


Article

Efficient Tensor Sensing For RF Tomographic Imaging on GPUs

Tao Zhang ^{1,†,‡} , Da Xu ^{1,‡} and Firstname Lastname ^{2,*}

¹ Affiliation 1; e-mail@e-mail.com

² Affiliation 2; e-mail@e-mail.com

* Correspondence: e-mail@e-mail.com; Tel.: +x-xxx-xxx-xxxx

† Current address: Affiliation 3

‡ These authors contributed equally to this work.

Version December 28, 2018 submitted to Journal Not Specified

Abstract: Radio-frequency(RF)tomographic imaging is a promising technique for inferring multi-dimensional physical space by processing RF signals traversed across a region of interest. The transform-based tensor model is more considered to be more appropriate than conventional model based on vector. The Alt-Min approach proposed by Deng. demonstrate significant improvement in recovery error and convergence speed compared to prior tensor-based compressed sensing. However, the running time of Tubal-Alt-Min increases exponentially with the dimension of tensors, thus not very practical for medium- or large-scale tensors. In this paper, we address this problem by exploiting massively parallel GPUs. We design, implement and optimize the Alt-Min algorithm on a GPU and evaluate the performance in terms of running time, recovery error, and convergence speed.

Keywords: keyword 1; keyword 2; keyword 3 (list three to ten pertinent keywords specific to the article, yet reasonably common within the subject discipline.)

0. How to Use this Template

The template details the sections that can be used in a manuscript. Note that the order and names of article sections may differ from the requirements of the journal (e.g., the positioning of the Materials and Methods section). Please check the instructions for authors page of the journal to verify the correct order and names. For any questions, please contact the editorial office of the journal or support@mdpi.com. For LaTeX related questions please contact Janine Daum at latex-support@mdpi.com.

1. Introduction

2. Related Work

There are quite a lot of work on the reconstruction of RF tomography, which can be classified into vector-based and tensor-based methods. The vector-based RF tomography compression sensing methods are proposed in [? ?]. However, these methods can only infer two-dimensional space because they ignore the spatial structure of the data. The tensor-based approach is proposed in [?], which uses tensor nuclear norm(TNN) [?] to extend RF tomographic problems to three-dimensional case. The method requires tensor singular value decomposition(t-SVD) , which is high computational complexity. Furthermore, the recovery error of this method is relatively high and the size of the data that can be processed is too small for the actual scene. Reference []uses a transform-based tensor model[?] to efficiently explore three-dimensional spatial structures for higher accuracy and efficiency.

Many existing researches [?] [?] [?] [?] [?] have demonstrated the benefit of utilizing GPUs to accelerate general purpose computing. Because of the massive parallelism and high memory bandwidth, GPUs are often used in accelerating machine learning applications [?] [?] [?]. Recently it has been applied to accelerate tensor contractions [?] and the proposed new BLAS-like primitive called STRIDEDBATCHEDGEMM is capable of performing a wide range of tensor contractions on CPU and GPU efficiently. Sparse tensor-times-dense matrix multiply is a critical bottleneck in data analysis and mining applications and [?] proposed an efficient primitive on CPU and GPU platforms. Solving large numbers of small linear algebra problems simultaneously is becoming increasingly important in many application areas and [?] made a systematic work to optimize batched linear algebra for GPUs.

3. Notations and Tensor Sensing Problem

We first formulate the RF tomographic imaging task as a tensor sensing problem, then review the Alt-Min algorithm.

3.1. Notations and Definition

We use lowercase boldface letter $\mathbf{x} \in \mathbb{R}^{N_1}$ to denote a vector, uppercase boldface letter $\mathbf{X} \in \mathbb{R}^{N_1 \times N_2}$ to denote a matrix, and calligraphic letter $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ to denote a tensor. $[k]$ denotes the set $\{1, 2, \dots, k\}$. Let $\mathcal{A} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ denote a thirdorder tensor. $\mathcal{A}(:, j, k)$, $\mathcal{A}(i, :, k)$, $\mathcal{A}(i, j, :)$ denote mode-1, mode-2, mode-3 tubes of \mathcal{A} , and $\mathcal{A}(:, :, k)$, $\mathcal{A}(:, j, :)$, $\mathcal{A}(i, :, :)$ denote the frontal, lateral, and horizontal slices. The Frobenius norm of \mathcal{A} is defined as $\|\mathcal{A}\|_F = \sqrt{\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \sum_{k=1}^{N_3} \mathcal{A}_{ijk}^2}$. The operator $\text{vec}(\cdot)$ transforms tensors and matrices into vectors. Let \mathbf{X}^T and \mathcal{A}^\dagger denote the transposes of a matrix and a tensor, respectively.

