages-
↪  4.1-noarch:printing-4.1-amd64:printing-4.1-noarch
Distributor ID:        RedHatEnterpriseServer
Description:        Red Hat Enterprise Linux Server
↪  release 7.5 (Maipo)
Release:        7.5
Codename:        Maipo
++ uname -a
++ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    20
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6148 CPU
↪  @ 2.40GHz
Stepping:              4
CPU MHz:               2400.000
BogoMIPS:              4800.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              28160K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22
↪  ,24,26,28,30,32,34,36,38
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23
↪  ,25,27,29,31,33,35,37,39
Flags:                 fpu vme de pse tsc msr pae mce
↪  cx8 apic sep mtrr pge mca cmov pat pse36 clflush
↪  dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
↪  pdpe1gb rdtscp lm constant_tsc art arch_perfmon
↪  pebs bts rep_good nopl xtopology nonstop_tsc
↪  aperfmperf eagerfpu pni pclmulqdq dtes64 monitor
↪  ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
↪  pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt
↪  tsc_deadline_timer aes xsave avx f16c rdrand
↪  lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3
↪  intel_pt ssbd mba ibrs ibpb stibp tpr_shadow vnmi
↪  flexpriority ept vpid fsgsbase tsc_adjust bmi1
↪  hle avx2 smep bmi2 erms invpcid rtm cqm mpx rdt_a
↪  avx512f avx512dq rdseed adx smap clflushopt clwb
↪  avx512cd avx512bw avx512vl xsaveopt xsavec
↪  xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total
↪  cqm_mbm_local dtherm ida arat pln pts pku ospke
↪  spec_ctrl intel_stibp flush_l1d
++ cat /proc/meminfo
MemTotal:       394800924 kB
MemFree:        379435860 kB
MemAvailable:   381470572 kB
Buffers:          208484 kB
Cached:          2844152 kB
SwapCached:            0 kB
Active:          2252220 kB
Inactive:        2267268 kB
Active(anon):    1543664 kB
Inactive(anon):   517340 kB
Active(file):     708556 kB
Inactive(file):  1749928 kB
Unevictable:     4292464 kB
Mlocked:         4292464 kB
SwapTotal:      50331644 kB
SwapFree:       50331644 kB
Dirty:              236 kB
Writeback:            0 kB
AnonPages:       5759336 kB
Mapped:           564880 kB
Shmem:            551848 kB
Slab:            1804840 kB
```

```
SReclaimable:     714332 kB
SUnreclaim:      1090508 kB
KernelStack:       30432 kB
PageTables:        26380 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    247732104 kB
Committed_AS:     7106576 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      1810776 kB
VmallocChunk:   34156619752 kB
HardwareCorrupted:     0 kB
AnonHugePages:   5318656 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:     4712256 kB
DirectMap2M:    110243840 kB
DirectMap1G:    288358400 kB
++ inxi -F -c0
./collet_env.sh: line 15: inxi: command not found
++ lsblk -a
NAME             MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                8:0    0 931.5G  0 disk
└─sda1             8:1    0 931.5G  0 part
  ├─vg0-lv_state 253:0    0    16G  0 lvm
  ↪  /var/lib/stateless/state
  ├─vg0-lv_rw    253:1    0    16G  0 lvm
  ↪  /var/lib/stateless/writable
  ├─vg0-lv_swap  253:2    0    48G  0 lvm  [SWAP]
  └─vg0-lv_tmp   253:3    0 851.5G  0 lvm  /tmp
++ lsscsi -s
[2:0:0:0]    disk    ATA      HUS722T1TALA600  MU02
↪  /dev/sda   1.00TB
++ module list
+++ /apps/lmod/lmod/libexec/lmod bash list

Currently Loaded Modules:
  1) xalt/latest   2) cxx17/7.3.0   3) intel/18.0.3
  ↪  4) mvapich2/2.3   5) modules/au2018
```

```
++ eval 'MODULEPATH=/apps/lmodfiles/MPI/intel/18.0/m⌋
↪  vapich2/2.3:/apps/lmodfiles/Compiler/intel/18.0:⌋
↪  /apps/lmodfiles/Linux:/apps/lmodfiles/Core:/apps⌋
