Tensor Data Scattering and the Impossibility of Slicing Theorem

Wuming Pan

Sichuan University, Chengdu, 610065, China
panwuming@scu.edu.cn

Abstract. This paper proposes a standard way to represent sparse tensors. A broad theoretical framework for tensor data scattering methods used in various deep learning frameworks is established. This paper presents a theorem that is very important for performance analysis and accelerator optimization for implementing data scattering. The theorem shows how the impossibility of slicing happens in tenser data scattering. A sparsity measuring formula is provided, which can effectively indicate the storage efficiency of sparse tensor and the possibility of parallelly using it. The source code, including CUDA code, is provided in a related open-source project.

Keywords: tensor, pick, x-sparse tensor, sparsity.

1 Introduction

Most data used in AI and big data analysis is multidimensional in nature. Storing them in multi-dimension way, known as tensors, is more efficient than matrixes or two-dimensional arrays, hence tensor is gaining more and more importance in AI computing. Some current computing architectures support parallel computing along two or three dimensions on data, such as CUDA architectures [1], this facilitates computing related to tensors. However, many matrixes and tensors used in AI computing contain fewer data than their capacities, and they are often stored sparsely. For example, sparse attentions, which attract recent research interests, always result in sparse matrixes [2]. Sparsely stored matrixes or tensors are difficult to use the hardware features of machine learning accelerators.

Sparse tensors are typically stored with an array of indices and an array of values at corresponding indices, which is like what sparse matrixes are. Sparse tensors can be more easily found inherent duplication on their storing structures than sparse matrixes, hence they can be stored and used in different ways. Some APIs in deep learning frameworks, including TensorFlow and pyTorch, are developed to store and use sparse tensors in such ways, they are often called scatter APIs [3,4]. If a sparse tensor has duplicated storing structures along some dimensions, it can be transported and used in computing parallelly. However, there hasn’t common way to scattering tensor data. We will analyze the reasons for those difficulties in this paper. Though there are many tensor operations having been provided, such as NumPy array operations [5] and tubal-rank tensor operations [6], we still define new operations on tensors in this paper.

Next in this paper we will define tensors and related notions mathematically. In section 3, we define a useful operation called picks. Then, in the section 4, we define the tensor variator and its provision tensor that are critical for tensor data scattering. In the fifth section we discuss the uncertainty brought about by the applying of the variator, which is the source of uncertainty in the scattering of tensor data. In section 6, we define scattering and suggest how scatterings can be sliceable. In section 7, we define the x-sparse representation of sparse tensors and the x-scattering operation, suggest how to count sparsity of a sparse tensor, and show how the TensorFlow and pyTorch style APIs are mocked. The last section is the conclusion.

2 Tensor

To strictly discuss these problems, we need argue them mathematically. Other than in programming language, the syntax representations are not able to make two mathematical objects different. All mathematical objects eventually should be embodied in the set theory. We need start from elementary mathematic objects, such as the set $\mathbb{N}$ of natural numbers, the real number set $\mathbb{R}$, products of sets, mappings, and functions between sets, etc. We use $\mathbb{N}_m$ to denote the set of m nonnegative integers from 0 to $m-1$, i.e. $\mathbb{N}_m = \{0,1,\cdots,m-1\}$. Specifically, we think that $\mathbb{N}_0 = \{\,\} = \emptyset$. We also use $\mathbb{N}_{h:k}$ to denote the set $\{h, h+1, \cdots, k-1\}$ where $h \le k$ and both are integers, possibly negative.

A tuple is an element in a Cartesian product of some sets. For a tuple t, $\ell(t)$ denotes its length, i.e., the number of sets comprise product set which t is in. A tuple with length 1 is a number. A tuple with length 0 is not a number, just represented as $(\,)$. We treat tuples as they can be concatenated with operator + as python tuples. For example:

$$(1,2,3) + 4 = (1,2,3,4)$$

In deep learning area, a tensor is a multidimensional data array. Tensor's data elements can be accessed through their indices. An index of a data element is a tuple.