Definition 3.1. Given an invertible discrete transform $\mathcal{L} : \mathbb{R}^{1 \times 1 \times N_3} \rightarrow \mathbb{R}^{1 \times 1 \times N_3}$, the elementwise multiplication \circ , and $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{1 \times 1 \times N_3}$, the tubal-scalar multiplication is defined as:

$$\mathbf{a} \bullet \mathbf{b} = \mathcal{L}^{-1}(\mathcal{L}(\mathbf{a}) \circ \mathcal{L}(\mathbf{b})) \quad (1)$$

Definition 3.2. The \mathcal{L} -product $\mathcal{A} = \mathcal{B} \bullet \mathcal{C}$ of $\mathcal{B} \in \mathbb{R}^{N_1 \times r \times N_3}$ and $\mathcal{C} \in \mathbb{R}^{r \times N_2 \times N_3}$ is a tensor of size $N_1 \times N_2 \times N_3$, $\mathcal{A}(i, j, :) = \sum_{s=1}^r \mathcal{B}(i, s, :) \bullet \mathcal{C}(s, j, :)$, for $i \in [N_1]$ and $j \in [N_2]$.

Definition 3.3. The transform domain singular value decomposition \mathcal{L} -SVD of $\mathcal{A} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ is given by $\mathcal{A} = \mathcal{U} \bullet \mathcal{S} \bullet \mathcal{V}^\dagger$, where \mathcal{U} and \mathcal{V} are \mathcal{L} -orthogonal tensors of size $N_1 \times N_1 \times N_3$ and $N_2 \times N_2 \times N_3$ respectively, and \mathcal{S} is a diagonal tensor of size $N_1 \times N_2 \times N_3$. The entries of \mathcal{S} are called the singular values of \mathcal{A} , and the number of non-zero ones is called the \mathcal{L} -rank of \mathcal{A} .

We stack the RF signal measurements \mathbf{y}_m into a measurement vector $\mathbf{y} \in \mathbb{R}^M$. Given linear measurements $\mathbf{y}_m = \langle \mathcal{X}, \mathcal{A} \rangle = (\text{vec}(\mathcal{A}))^T \cdot \text{vec}(\mathcal{X})$, $1 \leq m \leq M$ of a loss field tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ with \mathcal{L} -rank r and the sensing tensors $\mathcal{A}_m \in \mathbb{R}^{N_1 \times N_2 \times N_3}$. With a linear map $\mathcal{H}(\cdot) : \mathbb{R}^{N_1 \times N_2 \times N_3} \rightarrow \mathbb{R}^M$. We get:

$$\mathbf{y} = \mathcal{H}(\mathcal{X}) + \mathbf{w}, \quad (2)$$

where \mathbf{w} denotes the noise vector.

The goal of RF tomographic imaging is to recover the loss field tensor \mathcal{X} from the measurement vector \mathbf{y} . We formulate this problem as a low \mathcal{L} -rank tensor sensing problem:

$$\hat{\mathcal{X}} = \arg \min_{\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}} \|\mathbf{y} - \mathcal{H}(\mathcal{X})\|_F^2, \text{ s.t. } \text{rank}(\mathcal{X}) \leq r. \quad (3)$$

procedure 1 Alt-min: $\text{AM}(\mathcal{H}(\cdot), \mathbf{y}, r, L)$

Input: linear map $\mathcal{H}(\cdot)$, measurement vector \mathbf{y} , \mathcal{L} -rank r , iteration number L .

- 1: Initialize \mathcal{U}^0 randomly;
- 2: **for** $\ell = 1$ to L **do**
- 3: $\mathcal{V}^\ell \leftarrow \text{LS}(\mathcal{H}(\cdot), \mathcal{U}^{\ell-1}, \mathbf{y}, r)$
- 4: $\mathcal{U}^\ell \leftarrow \text{LS}(\mathcal{H}(\cdot), \mathcal{V}^\ell, \mathbf{y}, r)$
- 5: **end for**

Output: : Pair of tensors $(\mathcal{U}^L, \mathcal{V}^L)$.

procedure 2 Least Squares Minimization: $\text{LS}(\mathcal{H}(\cdot), \mathcal{U}, r, \mathbf{Y})$

Input: linear map $\mathcal{H}(\cdot)$, tensor $\mathcal{U} \in \mathbb{R}^{N_1 \times r \times N_3}$, measurement vector \mathbf{y} , \mathcal{L} -rank r .

$$\hat{\mathcal{V}} = \arg \min_{\mathcal{V} \in \mathbb{R}^{r \times N_2 \times N_3}} \|\mathbf{y} - \mathcal{H}(\mathcal{U} \bullet \mathcal{V})\|_F^2,$$

Output: tensor \mathcal{V}

59 3.2. The Alt-min Algorithm

To enable alternating minimization, we represent the loss field tensor as the \mathcal{L} -product of two smaller tensors, i.e., $\mathcal{X} = \mathcal{U} \bullet \mathcal{V}$, $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$, $\mathcal{U} \in \mathbb{R}^{N_1 \times r \times N_3}$ and $\mathcal{V} \in \mathbb{R}^{r \times N_2 \times N_3}$. Then we reformulate equation (3) as the following non-convex optimization problem:

$$\hat{\mathcal{X}} = \arg \min_{\mathcal{U} \in \mathbb{R}^{N_1 \times r \times N_3}, \mathcal{V} \in \mathbb{R}^{r \times N_2 \times N_3}} \|\mathbf{y} - \mathcal{H}(\mathcal{U} \bullet \mathcal{V})\|_F^2, \quad (4)$$

60 The main idea if Alt-min is to iteratively estimate two low \mathcal{L} -rank tensors \mathcal{U} and \mathcal{V} , each of \mathcal{L}
 61 rank r . The key step is least squares(LS) minimization. The detailed implementation of LS minimization
 62 are given below.