↪  /lmod/lmod/modulefiles/Core;' export
↪  'MODULEPATH;' '_ModuleTable001_=X01vZHVsZVRhYmxl⌋
↪  Xz17WyJNVHZlcnNpb24iXT0zLFsiY19yZWJ1aWxkVGltZSJd⌋
↪  PTg2NDAwLFsiY19zaG9ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9⌋
↪  e30sZmFtaWx5PXtbImNvbXBpbGVyIl09ImludGVsIixbIm1w⌋
↪  aSJdPSJtdmFwaWNoMiIsfSxtVD17Y3h4MTc9e1siZm4iXT0i⌋
↪  L2FwcHMvbG1vZGZpbGVzL0NvbXBpbGVyL2ludGVsLzE4LjAv⌋
↪  Y3h4MTcvNy4zLjAubHVhIixbImZ1bGxOYW1lIl09ImN4eDE3⌋
↪  LzcuMy4wIixbImxvYWRPcmRlciJdPTIscHJvcFQ9e30sWyJz⌋
↪  dGFja0RlcHRoIl09MixbInN0YXR1cyJdPSJhY3RpdmUiLFsi⌋
↪  dXNlck5hbWUiXT0iY3h4MTciLH0saW50ZWw9e1siZm4iXT0i⌋
↪  L2FwcHMvbG1vZGZpbGVzL0NvcmUvaW50ZWwvMTguMC4zLmx1⌋
↪  YSIsWyJmdWxsTmFt;' export '_ModuleTable001_;'
↪  '_ModuleTable002_=ZSJdPSJpbnRlbC8xOC4wLjMiLFsibG⌋
↪  9hZE9yZGVyIl09Myxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT⌋
↪  0xLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPS⌋
↪  JpbnRlbCIsfSxtb2R1bGVzPXtbImZuIl09Ii9hcHBzL2xtb2⌋
↪  RmaWxlcy9Db3JlL21vZHVsZXMvYXUyMDE4Lmx1YSIsWyJmdW⌋
↪  xsTmFtZSJdPSJtb2R1bGVzL2F1MjAxOCIsWyJsb2FkT3JkZX⌋
↪  IiXT01LHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTAsWyJzdG⌋
↪  F0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09Im1vZHVsZX⌋
↪  MiLH0sbXZhcGljaDI9e1siZm4iXT0iL2FwcHMvbG1vZGZpbG⌋
↪  VzL0NvbXBpbGVyL2ludGVsLzE4LjAvbXZhcGljaDIvMi4zLm⌋
↪  x1YSIsWyJmdWxsTmFtZSJdPSJtdmFwaWNoMi8yLjMiLFsibG⌋
↪  9h;' export '_ModuleTable002_;'
↪  '_ModuleTable003_=ZE9yZGVyIl09NCxwcm9wVD17fSxbIn⌋
↪  N0YWNrRGVwdGgiXT0xLFsic3RhdHVzIl09ImFjdGl2ZSIsWy⌋
↪  J1c2VyTmFtZSJdPSJtdmFwaWNoMiIsfSx4YWx0PXtbImZuIl⌋
↪  09Ii9hcHBzL2xtb2RmaWxlcy9Db3JlL3hhbHQvbGF0ZXN0Lm⌋
↪  x1YSIsWyJmdWxsTmFtZSJdPSJ4YWx0L2xhdGVzdCIsWyJsb2⌋
↪  FkT3JkZXIiXT0xLHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPT⌋
↪  EsWyJzdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09In⌋
↪  hhbHQiLH0sfSxtcGF0aEE9eyIvYXBwcy9sbW9kZmlsZXMvTV⌋
↪  BJL2ludGVsLzE4LjAvbXZhcGljaDIvMi4zIiwiL2FwcHMvbG⌋
↪  1vZGZpbGVzL0NvbXBpbGVyL2ludGVsLzE4LjAiLCIvYXBwcy⌋
↪  9sbW9kZmlsZXMvTGludXgiLCIvYXBwcy9sbW9kZmlsZXMvQ2⌋
↪  9y;' export '_ModuleTable003_;'
↪  '_ModuleTable004_=ZSIsIi9hcHBzL2xtb2QvbG1vZC9tb2⌋
↪  R1bGVmaWxlcy9Db3JlIix9LFsic3lzdGVtQmFzZU1QQVRIIl⌋
↪  09Ii9hcHBzL2xtb2RmaWxlcy9MaW51eDovYXBwcy9sbW9kZm⌋
↪  lsZXMvQ29yZTovYXBwcy9sbW9kL2xtb2QvbW9kdWxlZmlsZX⌋
↪  MvQ29yZSIsfQ==;' export '_ModuleTable004_;'
↪  '_ModuleTable_Sz_=4;' export '_ModuleTable_Sz_;'
+++ MODULEPATH=/apps/lmodfiles/MPI/intel/18.0/mvapic⌋
↪  h2/2.3:/apps/lmodfiles/Compiler/intel/18.0:/apps⌋
↪  /lmodfiles/Linux:/apps/lmodfiles/Core:/apps/lmod⌋
↪  /lmod/modulefiles/Core
+++ export MODULEPATH
```

```
+++ _ModuleTable001_=X01vZHVsZVRhYmxlXz17WyJNVHZlcnN
↪  pb24iXT0zLFsiY19yZWJ1aWxkVGltZSJdPTg2NDAwLFsiY19
↪  zaG9ydFRpbWUiXT1mYWxzZSxkZXB0aFQ9e30sZmFtaWx5PXt
↪  bImNvbXBpbGVyIl09ImludGVsIixbIm1waSJdPSJtdmFwaWN
↪  oMiIsfSxtVD17Y3h4MTc9e1siZm4iXT0iL2FwcHMvbG1vZGZ
↪  pbGVzL0NvbXBpbGVyL2ludGVsLzE4LjAvY3h4MTcvNy4zLjA
↪  ubHVhIixbImZ1bGxOYW1lIl09ImN4eDE3LzcuMy4wIixbImx
↪  vYWRPcmRlciJdPTIscHJvcFQ9e30sWyJzdGFja0RlcHRoIl0
↪  9MixbInN0YXR1cyJdPSJhY3RpdmUiLFsidXNlck5hbWUiXT0
↪  iY3h4MTciLH0saW50ZWw9e1siZm4iXT0iL2FwcHMvbG1vZGZ
↪  pbGVzL0NvcmUvaW50ZWwvMTguMC4zLmx1YSIsWyJmdWxsTmFt
+++ export _ModuleTable001_
+++ _ModuleTable002_=ZSJdPSJpbnRlbC8xOC4wLjMiLFsibG9
↪  hZE9yZGVyIl09Myxwcm9wVD17fSxbInN0YWNrRGVwdGgiXT0
↪  xLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ1c2VyTmFtZSJdPSJ
↪  pbnRlbCIsfSxtb2R1bGVzPXtbImZuIl09Ii9hcHBzL2xtb2R
↪  maWxlcy9Db3JlL21vZHVsZXMvYXUyMDE4Lmx1YSIsWyJmdWx
↪  sTmFtZSJdPSJtb2R1bGVzL2F1MjAxOCIsWyJsb2FkT3JkZXI
↪  iXT01LHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTAsWyJzdGF
↪  0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09Im1vZHVsZXM
↪  iLH0sbXZhcGljaDI9e1siZm4iXT0iL2FwcHMvbG1vZGZpbGV