Definition 1. *A* ***tensor*** *is a function*

$$E\colon \prod_{i=0}^{k-1} \mathbb{N}_{m_i} \to \mathbb{R}$$

The tuple $S = (m_0, m_1, \cdots, m_{k-1})$ is called the ***shape*** *of tensor E, denoted with $\mathbb{s}_E$. And the elements in $\prod_{i=0}^{k-1} \mathbb{N}_{m_i}$ is called* ***indices*** *of E. The notation $\mathbb{I}_S$ stands for the set of all indices of shape S, i.e., $\mathbb{I}_S = \prod_{i=0}^{k-1} \mathbb{N}_{m_i}$, and the notation $\mathbb{I}_E$ stands for the set of all indices of tensor E as well. For any $(j_0, j_1, \cdots, j_{k-1}) \in \mathbb{I}_S$, we simply write $E\big((j_0, j_1, \cdots, j_{k-1})\big)$ as $E[j_0, j_1, \cdots, j_{k-1}]$. The number of total elements in E is called its* ***size****, denoted as Π_E or Π_S. The operator $\mathbb{t}$ change a one-dimension tensor to a tuple. A tensor with shape $(\,)$ is an empty tensor denoted as $[\,]$. We define that $\mathbb{t}([\,]) = (\,)$.*

To represent a tensor, we use [and] to bracket tensor data. Unlike representing a matrix, the subtensors are arranged horizontally or vertically in the same dimensions, but not at both directions. For example, a tensor $E0$ has the shape (3,3,2), and $E0$ is represented as

$$E\_0 = \big[[[0\quad 0][0\quad 1][0\quad 2]]\quad [[1\quad 0][1\quad 1][1\quad 2]]\quad [[2\quad 0][2\quad 1][2\quad 2]]\big]$$

or

$$E\_0 = \begin{bmatrix} [[0\quad 0][0\quad 1][0\quad 2]] \\ [[1\quad 0][1\quad 1][1\quad 2]] \\ [[2\quad 0][2\quad 1][2\quad 2]] \end{bmatrix}$$

These are different than matrix data arrange format.

3 Pick and Slice

Definition 2. *A* ***pick*** *is a finite integer function $p: \mathbb{N}_n \to \mathbb{N}$ for some nonnegative integer n. We also use $\ell(p)$ to denote n. The pick p is* ***smooth*** *if only if p monotonically maps consecutive numbers to consecutive numbers. We define that $\mathbb{c}(p) = p(\mathbb{N}_n)$. We use $max(p)$ to denote the maximum element in $\mathbb{c}(p)$. If $p(\mathbb{N}_n) = \mathbb{N}_n$, then we call p is a* ***shuffle****. If p is one to one, then we call p is* ***simple****. Let I be a tuple with $\ell(I) \geq max(p)$, $p(I)$ is a tuple J such that $J[i] = I[p(i)]$. If p is simple, then it has an inverse partial function $p^{-1}: \mathbb{c}(p) \to \mathbb{N}_n$. We define picks $\mathbb{i}_n: \mathbb{N}_n \to \mathbb{N}_n$ as $\mathbb{i}_n(j) = j$ and call them* ***identity picks*** *with rank n. We define picks $\mathbb{i}_{n:m}: \mathbb{N}_{n:m} \to \mathbb{N}_{n:m}$ as $\mathbb{i}_{n:m}(j - n) = j$ and call them* ***identity picks*** *with rank n to m.*

Sometimes a pick is used as a projection into indices along a dimension, at other times a pick can be used to select dimensions of a tensor. A pick can be written as a one-dimensional integer tensor.