We use the operator $\text{circ}(\cdot)$ to map circulants to their corresponding circular matrices which are tagged with the superscript c , i.e. $\underline{\alpha}^c, \underline{\mathbf{A}}^c$:

$$\underline{\alpha}^c = \text{circ}(\underline{\alpha}) = \begin{bmatrix} \alpha_1 & \alpha_{N_3} & \cdots & \alpha_2 \\ \alpha_2 & \alpha_1 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \alpha_{N_3} \\ \alpha_{N_3} & \alpha_{N_3-1} & \cdots & \alpha_1 \end{bmatrix}$$

$$\underline{\mathbf{A}}^c = \text{circ}(\underline{\mathbf{A}}) = \begin{bmatrix} \text{circ}(\underline{\mathbf{A}}_{1,1}) & \cdots & \text{circ}(\underline{\mathbf{A}}_{1,N_2}) \\ \vdots & \vdots & \vdots \\ \text{circ}(\underline{\mathbf{A}}_{N_1,1}) & \cdots & \text{circ}(\underline{\mathbf{A}}_{N_1,N_2}) \end{bmatrix}$$

For simplicity, we use \mathcal{A}^c to represent the circular matrix of tensor \mathcal{A} . Then the \mathcal{L} -product $\mathcal{X} = \mathcal{U} \bullet \mathcal{V}$ has an equivalent matrix-product as:

$$\mathbf{X}^c = \mathbf{U}^c \mathbf{V}^c, \quad (5)$$

where $\mathbf{X}^c \in \mathbb{R}^{N_1 N_3 \times N_2 N_3}$, $\mathbf{U}^c \in \mathbb{R}^{N_1 N_3 \times r N_3}$, $\mathbf{V}^c \in \mathbb{R}^{r N_3 \times N_2 N_3}$. We can transform the LS minimization in Alg.2 to the corresponding circular matrix representation:

$$\hat{\mathbf{V}}^c = \arg \min_{\mathbf{V}^c \in \mathbb{R}^{r N_3 \times N_2 N_3}} \|\mathbf{y} - \mathcal{H}(\mathbf{U}^c \mathbf{V}^c)\|_F^2, \quad (6)$$

where $\mathcal{H}(\cdot) : \mathbb{R}^{N_1 N_2 \times N_2 N_3} \rightarrow \mathbb{R}^M$ is the corresponding linear map in the circular matrix representation, with $\mathbf{y} = \mathcal{H}(\mathbf{X}^c)$. Each sensing tensor \mathcal{A}_m is transformed into its circular matrix $\mathbf{A}_m^c \in \mathbb{R}^{N_1 N_3 \times N_2 N_3}$, and $\mathbf{y}_m = \langle \mathbf{X}^c, \mathbf{A}_m^c \rangle$, $1 \leq m \leq M$. Similarly, we can estimate \mathbf{U}^c in the follow way:

$$\hat{\mathbf{U}}^c = \arg \min_{\mathbf{U}^c \in \mathbb{R}^{r N_3 \times N_2 N_3}} \|\mathbf{y} - \mathcal{H}(\mathbf{U}^c \mathbf{V}^c)\|_F^2. \quad (7)$$

63 We perform the following steps to solve this non-convex optimization problem:

Step 1). \mathbf{U}^c is used to form a block diagonal matrix \mathbf{B}_1 of size $N_1 N_2 N_3^2 \times r N_2 N_3^2$, and the number of \mathbf{U}^c is $N_2 N_3$,

$$\mathbf{B}_1 = \begin{bmatrix} \mathbf{U}^c & & & \\ & \mathbf{U}^c & & \\ & & \ddots & \\ & & & \mathbf{U}^c \end{bmatrix} \quad (8)$$

Step 2). Stack all the columns of \mathbf{V}^c , and then \mathbf{V}^c is vectorized to a vector \mathbf{b} of size $r N_2 N_3 N_3^2 \times 1$ as follows:

$$\mathbf{b} = \text{vec}(\mathbf{V}^c) = [\mathbf{V}^c(:, 1)^T, \mathbf{V}^c(:, 2)^T, \dots, \mathbf{V}^c(:, N_2 N_3)^T]^T. \quad (9)$$