↪  zL0NvbXBpbGVyL2ludGVsLzE4LjAvbXZhcGljaDIvMi4zLmx
↪  1YSIsWyJmdWxsTmFtZSJdPSJtdmFwaWNoMi8yLjMiLFsibG9h
+++ export _ModuleTable002_
+++ _ModuleTable003_=ZE9yZGVyIl09NCxwcm9wVD17fSxbInN
↪  0YWNrRGVwdGgiXT0xLFsic3RhdHVzIl09ImFjdGl2ZSIsWyJ
↪  1c2VyTmFtZSJdPSJtdmFwaWNoMiIsfSx4YWx0PXtbImZuIl0
↪  9Ii9hcHBzL2xtb2RmaWxlcy9Db3JlL3hhbHQvbGF0ZXN0Lmx
↪  1YSIsWyJmdWxsTmFtZSJdPSJ4YWx0L2xhdGVzdCIsWyJsb2F
↪  kT3JkZXIiXT0xLHByb3BUPXt9LFsic3RhY2tEZXB0aCJdPTE
↪  sWyJzdGF0dXMiXT0iYWN0aXZlIixbInVzZXJOYW1lIl09Inh
↪  hbHQiLH0sfSxtcGF0aEE9eyIvYXBwcy9sbW9kZmlsZXMvTVB
↪  JL2ludGVsLzE4LjAvbXZhcGljaDIvMi4zIiwiL2FwcHMvbG1
↪  vZGZpbGVzL0NvbXBpbGVyL2ludGVsLzE4LjAiLCIvYXBwcy9
↪  sbW9kZmlsZXMvTGludXgiLCIvYXBwcy9sbW9kZmlsZXMvQ29y
+++ export _ModuleTable003_
+++ _ModuleTable004_=ZSIsIi9hcHBzL2xtb2QvbG1vZC9tb2R
↪  1bGVmaWxlcy9Db3JlIix9LFsic3lzdGVtQmFzZU1QQVRIIl0
↪  9Ii9hcHBzL2xtb2RmaWxlcy9MaW51eDovYXBwcy9sbW9kZml
↪  sZXMvQ29yZTovYXBwcy9sbW9kL2xtb2QvbW9kdWxlZmlsZXM
↪  vQ29yZSIsfQ==
+++ export _ModuleTable004_
+++ _ModuleTable_Sz_=4
+++ export _ModuleTable_Sz_
+++ : -s sh
++ eval
++ nvidia-smi
Thu Apr 11 06:33:06 2019
+-----------------------------------------------------
↪  ---------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79
↪  CUDA Version: 10.0     |
|-------------------------------+----------------------
↪  ---+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A
↪  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage
↪  | GPU-Util  Compute M. |
|===============================+======================
↪  ===+======================|
|   0  Tesla V100-PCIE...  On   | 00000000:3B:00.0 Off
↪  |                  Off |
| N/A   31C    P0    43W / 250W |     11MiB / 16130MiB
↪  |      0%   E. Process |
+-------------------------------+----------------------
↪  ---+----------------------+

+-----------------------------------------------------
↪  ---------------------------+
| Processes:
↪             GPU Memory |
|  GPU       PID   Type   Process name
↪              Usage      |
|=============================================================
↪  ===========================|
|  No running processes found
↪                            |
+-----------------------------------------------------
↪  ---------------------------+
++ cat
++ lshw -short -quiet -sanitize
WARNING: you should run this program as super-user.
H/W path         Device  Class          Description
=============================================
                         system         Computer
/0                       bus            Motherboard
/0/0                     memory         382GiB System
↪  memory
/0/1                     processor      Intel(R)
↪  Xeon(R) Gold 6148 CPU @ 2.40GHz
/0/3                     processor      Intel(R)
↪  Xeon(R) Gold 6148 CPU @ 2.40GHz
/0/100                   bridge         Sky Lake-E DMI3
↪  Registers
/0/100/5                 generic        Sky Lake-E
↪  MM/Vt-d Configuration Registers
/0/100/5.2               generic        Intel
↪  Corporation
/0/100/5.4               generic        Intel
↪  Corporation
/0/100/8                 generic        Sky Lake-E Ubox
↪  Registers
/0/100/8.1               generic        Sky Lake-E Ubox
↪  Registers
/0/100/8.2               generic        Sky Lake-E Ubox
↪  Registers
/0/100/11                generic        Intel
↪  Corporation
/0/100/11.5              storage        Lewisburg
↪  SSATA Controller [AHCI mode]
```

```
/0/100/14               bus            Lewisburg USB
↪ 3.0 xHCI Controller
/0/100/14.2             generic        Lewisburg
↪ Thermal Subsystem
/0/100/16               communication  Lewisburg
↪ CSME: HECI #1
/0/100/16.1             communication  Lewisburg
↪ CSME: HECI #2
/0/100/16.4             communication  Lewisburg