Definition 3. *Given a simple pick p, let $\mathbb{I}_S/_p$ be the set of some subsets of $\mathbb{I}_S$ such that for any $C \in \mathbb{I}_S/_p$, there is a $I_C \in \mathbb{I}_S$, and*

$$C = \{K | p(I_C) = p(K), K \in \mathbb{I}_S\}$$

and vice versa. Let E be a tensor with shape S, we call $E(C)$ is a ***slice*** *of E* ***picked*** *by p. If $p = \mathbb{i}_m$ for some m, then we call $E(C)$ a* ***subtensor*** *of E, also denoted as $E(C) = E[p(I_C)]$.*

For example, let

$$E\_00 = [[0\quad 0][0\quad 1][0\quad 2]]$$
$$E\_01 = [[1\quad 0][1\quad 1][1\quad 2]]$$
$$E\_02 = [[2\quad 0][2\quad 1][2\quad 2]]$$

then $E00$, $E01$ and $E02$ are subtensors of a tensor $E0$ which can be represented as

$$\begin{aligned} E\_0 &= [E\_00\quad E\_01\quad E\_02] \\ &= [E\_0[0]\quad E\_0[1]\quad E\_0[2]] \end{aligned}$$

4 Tensor Variator and its Provision Tensor

Definition 4. *A **tensor variator** is a map $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$, it can be used as an operation on tensors. Let A be a tensor with shape S_0, then $T(A)$ is a set of tensors. For any $B \in T(A)$ and $I \in \mathbb{I}_{S_1}$, there exists a $J \in T^{-1}(T(I))$, such that $B[T(J)] = A[J]$.*

A variator can be defined by a tensor.

Definition 5. *Given a variator $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$, let $\mathbb{e}(T)$ be a tensor E with shape $S_0 + \ell(S_1)$ and it is defined by*

$$T(I) = \mathbb{t}(E[I])$$

*then we call T is **provisioned** by E, and E is a **provisioner** of T.*

Notes and Comments. $E[I]$ is a one-dimensional tensor of E.

For example, let $S_1 = (4,2)$ and $S_2 = (2,2,2,2)$, given tensor of shape $(4,2,4)$

$$E_1 = \begin{bmatrix} \begin{bmatrix} [0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 1] \end{bmatrix} \\ \begin{bmatrix} [0 & 0 & 1 & 0] \\ [0 & 0 & 1 & 1] \end{bmatrix} \\ \begin{bmatrix} [0 & 1 & 0 & 0] \\ [0 & 1 & 0 & 1] \end{bmatrix} \\ \begin{bmatrix} [0 & 1 & 1 & 0] \\ [0 & 1 & 1 & 1] \end{bmatrix} \end{bmatrix}$$

a variator T can be defined as

$$T(I) = (E\_1[I(0), I(1), 0], E\_1[I(0), I(1), 1], E\_1[I(0), I(1), 2], \\ E\_1[I(0), I(1), 3])$$

If $I = (3,0)$ then $T(I) = (0,1,1,0)$.

5 Nondeterministic of Applying Variator

Given a variator $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$, let A be a tensor with shape S_0, and a tensor $B \in T(A)$. There are non-deterministic cases when applying a variator on a tensor:

1. For any $B \in T(A)$, for any $I \in \mathbb{I}_{S_0}$, and for any $J \in T^{-1}(T(I))$, that $B(T(I)) = A(I)$ or $B(T(I)) = A(J)$ holds is both possible. If $A(I) \neq A(J)$, then only one is possibly true.
2. Moreover, for some $K \in \mathbb{I}_{\mathbb{s}_B}$, possibly $K \notin T(\mathbb{I}_{S_0})$ is true.
3. The shape of tensor B is not unique. Any shape S with $T(\mathbb{I}_{S_0}) \subset \mathbb{I}_S$ can be the shape of B. Only length of the shape B is definite.

Now we can investigate the scattering algorithms in deep learning frameworks. Those algorithms can eliminate the second and the third indeterminate problem above.

6 Scattering

6.1 Scatter APIs in Two Popular Deep Learning Frameworks

In TensorFlow, the typical scattering API looks like [5]:

```
tensor_scatter_nd_update(ts, indices, updates, name=None)
```

where *ts*, *indices*, *updates* in argument list are all tensors. Use the notions in this paper, the *indices* argument in this API is used to form a provisioner of a variator $T_{indices}\colon \mathbb{I}_{S_0} \to \mathbb{I}_{Si}$ where