Step 3). Each \mathbf{A}_m^c , $1 \leq m \leq M$ is represented as a vector \mathbf{c}_m of size $N_1 N_2 N_3^2 \times 1$ in the following way:

$$\mathbf{c}_m = \text{vec}(\mathbf{A}_m^c) = [\mathbf{A}_m^c(:, 1)^T, \mathbf{A}_m^c(:, 2)^T, \dots, \mathbf{A}_m^c(:, N_2 N_3)^T]^T. \quad (10)$$

and then all the \mathbf{c}_m are transformed into a matrix \mathbf{B}_2 of size $M \times N_1 N_2 N_3^2$:

$$\mathbf{B}_2 = [\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_M]^T \quad (11)$$

Therefore, the estimation of \mathbf{V}^c is transformed into the following standard least squares minimization problem:

$$\hat{\mathbf{b}} = \arg \min_{\mathbf{b} \in \mathbb{R}^{r N_2 N_3^2 \times 1}} \|\mathbf{y} - \mathbf{B}_2 \mathbf{B}_1 \mathbf{b}\|_F^2, \quad (12)$$

64 4. Algorithm implementation on GPU

65 4.1. Optimization of Alt-min

In circular algebra, it is obvious that the first column of \mathcal{X}^c already contains all the entries of itself, and there is no need to recover the redundant information. We only need to recover the first column of each $\text{circ}(\mathbf{X}_{i,j})$. We use the Matlab function `squeeze(·)` to get a new definition:

$$\mathbf{X}^s = \begin{bmatrix} \text{squeeze}(\mathcal{X}(1, 1, :)) & \cdots & \text{squeeze}(\mathcal{X}(1, N_2, :)) \\ \vdots & \ddots & \vdots \\ \text{squeeze}(\mathcal{X}(N_1, 1, :)) & \cdots & \text{squeeze}(\mathcal{X}(N_1, N_2, :)) \end{bmatrix}$$

66 , where `squeeze($\mathcal{X}(i, j, :)$)` transforms the i -th tube of the j -th lateral slice of \mathcal{X} into a vector of size
67 $N_3 \times 1$.

procedure 3 Tensor Sensing on CPU

Input: randomly initialized \mathbf{U}_0 , measurement vector $\mathbf{y} \in \mathbb{R}^M$, matrix $\mathbf{A} \in \mathbb{R}^{M \times N_1 N_2 N_3}$ converted from

M sensing tensors \mathcal{A}_m and the corresponding transpose matrix \mathbf{A}_t

Output: $\mathbf{X}_s \in \mathbb{R}^{N_1 N_3 \times N_2}$

```

1: for  $i = 1 \rightarrow IterNum$  do
2:    $\mathbf{U}_d = \text{diag}(\mathbf{U})$ 
3:    $\mathbf{W} = \mathbf{A}\mathbf{U}_d$ 
4:    $\mathbf{V}_v = \mathbf{W} \setminus \mathbf{y}$ 
5:    $\text{clear}(\mathbf{W})$ 
6:    $\mathbf{V} = \text{Vec2Mat}(\mathbf{V}_v)$ 
7:    $\mathbf{V}_t = \text{transpose}(\mathbf{V})$ 
8:    $\mathbf{V}_{t_d} = \text{diag}(\mathbf{V}_t)$ 
9:    $\mathbf{W} = \mathbf{A}_t \mathbf{V}_{t_d}$ 
10:   $\mathbf{U}_v = \mathbf{W} \setminus \mathbf{y}$ 
11:   $\text{clear}(\mathbf{W})$ 
12:   $\mathbf{U} = \text{transpose}(\text{Vec2Mat}(\mathbf{U}_v))$ 
13: end for
14: return  $\mathbf{X} = \mathbf{U}\mathbf{V}$ 

```

We use the notation \Leftrightarrow to denote a new mapping for \mathcal{L} -product as follows:

$$\mathcal{X} = \mathcal{U} \bullet \mathcal{V} \Leftrightarrow \mathbf{X}^s = \mathbf{U}^c \mathbf{V}^s, \quad (13)$$

where $\mathbf{X}^s \in \mathbb{R}^{N_1 N_3 \times N_2}$, $\mathbf{U}^c \in \mathbb{R}^{N_1 N_3 \times N_3}$, $\mathbf{V}^s \in \mathbb{R}^{r N_3 \times N_2}$. We can transform the LS minimization in Alg.2 to the following representation:

$$\hat{\mathbf{V}}^s = \arg \min_{\mathbf{V}^s \in \mathbb{R}^{r N_3 \times N_2}} \|\mathbf{y} - \mathcal{H}^s(\mathbf{U}^c \mathbf{V}^s)\|_F^2, \quad (14)$$

where $\mathcal{H}^s(\cdot) : \mathbb{R}^{N_1 N_3 \times N_2} \rightarrow \mathbb{R}^M$ is the corresponding linear map, with $\mathbf{y} = \mathbf{H}^s(\mathbf{X}^s)$, $\mathbf{y}_m = \langle \mathbf{X}^s, \mathbf{A}_m^s \rangle$, $1 \leq m \leq M$. Similarly, we can estimate \mathbf{U}^c in the following way:

$$\hat{\mathbf{U}}^c = \arg \min_{\mathbf{U}^c \in \mathbb{R}^{N_1 N_3 \times r N_3}} \|\mathbf{y} - \mathcal{H}^{sT}(\mathbf{U}^{cT} \mathbf{V}^{sT})\|_F^2, \quad (15)$$

