Flux.jl: Sparse GPU and ML support

Let CuArrays support sparse array and implement it into Flux.jl

Ziyi Xi(jack Xi)
xiziyi@msu.edu

Ph.D student in the department of Computational Mathematics Science and Engineering
Michigan State University, East Lansing, MI, 48824

1 Personal Background:

A brief intro

I am a first year PHD student in the department of Computational Mathematics Science and Engineering of Michigan State University, with rich numerical calculation experience and some background in parallel computing and GPU computing.

Research Experience

I was major in geophysics and computer science when I was an undergraduate in the University of Science and Technology of China (USTC). After coming to MSU, my project is mainly about computational seismology. By using the spectral element method, we simulate the seismic wave propagation in the Earth. By comparing the difference between the recorded seismic wave with the simulated result, we invert for the structure of the Earth. I'm also interested in using machine learning methods to have a better understanding of the earthquakes.

Motivation for the project

It's said that the 21st century is the age of computing. And using the technique of high performance computing is of vital importance for solving real world problems. I'm working on numerical simulations which usually requires thousands of CPU cores. By using GPUs we may improve the computing efficiency and reduce a huge amount of time and energy. Luckily Julia has the package about GPU computing, and obviously it performs very well. I love the idea of JuliaGPU and really want to have my own contribution to the project. And I could learn and use it in my own research project.

Flux.jl is also a very fascinating project as it's so elegant. And it's really young! So I am willing to develop myself with contributing to the project, and have a better understanding of the core functions of a machine learning package.

Programming Skills

- Know well: Python, FORTRAN, Julia, JavaScript, MPI
- Familiar with: C/C++, Shell, CUDA

Languages

Mandarin Chinese, English

2 Objective:

This project aims to provide CuArrays.jl the ability to construct the sparse array in the way similar to CuArray in the package, and make it possible to call CuSparse as the default back-end. After implementing the above part, we can make the Flux.jl directly use such the new sparse array since Flux.jl constructs its operatons on AbstractArray.

After finishing all the parts above, we can have some examples using such the sparse array for some specific machine learning problems, and benchmark the performance accordingly.

3 Introduction:

According to the description of the project in the idea page, the main difficulty of this project is to provide an up-to-date wrappers for sparse operations. At the moment, the CuArrays.jl has only exported CuArray which supports dense matrix operations. That's because CuArray inherits from GPUArray in GPUArrays.jl, and provides CuBlas as the back-end. If we want the CuArray supports sparse array, we have to use CuSparse as the back-end of GPUArray. But the problem here is that GPUArray is specially designed to use Blas-like operations, and sometimes is not so compatiable with CuSparse. Also some operations provided by CuBlas are not supported by CuSparse, so we have to convert the sparse matrix to dense matrix in such the case.

So, There are some stages I want to gradually implement listed below:

3.1 Wrap the sparse arrays to behave like the dense

The idea of this part is to wrap APIs in CuSparse.jl to the form in CuBlas.jl. Since people may wish to use the sparse array (we call it CuSparseArray for example) the same way with CuArray, wrapping APIs should be good for keeping the structure of the package and make the best use of the existing code.

Firstly, we could wrap APIs that are similar in CuSparse.jl and CuBlas.jl. Next, we could let the CuSparseArray supports other operations in CuBlas.jl, just by converting CuSparseArray to CuArray and do operations thereon.

Take the matrix multiplication as an example, if we multiply two sparse matrix A and B, like:

```
A=cu(rand(5,5),sparse=true) # provide a flag? Or A=cuSparse(ramd(5,5)
B=cu([1,2,3],sparse=false)
C=A*B
```

Firstly it will use the API like:

```
SparseArrays.mul!(A::CuSparseMat,B::CuArray)
```

or we use *LinearAlgebra.mul*!, for being coordinated with the Dense Array's API. And then it will use the CuSparse.jl as the backend, perform the operation as:

```
mm!(transa::SparseChar, alpha::BlasFloat, A::CuSparseMatrix, B::CuMatrix,
beta::BlasFloat, C::CuMatrix, index::SparseChar)
```

Or we use *mm*2!.

3.2 Directly use CuSparse.jl with the wrapped APIs as the back-end of GPUArray

Since the wrapped CuSparse.jl has the same APIs with CuBlas.jl, we can directly use it as the backend of GPUArray (maybe with some slightly modification of the code in GPUArray.jl). However, in GPUArrays.jl, there are some overload for functons in LinearAlgebra module while the sparse Arrays should be corresponding to functions in SparseArrays module. So we have to do some modifications in GPUArray.jl.

3.3 Use CuSparseArray in Flux.jl

Since CuSparseArray will have the same APIs with the CuArray. We can directly use CuSparseArray in Flux.jl as all the operations in Flux.jl are on AbstractArray. We may have to specially provide an interface in Flux.jl like what CuArray has.

4 Time line:

I suppose the detailed time line should be discussed with my potential mentors if I could be selected. So here I have only provided an overview of what should I do in each time periods.

4.1 Community Bonding Period

before May 6

- To familiarize myself with all the related packages about GPU computing in Julia.
- Survey on GPU computing packages in other languages.
- To familiarize myself with the realization of LinearAlgebra and SparseArrays in Julia.

May 6 - May 27 (Before the official coding time)

- Communicate with community members in Julia GPU computing and discuss the structure and final goal the project should look like.
- Start to write some prototype of the ideas about the project.

4.2 Coding Period

Week 1

- Write down the all the interface I want use in GPUArrays.jl and CuArrays.jl, and discuss if the structure of the package should be modified.

Week 2-4

- Code all the parts up in GPUArrays.jl and CuArrays.jl according to the interface that has been designed.

Week 5-7

- Have discussion with mentors and community members about the quality of the written code and have a code review.

Week 8-9

- Implement the sparse CuArray to Flux.jl and make some modification in Flux.jl accordingly.

Week 10-13

- Use the result from previous weeks to do some benchmark using the sparse Array in GPU. Write down the documentation and full-fill the testing part.