$$S_0 = \mathbb{i}_{\ell(\mathbb{s}_{indices})-1}(\mathbb{s}_{indices})$$

and there are tuples Si and Su such that

$$\mathbb{s}_{ts} = S_2 = Si + Su$$

Let

$$S_1 = S_0 + Su$$

and $T_1\colon \mathbb{I}_{S_1} \to \mathbb{I}_{S_2}$ be a variator such that for any $I \in \mathbb{I}_{S_0}$ and $J \in \mathbb{I}_{Su}$

$$T_1(I + J) = T_{indices}(I) + J$$

This API creates a tensor B which is a result of variator T_1 being applied on tensor *updates*, such that there is an

$$I' \in T_{indices}^{-1}\big(T_{indices}(I)\big)$$

for each $I \in \mathbb{I}_{S_0}$ and

$$ts[T_{indices}(I)] = updates[I']$$

And for any

$$K \notin T_1\big(\mathbb{I}_{S_1}\big)$$

the identity $B(K) = ts(K)$ must holds.

In pyTorch, the typical scattering API looks like [6]:

```
scatter(self, dim, index, src, reduce=None)
```

where *self*, *index*, *src* in argument list are tensors having same shape, while *dim* is an integer. Still use the notions in this paper, the *index* argument in this API is a tensor to form a provisioner E_{index} of a variator $T_{E_{index}}\colon \mathbb{I}_{S_{index}} \to \mathbb{i}_1$ where $E_{index}[I] = \big[index[I]\big]$. Then we can define a variator $T_{scatter}\colon \mathbb{I}_{S_{index}} \to \mathbb{I}_{S_{index}}$ such that

$$T_{scatter}(I) = \mathbb{i}_{dim}(I) + T_{E_{index}}(I) + \mathbb{i}_{dim+1:\ell(I)}(I)$$

And then this API creates a tensor C which is a result of variator $T_{scatter}$ being applied on the tensor $self$, for any $I \in \mathbb{I}_{S_1}$, there is an

$$I' \in T_{scatter}^{-1}\big(T_{scatter}(I)\big)$$

such that

$$C\big(T_{scatter}(I)\big) = src(I')$$

For any $K \notin \mathbb{I}_{S_{index}}$, the identity $C(K) = self(K)$ must holds.

6.2 Defining Scattering

A scattering is a tensor variator being applied on a tensor A and the result tensor B is restricted by a tensor X.

Definition 6. *A* ***scattering*** *is a triple* $e = (T, A, X)$, *where* $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$ *is a tensor variator, A and X are two tensors. A result of scattering e is a result B of T being applied on A, and for any I, if* $I \in T(\mathbb{I}_{S_0})$ *then there is some* $J \in T^{-1}(I)$ *such that* $B[I] = A[J]$ *holds; if* $I \notin T(\mathbb{I}_{S_1})$, *then* $B[I] = X[I]$ *holds.*

Since a variator can be represented by a tensor, a scattering is also defined by a triple (E, A, X) of tensors E, A and X. For an instance, using tensor E_1 in section 4 to provision a variator, given

$$A\_1 = [[1 \quad 2] \quad [3 \quad 4] \quad [5 \quad 6] \quad [7 \quad 8]]$$

and

$$X\_1 = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} & \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} & \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

then the result of applying E_1 on A_1 into X_1 is a tensor

$$B\_1 = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} [1 & 2] \\ [3 & 4] \end{bmatrix} \\ \begin{bmatrix} [5 & 6] \\ [7 & 8] \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} \\ \begin{bmatrix} [0 & 0] \\ [0 & 0] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The result of scattering is also indeterministic.

6.3 Sliceable Scattering

In some case, when we scatter data from a source tensor into a target tensor, we can replace a slice of target tensor with a slice of source tensor, and the two slices have the same shape. If both slices can be specified with last few dimensions of shapes of both tensors, we can call such scatter operation sliceable. We extend this idea here:

Definition 7. *A tensor variator* $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$ *is called* ***sliceable****, if and only if there are two picks* p_1 *and* p_2*, and the three conditions are satisfied:*