So we can get the pseudocode of the tensor sensing as following:

4.2. Data Struct

In procedure 3, after least squares minimization(line 4 and 10), we get vectorized matrices. For a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the corresponding vectorized \mathbf{A} is $\mathbf{A}_v \in \mathbb{R}^{mn \times 1}$. And the vectorized matrices are converted back to the original matrices(line 5 and 11). In some scientific computing programming languages, such as Matlab, this conversion must be done with the appropriate conversion function. We adopt the column-first storage format to store matrices and vectors, which not only ensures read

75 and write continuity but also avoids explicit vector-to-matrix conversions that occur in original Matlab
 76 code because in this format \mathbf{A}_v and \mathbf{A} are the same in memory.

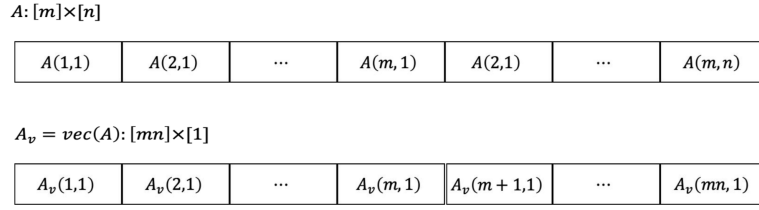


Figure 1. \mathbf{A} and $\text{vec}(\mathbf{A})$ in memory

77 4.3. Multiply of Block Diagonal Matrices

Using the operational properties of the block matrix we get:

$$[\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_{N_2}] \begin{bmatrix} \mathbf{U}^c & & \\ & \mathbf{U}^c & \\ & & \ddots \\ & & & \mathbf{U}^c \end{bmatrix} = [\mathbf{A}_1 \mathbf{U}^c, \mathbf{A}_2 \mathbf{U}^c, \dots, \mathbf{A}_{N_2} \mathbf{U}^c] \quad (16)$$

78 It shows that Multiplication of block diagonal matrices can be transformed into a batch of small matrix
 79 multiplications. As we use column-first format to store \mathbf{A} , the batch of \mathbf{A}_i are stored in constant stride.
 80 Let p indicate the location of the first element of \mathbf{A}_0 , then the location of the first element of \mathbf{A}_i is
 81 $p + i \times N_1 N_3$. We adopt *cublas* $< t > \text{gemmStridedBatchd}()$ function in cuBlas Library to achieve
 82 parallelism of operations.

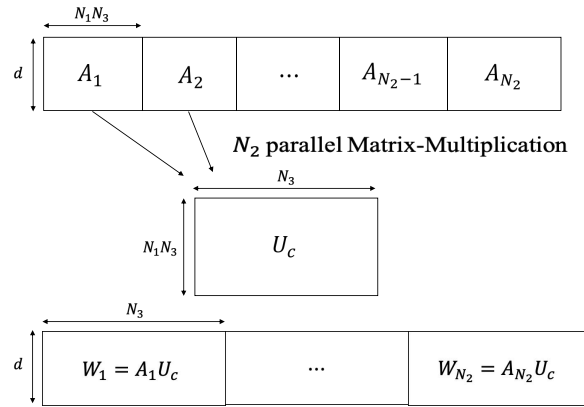


Figure 2. Multiplication of block diagonal matrices on GPU

83 4.4. Eliminate transpose operations

84 We noticed that after each Least Squares method, the transpose of the target matrix is obtained.
 85 The transpose operation of the matrix needs to be performed (line 7 and 12), while the transpose
 86 operation is quite complicated and will occupy more computing resources. As the operation after
 87 transpose of the matrix is multiplication of diagonal matrices and there is a parameter to control
 88 the transpose of the input matrices in the CUDA api *cublas* $< t > \text{gemmStridedBatchd}$, we set the
 89 parameter to perform the transpose implicitly.