↪ CSME: HECI #3
/0/100/17               storage        Lewisburg SATA
↪ Controller [AHCI mode]
/0/100/1c               bridge         Lewisburg PCI
↪ Express Root Port #1
/0/100/1c/0      em3    network        I350 Gigabit
↪ Network Connection
/0/100/1c/0.1    em4    network        I350 Gigabit
↪ Network Connection
/0/100/1c.4             bridge         Lewisburg PCI
↪ Express Root Port #5
/0/100/1c.4/0           bridge         PLDA
/0/100/1c.4/0/0         display        Integrated
↪ Matrox G200eW3 Graphics Controller
/0/100/1f               bridge         Lewisburg LPC
↪ Controller
/0/100/1f.2             memory         Memory
↪ controller
/0/100/1f.4             bus            Lewisburg SMBus
/0/100/1f.5             bus            Lewisburg SPI
↪ Controller
/0/2                    bridge         Sky Lake-E PCI
↪ Express Root Port C
/0/2/0          em1     network        Ethernet
↪ Controller X710 for 10GbE SFP+
/0/2/0.1        em2     network        Ethernet
↪ Controller X710 for 10GbE SFP+
/0/4                    generic        Intel Corporation
/0/6                    generic        Sky Lake-E RAS
↪ Configuration Registers
/0/7                    generic        Intel Corporation
/0/9                    generic        Sky Lake-E CHA
↪ Registers
/0/a                    generic        Sky Lake-E CHA
↪ Registers
/0/b                    generic        Sky Lake-E CHA
↪ Registers
/0/c                    generic        Sky Lake-E CHA
↪ Registers
/0/d                    generic        Sky Lake-E CHA
↪ Registers
/0/e                    generic        Sky Lake-E CHA
↪ Registers
/0/f                    generic        Sky Lake-E CHA
↪ Registers
/0/10                   generic        Sky Lake-E CHA
↪ Registers
/0/11                   generic        Sky Lake-E CHA
↪ Registers
/0/12                   generic        Sky Lake-E CHA
↪ Registers
/0/13                   generic        Sky Lake-E CHA
↪ Registers
/0/14                   generic        Sky Lake-E CHA
↪ Registers
/0/15                   generic        Sky Lake-E CHA
↪ Registers
/0/16                   generic        Sky Lake-E CHA
↪ Registers
/0/17                   generic        Sky Lake-E CHA
↪ Registers
/0/18                   generic        Sky Lake-E CHA
↪ Registers
/0/19                   generic        Sky Lake-E CHA
↪ Registers
/0/1a                   generic        Sky Lake-E CHA
↪ Registers
/0/1b                   generic        Sky Lake-E CHA
↪ Registers
/0/1c                   generic        Sky Lake-E CHA
↪ Registers
/0/1d                   generic        Sky Lake-E CHA
↪ Registers
/0/1e                   generic        Sky Lake-E CHA
↪ Registers
/0/1f                   generic        Sky Lake-E CHA
↪ Registers
/0/20                   generic        Sky Lake-E CHA
↪ Registers
/0/21                   generic        Sky Lake-E CHA
↪ Registers
/0/22                   generic        Sky Lake-E CHA
↪ Registers
/0/23                   generic        Sky Lake-E CHA
↪ Registers
/0/24                   generic        Sky Lake-E CHA
↪ Registers
/0/25                   generic        Sky Lake-E CHA
↪ Registers
/0/26                   generic        Sky Lake-E CHA
↪ Registers
/0/27                   generic        Sky Lake-E CHA
↪ Registers
/0/28                   generic        Sky Lake-E CHA
↪ Registers
/0/29                   generic        Sky Lake-E CHA
↪ Registers
/0/2a                   generic        Sky Lake-E CHA
↪ Registers
```

```
/0/2b                  generic        Sky Lake-E CHA
↪ Registers
/0/2c                  generic        Sky Lake-E CHA
↪ Registers
/0/2d                  generic        Sky Lake-E CHA
↪ Registers