1. p_1 *is simple.*
2. *For any* $I \in \mathbb{I}_{S_0}$, $\mathbb{c}(p_1) \cap \mathbb{c}(p_2) = \emptyset$.
3. *There is a variator* T' *such that for any* $I \in \mathbb{I}_{S_0}$, $T(I) = T'(p_1(I)) + p_2(I)$ *holds.*

p_2 *is called a* ***sliceable end pick*** *of T. If there does not exist a nonempty sliceable pick for T, we call it* ***not sliceable****.*

Theorem 1. **(Slice Theorem).** *Conditions being as in above definition, for any*

$$C \in {}^{\mathbb{I}_{S_0}}\!/_{p_1}$$

there is

$$T(C) \in {}^{\mathbb{I}_{S_1}}\!/_{\mathbb{i}_{\ell(S_1)-\ell(p_2)}}$$

Proof. Clear.

Notes and Comments. When $\mathbb{c}(p_1) \cap \mathbb{c}(p_2) \neq \emptyset$, the conclusion of this theorem no longer holds. In such case, there would be some $C' \in {}^{\mathbb{I}_{S_1}}\!/_{\mathring{\mathbb{I}}_{\ell(S_1)-\ell(p_2)}}$ and $T(C) \subseteq C'$.

Let's see an example. The tensor variator with a provision tensor

$$E\_2 = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} [0 & 0 & 0 & 0] \\ [0 & 0 & 0 & 1] \end{bmatrix} \\ \begin{bmatrix} [0 & 0 & 1 & 0] \\ [0 & 0 & 1 & 1] \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} [1 & 1 & 0 & 0] \\ [1 & 1 & 0 & 1] \end{bmatrix} \\ \begin{bmatrix} [1 & 1 & 1 & 0] \\ [1 & 1 & 1 & 1] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

is sliceable. Because there is a pick $p\_2\_1 = [0]$ and a pick $p\_2\_2 = [1 \quad 2]$ and a variator T'_2 whose provision tensor is

$$\mathbb{e}(T'\_2) = \begin{bmatrix} [0 & 0] \\ [1 & 1] \end{bmatrix}$$

7 Sparse Tensor with X-Sparse Representation

7.1 The Limitations in Current Scattering APIs

Not any kinds of scattering are implemented in deep learning framework currently. There are limitations in current scattering APIs:

1. The scattering APIs are not compatible with each other. For example, TensorFlow scattering APIs is sliceable, whereas pyTorch scattering APIs is almost not sliceable. Both cannot efficiently mock the behavior of each other.
2. A tensor variator properly represented as weakly sliceable can be stored efficiently. However, weakly sliceable scattering is not implemented in any deep learning frameworks. We will define weakly sliceable later.

Now we design a scattering algorithm which can directly address weak sliceable possibility of scattering. A weakly sliceable scattering API can incorporate functionalities of both TensorFlow and pyTorch scattering APIs.

7.2 X-Sparse Tensor

A x-variator T is decorated with three picks and a variator.

Definition 8. *A tensor variator $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$ is called a **x-variator**, if and only if there are three picks p_1, p_2, p and a variator $f: \mathbb{I}_{p_1(S_0)} \to \mathbb{I}_{S_2}$, and for any $I \in \mathbb{I}_{S_0}$ there is*

$$T(I) = p\left(f\big(p_1(I)\big) + p_2(I)\right)$$

*We call the tuple (f, p_1, p_2, p) a **x-representation**, or simply a **representation**, of T. If $max(p) > \ell(S_2)$, then the tuple is called a **normal** representation. If $\mathbb{c}(p_1) \cap \mathbb{c}(p_2) \neq \emptyset$, and there is some $i \in \mathbb{c}(p_1) \cap \mathbb{c}(p_2)$ and some $j \in p_2^{-1}(i)$*

$$j + \ell(S_2) \in \mathbb{c}(p)$$

*We call (f, p_1, p_2, p) is an **entangled** representation of T.*

Any variator $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$ evidently have a x-variator representation

$$\left(T, \mathbb{i}_{\ell(S_0)}, \mathbb{i}_0, \mathbb{i}_{\ell(S_1)}\right)$$

That is, for any $I \in \mathbb{I}_{S_0}$ we have

$$T(I) = \mathbb{i}_{\ell(S_1)}\left(T\left(\mathbb{i}_{\ell(S_0)}(I)\right) + \mathbb{i}_0(I)\right)$$

Corollary 1. *Given any variator T, it is a x-variator.*

The representation of a variator as a x-variator is not unique.