Table 1. *cublas < t > gemmStridedBatchd* parameter settings

Parameters	meaning	value
transA	operation op(A) that is non- or transpose	non-transpose
transU	operation op(U) that is non- or transpose	transpose
A	pointer to the A matrix corresponding to the first instance of the batch	d_A
U	pointer to the U matrix	d_y
W	pointer to the W matrix	d_W
strideA	the address offset between \mathbf{A}_i and \mathbf{A}_{i+1}	$M \times N_1 N_3$
strideU	the address offset between \mathbf{U}_i and \mathbf{U}_{i+1}	0
strideW	the address offset between \mathbf{W}_i and \mathbf{W}_{i+1}	$M \times N_3$
batchNum	number of <i>gemm</i> to perform in the batch	N_2

90 4.5. Least Squares Minimization

As shown in 3, LSM (Least Squares Minimization) is the main step of the tensor sensing algorithm, which is the most time-consuming part of the entire algorithm. There are many approaches for least squares minimization. QR factorization is one of the most efficient approaches, which is well supported by CUDA. The least squares problem can be formulated as:

$$\tilde{\mathbf{x}} = \arg \min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_F^2,$$

where **A** is a matrix of size $m \times n$, **x** is a vector of size $k \times 1$, and **b** is a vector of size $n \times 1$. In the tensor algorithm, the system we work on is an overdetermined system, where $n > k$. We perform QR factorization on **A** and get $\mathbf{A} = \mathbf{Q}\mathbf{R}$, where **Q** is an orthogonal matrix of size $n \times k$ and **R** is an upper triangular matrix of size $k \times k$. Taking advantage of the special nature of these matrices we obtain **x** as following:

$$\mathbf{x} = \mathbf{R}^{-1} \mathbf{Q}^T * \mathbf{b}.$$

91 4.6. Data reuse and reduce data exchange

92 The two main operations in the loop are completed on the device side. Frequent data exchange
 93 will seriously affect the efficiency of the program. We use data reuse strategy to reduce data exchange
 94 and reduce storage resources occupied on the device side as much as possible. In the whole loop,
 95 we only do two data exchanges. One is that the data is transferred from the CPU to the GPU at the
 96 beginning of the loop, and the final result matrix is sent back from the GPU to the CPU. All intermediate
 97 results are covered with new results. One noteworthy thing is that since we use QR decomposition to
 98 solve the least squares problem, the input vector **y** is covered by the result vector (**U**, **V**), so we need to
 99 re-assign the vector **y** each time we perform the least squares method. However, it is obviously too
 100 time-consuming to read the original data of the vector **y** from the CPU every time. In order to solve
 101 this problem, we pre-apply a dyL that stores the original data of the vector **y**. Every time we need to
 102 reassign the dy, we use the api. . In this way, since the device-to-device assignment is much faster than
 103 the data exchange from the CPU to the GPU, at the expense of a small amount of storage space (the
 104 space of the vector **y** is small compared to the data such as **A**).) This is a very good acceleration we
 105 have achieved, which will show up in the experimental results in next section.

106 5. Experiment Methodology

107 In this section, we describe the experiment methodology including the hardware and software
 108 platform, testing data, testing process, and the comparison metrics.

109 5.1. Hardware and Software Platform

110 We use an NVIDIA Tesla V100 GPU card to evaluate performance of the method we propose.
 111 Tesla V100 is powered by NVIDIA Volta architecture, comes in 32GB configuration. It is installed on a

procedure 4 Tensor Sensing on GPU

Input: data on CPU memory: randomly initialized \mathbf{U}^0 , measurement vector $\mathbf{y} \in \mathbf{R}^M$, matrix $\mathbf{A} \in$

$\mathbf{R}^{M \times N_1 N_2 N_3}$ converted from M sensing tensors \mathcal{A}_m and the corresponding transpose matrix \mathbf{A}_t

Output: $\mathbf{X}_s \in \mathbf{R}^{N_1 N_3 \times N_2}$

- 1: apply for memory on GPU device: $d\mathbf{y}, d\mathbf{A}, d\mathbf{A}_t, d\mathbf{W}, d\mathbf{yL}$
- 2: data transfer: $\mathcal{A}, \mathcal{A}_t, \mathbf{y}, \mathbf{U}^0 \xrightarrow{\text{cudaMemcpyHostToDevice}} d\mathbf{A}(\mathcal{A}), d\mathbf{A}_t(\mathcal{A}_t), d\mathbf{y}(\mathbf{U}^0), d\mathbf{yL}(\mathbf{y})$ ($d\mathbf{A}(\mathcal{A})$ means that the content in $d\mathbf{A}$ is \mathcal{A} , the same below)
- 3: **for** $i = 0 \rightarrow \text{IterNum}$ **do**
- 4: $d\mathbf{W}(\text{uninitialized}), d\mathbf{A}(\mathcal{A}), d\mathbf{y}(\mathbf{U}^i) \xrightarrow{\text{cublas}<t>\text{gemmStridedBatchd}} d\mathbf{W}(\mathcal{A} \text{diag}(\mathbf{U}^i)), d\mathbf{A}(\mathcal{A}), d\mathbf{y}(\mathbf{U}^i)$
- 5: $d\mathbf{y}(\mathbf{U}^i), d\mathbf{yL}(\mathbf{y}) \xrightarrow{\text{cudaMemcpyDeviceToDevice}} d\mathbf{y}(\mathbf{y}), d\mathbf{yL}(\mathbf{y})$
- 6: $d\mathbf{W}(\mathcal{A} \text{diag}(\mathbf{U}^i)), d\mathbf{y}(\mathbf{y}) \xrightarrow{\text{cusolver}<t>\text{qr}} d\mathbf{y}(\mathbf{V}), d\mathbf{W}(\text{uninitialized})$
- 7: $d\mathbf{W}(\text{uninitialized}), d\mathbf{A}_t(\mathcal{A}_t), d\mathbf{y}(\mathbf{V}^i) \xrightarrow{\text{cublas}<t>\text{gemmStridedBatchd}} d\mathbf{W}(\mathcal{A}_t \text{diag}(\mathbf{V}^i)), d\mathbf{A}_t(\mathcal{A}_t), d\mathbf{y}(\mathbf{V}^i)$
- 8: $d\mathbf{y}(\mathbf{V}^i), d\mathbf{yL}(\mathbf{y}) \xrightarrow{\text{cudaMemcpyDeviceToDevice}} d\mathbf{y}(\mathbf{y}), d\mathbf{yL}(\mathbf{y})$
- 9: $d\mathbf{W}(\mathcal{A}_t \text{diag}(\mathbf{V}^i)), d\mathbf{y}(\mathbf{y}) \xrightarrow{\text{cusolver}<t>\text{qr}} d\mathbf{y}(\mathbf{U}^i), d\mathbf{W}(\text{uninitialized})$
- 10: **end for**
- 11: **return** $\mathbf{X}^s = \mathbf{UV}$