/0/2e                  generic        Sky Lake-E CHA
↪ Registers
/0/2f                  generic        Sky Lake-E CHA
↪ Registers
/0/30                  generic        Sky Lake-E CHA
↪ Registers
/0/31                  generic        Sky Lake-E CHA
↪ Registers
/0/32                  generic        Sky Lake-E CHA
↪ Registers
/0/33                  generic        Sky Lake-E CHA
↪ Registers
/0/34                  generic        Sky Lake-E CHA
↪ Registers
/0/35                  generic        Sky Lake-E CHA
↪ Registers
/0/36                  generic        Sky Lake-E CHA
↪ Registers
/0/37                  generic        Sky Lake-E CHA
↪ Registers
/0/38                  generic        Sky Lake-E CHA
↪ Registers
/0/39                  generic        Sky Lake-E CHA
↪ Registers
/0/3a                  generic        Sky Lake-E CHA
↪ Registers
/0/3b                  generic        Sky Lake-E CHA
↪ Registers
/0/3c                  generic        Sky Lake-E CHA
↪ Registers
/0/3d                  generic        Sky Lake-E CHA
↪ Registers
/0/3e                  generic        Sky Lake-E CHA
↪ Registers
/0/3f                  generic        Sky Lake-E CHA
↪ Registers
/0/40                  generic        Sky Lake-E CHA
↪ Registers
/0/41                  generic        Sky Lake-E CHA
↪ Registers
/0/42                  generic        Sky Lake-E CHA
↪ Registers
/0/43                  generic        Sky Lake-E CHA
↪ Registers
/0/44                  generic        Sky Lake-E CHA
↪ Registers
/0/45                  generic        Sky Lake-E PCU
↪ Registers
/0/46                  generic        Sky Lake-E PCU
↪ Registers
/0/47                  generic        Sky Lake-E PCU
↪ Registers
/0/48                  generic        Sky Lake-E PCU
↪ Registers
/0/49                  generic        Sky Lake-E PCU
↪ Registers
/0/4a                  generic        Sky Lake-E PCU
↪ Registers
/0/4b                  generic        Sky Lake-E PCU
↪ Registers
/0/101                 bridge         Sky Lake-E PCI
↪ Express Root Port A
/0/101/0               display        GV100GL [Tesla
↪ V100 PCIe]
/0/4c                generic      Intel Corporation
/0/4d                  generic        Sky Lake-E RAS
↪ Configuration Registers
/0/4e                generic      Intel Corporation
/0/4f                generic      Intel Corporation
/0/50                generic      Intel Corporation
/0/51                generic      Intel Corporation
/0/52                generic      Intel Corporation
/0/53                generic      Intel Corporation
/0/54                generic      Intel Corporation
/0/55                generic      Intel Corporation
/0/56                generic      Intel Corporation
/0/57                generic      Intel Corporation
/0/58                generic      Intel Corporation
/0/59                generic      Intel Corporation
/0/5a                generic      Intel Corporation
/0/5b                generic      Intel Corporation
/0/5c                generic      Intel Corporation
/0/5d                generic      Intel Corporation
/0/5e                generic      Intel Corporation
/0/5f                generic      Intel Corporation
/0/60                generic      Intel Corporation
/0/61                generic      Intel Corporation
/0/62                generic      Intel Corporation
/0/63                generic      Intel Corporation
/0/64                generic      Intel Corporation
/0/65                generic      Intel Corporation
/0/66                generic      Intel Corporation
/0/67                generic      Intel Corporation
/0/68                generic      Intel Corporation
/0/69                generic      Intel Corporation
/0/6a                  generic        Sky Lake-E RAS