Theorem 2. **(Impossibility of Slicing Theorem).** *Given a variator T, if for any representation (f, p_1, p_2, p) of T, that the condition $\mathbb{c}(p_1) \cap \mathbb{c}(p_2) \neq \emptyset$ holds implies that the representation is entangled, then T is not sliceable.*

Proof. It is clear.

Definition 9. *We say a tensor variator $T: \mathbb{I}_{S_1} \to \mathbb{I}_{S_2}$ is **weakly sliceable**, if and only if T has a normal representation (f, p_1, p_2, p), such that $\mathbb{c}(p_1) \cap \mathbb{c}(p_2) = \emptyset$.*

Let's see another example. The variator

$$E\_3 = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} [0 & 0 & 0] \\ [1 & 0 & 1] \end{bmatrix} \\ \begin{bmatrix} [0 & 1 & 0] \\ [1 & 1 & 1] \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} [0 & 0 & 0] \\ [1 & 0 & 1] \end{bmatrix} \\ \begin{bmatrix} [0 & 1 & 0] \\ [1 & 1 & 1] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

is weakly sliceable. Because there are picks

$$p\_3\_1 = [0]$$
$$p\_3\_2 = [1]$$
$$p\_3 = [0 \quad 2 \quad 1]$$

and a variator f_3 whose provision tensor is

$$\mathbb{e}(f\_3) = \begin{bmatrix} [0 & 0] \\ [1 & 1] \end{bmatrix}$$

Definition 10. *Given a tensor variator $T: \mathbb{I}_{S_0} \to \mathbb{I}_{S_1}$ which has a normal representation (f, p_1, p_2, p), where p is a shaffle.* Let F be the provisioner tensor of the *variator $f: \mathbb{I}_{p_1(S_0)} \to \mathbb{I}_{S_2}$, and let V be a tensor with shape S_0, then (F, p_1, p_2, p, V) is called a **x-sparse tensor representation**, or simply **x-sparse tensor**. A **x-scattering** is a binary $e = (A, X)$, where $A = (F, p_1, p_2, p, V)$ is the x-sparse tensor, X is a tensor. A result of x-scattering e is a tensor B defined as for any J, if $J \in T\left(\mathbb{I}_{S_0}\right)$, then there is some $I \in \mathbb{I}_{S_0}$, such that*

$$J = p(J_0)$$

where

$$J_0 = p_1(I) + p_2(I)$$

and $B[J] = V(I)$; if $J \in \mathbb{I}_{S_B}$ and $J \notin T\left(\mathbb{I}_{S_0}\right)$, then $B[J] = X[J]$ holds

The result tensor of a x-scattering also cannot be certainly determined.

We implement x-scattering using python code [7] and call it scatterX API. C++ and CUDA code of x-scattering are also provided to move slices parallelly and utilize CUDA streams.

7.3 Counting Sparsity and Analyzing Performance

A dense tensor E with shape S has a x-sparse tensor representation $(\,[\,], \mathbb{i}_0, \mathbb{i}_n, \mathbb{i}_n, E)$. Once we randomly remove few elements from E and get a sparse tensor E', then E' has a x-sparse representation $(\,indices, \mathbb{i}_1, \mathbb{i}_0, \mathbb{i}_n, V)$ where $indices$ is a provisioner of a variator $T: \mathbb{I}_{\mathbb{i}_1} \rightarrow \mathbb{I}_{\mathbb{i}_n}$ and V is a one-dimensional tensor which contains elements of E'. Thus, the inner variator in a x-sparse tensor indicates the efficiency of storing sparse indices.