server with 128 GB memory and two Intel i7-7820x CPUs clocked at 3.6 GHz. The server is running Ubuntu 18.04.1 LTS with the kernel version 4.15.0. There is a Matlab R2017b installed that executes the CPU implementation in previous work [ref] as the baseline for comparison with our method. NVIDIA CUDA 10.0 [ref] is used in all experiments of our method execution.

5.2. Testing Data

In the experiment, we use both synthetic and real datasets to test our method. The synthetic data is generated according to the compressed sensing [ref4]. For real dataset, we use a IKEA 3D dataset that generate a ground truth tensor of size $40 \times 40 \times 6$ with \mathcal{L} -rank 1.

5.3. Testing Process

The synthetic and real datasets are processed by the Matlab code on CPUs and our method on GPUs, respectively. We repeat each experiment five times and report the average results.

5.4. Comparison Metrics

We compare the CPU algorithm with the GPU algorithm in three metrics: running time, relative error rate.

- *running time*: varying the tensor size and fixing other parameter, we measure the execution time of original Matlab method, our method without data reuse strategy and our final accelerated method. Finally we calculate speedups as the Matlab time divided by our method on GPU time.
- *error rate*: we adopt the metric relative square error, defined as $RSE = \|\hat{\mathcal{X}} - \mathcal{X}\|_F / \|\mathcal{X}\|_F$.

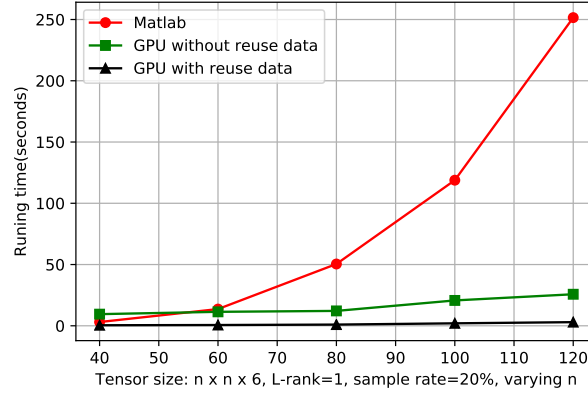


Figure 3. Running time of Matlab implementation and CUDA implementations

Table 2. Running time under different tensor size ($n \times n \times 6$).

n	40	60	80	100	120
Matlab time(S)	3.07	13.65	50.43	118.84	251.54
CUDA without reuse data time(S)	9.50	11.40	12.15	20.70	25.74
CUDA with reuse data time(S)	0.44	0.63	0.98	2.01	2.97
Speedups	6.98	21.67	51.46	59.12	84.70

6. Results and Analysis

6.1. Running time of Synthetic Data

Fig.?? shows that the running time of the two CUDA implementations and the Matlab implementation for \mathcal{X} of size $n \times n \times 6$ of \mathcal{L} -rank 1, where n varies from 40 to 120 at a step of 20. The sampling rate is set to 20%, and both CPU and GPU algorithms iterate 5 iterations for completion. The detailed time value is listed in ?? . While $M = N_1 \times N_2 \times N_3 \times \text{sampling rate}$ and $\mathcal{A} \in \mathbb{R}^{M \times N_1 N_2 N_3}$, the scale of the main operation matrix \mathcal{A} increases at a rate four times the rate of increase of n . It can be seen that the run time of the Matlab method without any parallel strategy is polynomial with the increase of n , and both CUDA implementations are linearly changed. As for CUDA implementation without data reuse, when the matrix size is small, the data exchange takes up most of the running time and the running time is even longer than the Matlab implementation. As for our final method, which is with data reuse and other parallel acceleration strategy, it achieves a good speedup at all scales and allows for greater speedups in larger data sizes where GPU memory is allowed.