↪ Configuration Registers
/0/6b                generic      Intel Corporation
/0/6c                generic      Intel Corporation
/0/6d                generic      Intel Corporation
/0/6e                generic      Intel Corporation
/0/6f                generic      Intel Corporation
/0/70                generic      Intel Corporation
/0/71                generic      Intel Corporation
```

```
/0/72                   generic       Sky Lake-E
↪  M3KTI Registers
/0/73                   generic       Sky Lake-E
↪  M3KTI Registers
/0/74                   generic       Sky Lake-E
↪  M3KTI Registers
/0/75                   generic       Sky Lake-E
↪  M3KTI Registers
/0/76                   generic       Sky Lake-E
↪  M3KTI Registers
/0/77                   generic       Sky Lake-E
↪  M2PCI Registers
/0/78                   generic       Sky Lake-E
↪  M2PCI Registers
/0/79                   generic       Sky Lake-E
↪  M2PCI Registers
/0/7a                   generic       Sky Lake-E
↪  M2PCI Registers
/0/7b                   generic       Sky Lake-E
↪  MM/Vt-d Configuration Registers
/0/7c                 generic       Intel Corporation
/0/7d                 generic       Intel Corporation
/0/7e                  generic       Sky Lake-E Ubox
↪  Registers
/0/7f                  generic       Sky Lake-E Ubox
↪  Registers
/0/80                  generic       Sky Lake-E Ubox
↪  Registers
/0/102                  bridge        Sky Lake-E PCI
↪  Express Root Port A
/0/102/0        ib0     network       MT27800 Family
↪  [ConnectX-5]
/0/102/0.1      ib1     network       MT27800 Family
↪  [ConnectX-5]
/0/81                 generic       Intel Corporation
/0/82                  generic       Sky Lake-E RAS
↪  Configuration Registers
/0/83                 generic       Intel Corporation
/0/84                  generic       Sky Lake-E CHA
↪  Registers
/0/8.1                 generic       Sky Lake-E CHA
↪  Registers
/0/8.2                 generic       Sky Lake-E CHA
↪  Registers
/0/85                  generic       Sky Lake-E CHA
↪  Registers
/0/86                  generic       Sky Lake-E CHA
↪  Registers
/0/87                  generic       Sky Lake-E CHA
↪  Registers
/0/88                  generic       Sky Lake-E CHA
↪  Registers
/0/89                  generic       Sky Lake-E CHA
↪  Registers
/0/8a                  generic       Sky Lake-E CHA
↪  Registers
/0/8b                  generic       Sky Lake-E CHA
↪  Registers
/0/8c                  generic       Sky Lake-E CHA
↪  Registers
/0/8d                  generic       Sky Lake-E CHA
↪  Registers
/0/8e                  generic       Sky Lake-E CHA
↪  Registers
/0/8f                  generic       Sky Lake-E CHA
↪  Registers
/0/90                  generic       Sky Lake-E CHA
↪  Registers
/0/91                  generic       Sky Lake-E CHA
↪  Registers
/0/92                  generic       Sky Lake-E CHA
↪  Registers
/0/93                  generic       Sky Lake-E CHA
↪  Registers
/0/94                  generic       Sky Lake-E CHA
↪  Registers
/0/95                  generic       Sky Lake-E CHA
↪  Registers
/0/96                  generic       Sky Lake-E CHA
↪  Registers
/0/97                  generic       Sky Lake-E CHA
↪  Registers
/0/98                  generic       Sky Lake-E CHA
↪  Registers
/0/99                  generic       Sky Lake-E CHA
↪  Registers
/0/9a                  generic       Sky Lake-E CHA
↪  Registers
/0/9b                  generic       Sky Lake-E CHA
↪  Registers
/0/9c                  generic       Sky Lake-E CHA
↪  Registers
/0/9d                  generic       Sky Lake-E CHA
↪  Registers
/0/9e                  generic       Sky Lake-E CHA