Definition 11: *Given a x-sparse tensor* $X = (F, p_1, p_2, p, V)$, *the* ***sparsity*** *of the x-sparse tensor is defined as*

$$\mathbb{a}_X = \frac{\Pi_F}{\Pi_V * \ell(p)}$$

Now we can count the sparsity of former examples:

$$\mathbb{a}_{(\,[\,], \mathbb{i}_0, \mathbb{i}_n, \mathbb{i}_n, E)} = 0$$
$$\mathbb{a}_{(\,indices, \mathbb{i}_1, \mathbb{i}_0, \mathbb{i}_n, V)} \approx 1$$

The sparsity is 1 means that the x-sparse tensor hardly can be parallelly used. The x-sparse tensor has smaller sparsity will have high possibility to be parallelly used.

7.4 Mocking Current Scattering APIs

The counterpart scattering of the TensorFlow scatter API as in section 6 has a x-scattering representation

$$\big((indices, p_1, p_2, p, updates), ts\big)$$

where

$$p = \mathbb{i}_{\ell(\mathbb{s}_{ts})}$$
$$n = \ell(\mathbb{s}_{indices}) - 1$$
$$p_1 = \mathbb{i}_n$$
$$m = \ell\big(\mathbb{s}_{updates}\big)$$
$$p_2 = \mathbb{i}_{n:m}$$

The sparsity

$$\mathbb{a}_{(indices, p_1, p_2, p, updates)} = \frac{\Pi_{indices}}{\Pi_{updates} * \ell(p)} \leq \frac{1}{\Pi_S * \ell(p)} \leq \frac{1}{\ell(p)}$$

where

$$S = \mathbb{i}_{\mathbb{s}_{indices}:m}\big(\mathbb{s}_{updates}\big)$$

It can be any number smaller than 1. Whereas the counterpart scattering of the pyTorch scatter API as in section 6 has a x-scattering representation

$$\big((E_{index}, q_1, q_2, q, src), self\big)$$

where

$$q = \mathbb{i}_{1:(dim+1)} + 0 + \mathbb{i}_{(dim+1):\ell(\mathbb{s}_{src})}$$
$$q_1 = q_2 = \mathbb{i}_{\ell(\mathbb{s}_{src})}$$

The sparsity

$$\mathbb{a}_{(E_{index}, q_1, q_2, q, src)} = \frac{\Pi_{E_{index}}}{\Pi_{src} * \ell(q)} = \frac{1}{\ell(q)}$$

This means that pyTorch scatter API is not sliceable.

The key difference of these two kinds of APIs is how the variator in scattering is formed.

8 Conclusion

Tensor data scattering is a kind of task that is difficult to use the hardware features of machine learning accelerators. This article theoretically analyses the reasons for this difficulty. And a general theory and algorithm of tensor data scaterring is established in this article. Based on the theories and algorithms in this article, we will be able to implement algorithms that can make better use of accelerator features. Moreover, a standard approach is proposed to represent sparse tensor, which can facilitate parallel computing and data transporting in AI accelerators, and which can also provide a way to efficiently store sparse indices of sparse tensors. A sparsity measuring formula is provided at last section, which can effectively indicate the storage efficiency of sparse tensor and the possibility of parallelly using it. More experiments and comparisons with APIs in other deep learning frameworks require more time. We will continue our work in this area and display the results in the GitHub project [7].

References

1. Soyata, T.: GPU parallel program development using CUDA. CRC Press, Florida (2018).
2. Child, R., Gray, S., Radford, A., Sutskever, Ilya.: Generating long sequences with sparse variators. CoRR, abs/1904.10509, (2019). http://arxiv.org/abs/ 1904.10509.
3. TensorFlow API: tf.tensor_scatter_nd_update, https://www.tensorflow.org/api_docs/python/tf/tensor_scatter_nd_update.
4. PyTorch Docs: torch.Tensor.scatter, https://pytorch.org/docs/stable/tensors.html?Highlight=scatter#torch.Tensor.scatter.
5. Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.*: Array programming with NumPy. Nature 585, 357–362 (2020).
6. Zhang, T., Liu, X., Wang, X., Walid, A.: cuTensor-Tubal: Efficient Primitives for Tubal-Rank Tensor Learning Operations on GPUs, IEEE Transactions on Parallel and Distributed Systems, 31(3), 595-610 (2020).
7. Algebraic Tensor Project, https://github.com/wmpan/AlgebraicTensor.