6.2. Error Rate of Real Data

This experiment compares the error rate if the Matlab and CUDA implementation under different iterations for \mathcal{X} of size $40 \times 40 \times 6$ with \mathcal{L} -rank 1. The sampling rate is set to 50%.As shown in Fig. ??, when iteration varies from 1 to 30, the RSEs drop significantly. More importantly, the origin Matlab implementation and the CUDA implementation achieve almost the same RSEs at all iterations (the two curves in Fig. ?? overlap), which means that the two implementations have similar error rate performance in tensor sensing.

7. Conclusions

This section is not mandatory, but can be added to the manuscript if the discussion is unusually long or complex.

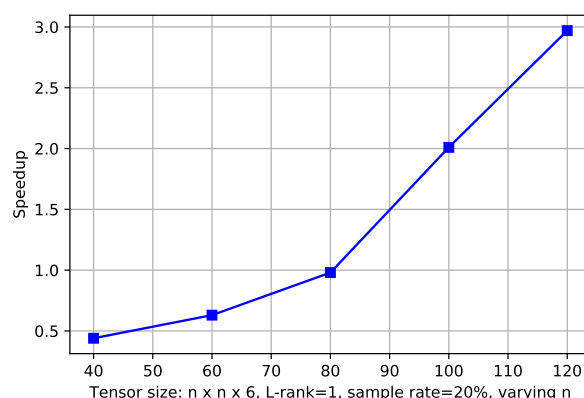


Figure 4. Speedups of Matlab implementation and CUDA implementation

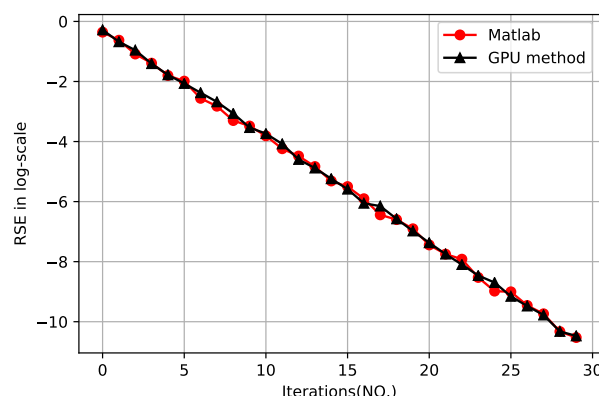


Figure 5. RSE of Matlab implementation and CUDA implementation

8. Patents

This section is not mandatory, but may be added if there are patents resulting from the work reported in this manuscript.

Author Contributions: For research articles with several authors, a short paragraph specifying their individual contributions must be provided. The following statements should be used “conceptualization, X.X. and Y.Y.; methodology, X.X.; software, X.X.; validation, X.X., Y.Y. and Z.Z.; formal analysis, X.X.; investigation, X.X.; resources, X.X.; data curation, X.X.; writing—original draft preparation, X.X.; writing—review and editing, X.X.; visualization, X.X.; supervision, X.X.; project administration, X.X.; funding acquisition, Y.Y.”, please turn to the [CRediT taxonomy](#) for the term explanation. Authorship must be limited to those who have contributed substantially to the work reported.

Funding: Please add: “This research received no external funding” or “This research was funded by NAME OF FUNDER grant number XXX.” and “The APC was funded by XXX”. Check carefully that the details given are accurate and use the standard spelling of funding agency names at <https://search.crossref.org/funding>, any errors may affect your future funding.

Acknowledgments: In this section you can acknowledge any support given which is not covered by the author contribution or funding sections. This may include administrative and technical support, or donations in kind (e.g., materials used for experiments).

Conflicts of Interest: Declare conflicts of interest or state “The authors declare no conflict of interest.” Authors must identify and declare any personal circumstances or interest that may be perceived as inappropriately influencing the representation or interpretation of reported research results. Any role of the funders in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript, or in the decision to publish the results must be declared in this section. If there is no role, please state “The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results”.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
DOAJ	Directory of open access journals
TLA	Three letter acronym
LD	linear dichroism

Appendix A

Appendix A.1

The appendix is an optional section that can contain details and data supplemental to the main text. For example, explanations of experimental details that would disrupt the flow of the main text, but nonetheless remain crucial to understanding and reproducing the research shown; figures of replicates for experiments of which representative data is shown in the main text can be added here if brief, or as Supplementary data. Mathematical proofs of results not central to the paper can be added as an appendix.

Appendix B

All appendix sections must be cited in the main text. In the appendixes, Figures, Tables, etc. should be labeled starting with 'A', e.g., Figure A1, Figure A2, etc.

© 2018 by the authors. Submitted to *Journal Not Specified* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).