↪  Registers
/0/9f                  generic       Sky Lake-E CHA
↪  Registers
/0/a0                  generic       Sky Lake-E CHA
↪  Registers
/0/a1                  generic       Sky Lake-E CHA
↪  Registers
/0/a2                  generic       Sky Lake-E CHA
↪  Registers
/0/a3                  generic       Sky Lake-E CHA
↪  Registers
/0/a4                  generic       Sky Lake-E CHA
↪  Registers
```

```
/0/a5                 generic        Sky Lake-E CHA
↪  Registers
/0/a6                 generic        Sky Lake-E CHA
↪  Registers
/0/a7                 generic        Sky Lake-E CHA
↪  Registers
/0/a8                 generic        Sky Lake-E CHA
↪  Registers
/0/a9                 generic        Sky Lake-E CHA
↪  Registers
/0/aa                 generic        Sky Lake-E CHA
↪  Registers
/0/ab                 generic        Sky Lake-E CHA
↪  Registers
/0/ac                 generic        Sky Lake-E CHA
↪  Registers
/0/ad                 generic        Sky Lake-E CHA
↪  Registers
/0/ae                 generic        Sky Lake-E CHA
↪  Registers
/0/af                 generic        Sky Lake-E CHA
↪  Registers
/0/b0                 generic        Sky Lake-E CHA
↪  Registers
/0/b1                 generic        Sky Lake-E CHA
↪  Registers
/0/b2                 generic        Sky Lake-E CHA
↪  Registers
/0/b3                 generic        Sky Lake-E CHA
↪  Registers
/0/b4                 generic        Sky Lake-E CHA
↪  Registers
/0/b5                 generic        Sky Lake-E CHA
↪  Registers
/0/b6                 generic        Sky Lake-E CHA
↪  Registers
/0/b7                 generic        Sky Lake-E CHA
↪  Registers
/0/b8                 generic        Sky Lake-E CHA
↪  Registers
/0/b9                 generic        Sky Lake-E CHA
↪  Registers
/0/ba                 generic        Sky Lake-E CHA
↪  Registers
/0/bb                 generic        Sky Lake-E CHA
↪  Registers
/0/bc                 generic        Sky Lake-E CHA
↪  Registers
/0/bd                 generic        Sky Lake-E CHA
↪  Registers
/0/be                 generic        Sky Lake-E PCU
↪  Registers
/0/bf                 generic        Sky Lake-E PCU
↪  Registers
/0/c0                 generic        Sky Lake-E PCU
↪  Registers
/0/c1                 generic        Sky Lake-E PCU
↪  Registers
/0/c2                 generic        Sky Lake-E PCU
↪  Registers
/0/c3                 generic        Sky Lake-E PCU
↪  Registers
/0/c4                 generic        Sky Lake-E PCU
↪  Registers
/0/103                bridge         Sky Lake-E PCI
↪  Express Root Port A
/0/103/0        ib2   network        MT27800 Family
↪  [ConnectX-5]
/0/103/0.1      ib3   network        MT27800 Family
↪  [ConnectX-5]
/0/c5                 generic        Intel Corporation
/0/c6                 generic        Sky Lake-E RAS
↪  Configuration Registers
/0/c7                 generic        Intel Corporation
/0/8                  generic        Intel Corporation
/0/c8                 generic        Intel Corporation
/0/c9                 generic        Intel Corporation
/0/ca                 generic        Intel Corporation
/0/cb                 generic        Intel Corporation
/0/cc                 generic        Intel Corporation
/0/cd                 generic        Intel Corporation
/0/ce                 generic        Intel Corporation
/0/cf                 generic        Intel Corporation
/0/d0                 generic        Intel Corporation
/0/d1                 generic        Intel Corporation
/0/d2                 generic        Intel Corporation
/0/d3                 generic        Intel Corporation
/0/d4                 generic        Intel Corporation
/0/d5                 generic        Intel Corporation
/0/d6                 generic        Intel Corporation
/0/d7                 generic        Intel Corporation
/0/d8                 generic        Intel Corporation
/0/d9                 generic        Intel Corporation
/0/da                 generic        Intel Corporation
/0/db                 generic        Intel Corporation
/0/dc                 generic        Intel Corporation
/0/dd                 generic        Intel Corporation
/0/de                 generic        Intel Corporation
/0/df                 generic        Intel Corporation
/0/e0                 generic        Intel Corporation
/0/104                  bridge         Sky Lake-E PCI
↪  Express Root Port A
/0/104/0                display        GV100GL [Tesla
↪  V100 PCIe]
/0/5                  generic        Intel Corporation
/0/5.2                  generic        Sky Lake-E RAS
↪  Configuration Registers
/0/5.4                   generic        Intel
↪  Corporation
```

```
/0/e1                 generic       Intel Corporation
/0/e2                 generic       Intel Corporation
/0/e3                 generic       Intel Corporation
/0/e4                 generic       Intel Corporation
/0/e5                 generic       Intel Corporation
/0/e6                 generic       Intel Corporation
/0/e7                   generic        Sky Lake-E
↪  M3KTI Registers
/0/e8                   generic        Sky Lake-E
↪  M3KTI Registers
/0/e9                   generic        Sky Lake-E
↪  M3KTI Registers
/0/ea                   generic        Sky Lake-E
↪  M3KTI Registers
/0/eb                   generic        Sky Lake-E
↪  M3KTI Registers
/0/ec                   generic        Sky Lake-E
↪  M2PCI Registers
/0/ed                   generic        Sky Lake-E
↪  M2PCI Registers
/0/ee                   generic        Sky Lake-E
↪  M2PCI Registers
/0/ef                   generic        Sky Lake-E
↪  M2PCI Registers
/0/f0                   system         PnP device
↪  PNP0b00
/0/f1                   system         PnP device
↪  PNP0c02
/0/f2                   communication  PnP device
↪  PNP0501
/0/f3                   communication  PnP device
↪  PNP0501
/0/f4                   system         PnP device
↪  PNP0c02
/0/f5                   system         PnP device
↪  PNP0c02
WARNING: output may be incomplete or inaccurate, you
↪  should run this program as super-user.
```

ARTIFACT EVALUATION

Verification and validation studies: We evaluate the performance of the proposed MM-CSF format against six benchmarks. Among these six frameworks, BCSF-ALL, ParTI-COO, and FCOO are GPU based frameworks. HiCOO, SPLATT-ALL and SPLATT-ONE are CPU based frameworks. We show performance in terms of GFLOPS on six datasets: deli, nell1, nell2, flick, fr_m, fr_s and darpa. The steps to collect datasets and the running procedure can be found in the repository. We expect MM-CSF to outperform all other benchmarks for all the cases.

Accuracy and precision of timings: The following two tables summarize the GFLOPS achieved by MM-CSF and other state-of-the-art benchmarks on CPUs and GPUs.The column header represents the name of the benchmarks. The row header represents the datasets.

GFLOPS of MM-CSF and other GPU based frameworks:

MM-CSF, ALL-BCSF, ParTI-COO, FCOO deli: 364, 333, 271, fails nell1: 285, 270, 176, fails nell2: 763, 607, 313, fails flick: 435, 327, 295,fails fr_m: 235, 194, 127, fails fr_s: 228, 203, fails, fails darpa: 327, 209, 100, 29

GFLOPS of MM-CSF and other CPU based frameworks:

MM-CSF, HiCOO, SPLATT-ALL, SPLATT-ONE deli: 364, 7, 8, 13 nell1: 285, 5, 17, 18 nell2: 763, 78, 150, 225 flick: 435, 4, 7, 25 fr_m: 235, 5, 5, 4 fr_s: 228, 5, 4, 4 darpa: 327, 7, 7, 1