GSoC 2022

Adding GPU Support to mlpack

Project size Large ~350 hours

- **Name:** Gopi M. Tatiraju
- **University/school:** Working
- **Field of study:** Computer Science
- **Date study was started:** June 2017
- **Date study ended:** June 2021
- **Occupation:** Software Developer(FinTech industry)
- **Email:** tgopimanohar@gmail.com
- **IRC nick (if applicable):** heisenbug

- **What languages do you know? Rate your experience level (1-5: rookie-guru) for each.**
  C++: 3.5
  Python: 4

- **How long have you been coding in those languages?**
  C was my first programming language which was around 2013, and in the same year, I started exploring C++ as a part of my high school curriculum. I started python in 2017 when I went to university.

- **Are you a contributor to other open-source projects?**
  Yes, I've been trying my best to contribute to open-source projects.

- **Do you have a link to any of your work (i.e. github profile)?**
  Github: https://github.com/heisenbuug

- **Have you taken any coursework relevant to machine learning?**
  I have taken the Machine Learning Course by Andrew Ng and some other MOOCs from MIT Opencourseware and such platforms.

Project Abstract

The project introduces **GPU support to mlpack**. Currently under the hood mlpack uses **Armadillo** as a high quality linear algebra library based on CPU. The goal of the project is to contribute to **Bandicoot** which is the GPU version of armadillo and is under development. I will be implementing **CUDA kernels** which mlpack can utilize to train models on NVIDIA GPUs.

Idea growth

The idea originates from the **Bandicoot** project. Currently mlpack is built on armadillo. Bandicoot is basically armadillo on a GPU(hence faster) and hence with some design changes(about which I will discuss later) we can introduce GPU support to mlpack.

Last year while working on my previous GSoC(2021) project with mlpack, me and my mentor Omar were already planning to work on some template design changes in mlpack but since it was a bit out of scope of the project we decided to work on it later. I guess the time is now.

With regards to the project we have to provide implementation for multiple backends, but I guess first we should work on one backend, **CUDA preferably**. Or maybe we should work on both?

We already had many discussions about this in mlpack's meet-up and considering the nature and hours of project this year I think this can be a really good project. This project would definitely not introduce full-fledged GPU support, but once we get started with some layers we can keep working our way. This project will be like an opening act for Mlpack GPU support.

Overview

The project will require work on 2 repos.

- mlpack
- bandicoot

Bandicoot is a GPU accelerator add-on for the Armadillo C++ linear algebra library on which mlpack is built. I will be basically providing support for the NVIDIA backend(CUDA).

Final design goal is that, to utilize bandicoot, all we have to do is replace the **arma** namespace with **coot** namespace. Bandicoot will contain all the kernel implementations and GPU related code.

My goal for this project is to add GPU support for CNNs to mlpack. I will be implementing all the **arma functions being utilized by mlpack's ANN codebase**. The goal is to have support for **training a CNN on a GPU using mlpack**. Project can be divided into further parts:

1. Making mlpack ready for bandicoot, i.e. changing **eT** to **MatType**.
2. Implementing CUDA kernels(toughest part)
3. Benchmarking the kernels and optimizing if required
4. Writing tests for bandicoot
5. Benchmarking the implementations
6. Compiling an example to train a network on GPU
7. A web page about mlpack GPU support

Example/Models using GPU

Considering the mlpack's ResNet model
Here we can divide code into the following parts. We will be working on each part and adapting them for **coot**.

- ### Declaring some basic variables

  This part is fairly simple and doesn't require any changes. Here we initialize variables like step_size, number of neurons, etc.

- ### Loading/Saving data

  The current parser is not **coot** ready, i.e. it is still hardcoded to load data into **arma** matrices only. These changes will mostly have nothing to do with parser performance since we are only making a design change to make mlpack more compatible with any matrix backend, in this case GPU backend.

- ### Preprocessing the data

  For starters functions like **Split** should be ready for bandicoot.
  https://github.com/mlpack/mlpack/blob/master/src/mlpack/core/data/split_data.hpp

  Most parts in the **mlpack/core/data/** need to be worked on. We will be working on parts like **parser** and **split function.**

- ### Specify NN model

  As mentioned above most of the layers are already adapted for **coot,** so we can worry less about that.

- ### Train the network

  This part is also mostly done, for reference the new **Train()** should look like this.

- ### Save the model

  Comes under the **mlpack/core/data/** needs some adaptation with new design.

Making mlpack ready for bandicoot

Most of the mlpack functions follow this signature, considering Forward

```
template<typename eT>
void Forward(const arma::Mat<eT>& input, arma::Mat<eT>& output);
```

eT is the element type of the **arma::Mat,** so that the user can pass an element type int, float etc. But if the user wants to use some other matrix this design will not work, here we are sure to use arma. Since the plan to introduce the GPU support, mlpack has decided to change this design.

Now we won't be passing the **Element Type**, but rather we will be passing the **Matrix Type** so that we can use any matrix backend.

```
void Forward(const MatType& input, MatType& output);
```

This function can either accept an **arma::mat**(CPU) or a **coot::mat**(GPU).
This makes sure that we don't have to write any kind of new wrapper classes for the GPU based matrix implementation.

Work done by Ryan on the Forward(ann-vtable branch) can be taken as inspiration for all the other parts of the library. These design changes are needed mostly everywhere in the code base; we will be mainly concentrating on functions utilized by ANN codebase.

I am working on a PR in which I will make the above mentioned changes for the **parser**(Data loader) of mlpack. Parser will include **loading/saving** of **.csv** files and **pre-build models**. If we have time I will also adapt other functions in **mlpack/core/data/** related to data-processing like **mlpack::split().**

I hope to get most of these changes before the GSoC starts since this is mostly just a design change.

Implementing CUDA kernels

We need to implement all the **arma functions** that ANN codebase uses. I am making this list keeping in mind that at the end of the project we want to support ResNet Model

Layers we need to work on, we won't be working on layers directly but rather on the **arma** functions which these layers are using. List of layers

- Linear
- Sequential
- AddMerge
- ReLU
- IdentityLayer
- Padding
- MaxPooling
- AdaptiveMeanPooling
- CrossEntropyError

I've mostly covered all the functions we have to work on, if anything more comes up we can add those as well.

List of CUDA kernels we need to implement

- **Functions of Vectors/Matrices/Cubes**
  - vectorise()
  - reshape()
  - flipr()
  - flipd()

- **Decompositions, Factorisations, Inverses and Equation Solvers**
  - solve()
  - qr_econ()
  - svd()
- **Signal & Image Processing**
  - fft2()
  - ifft2()

We can use libraries like thrust**,** cuSolver, cuRAND for the implementation of the above mentioned functions.

Not everything needs to be on the GPU, for example when considering **shuffle()** some part(generate random permutation list) of the code is better off on CPU while remaining on GPU.

Goal is to first come-up with a working implementation first. Then time the kernel and if it's slow or bottleneck then we can follow up with more complex strategies.

Function list for template parameter change

File/Function Name	Comments
network_init.hpp	Initialize, needs work
ann/layer/	Most of the layers are ready for **coot** Work on the required layer(if any) from not_adapted/
core/data/	split() is **not** yet adapted for coot
core/data/	load/save is **not** yet adapted for coot

List of layers that are still written with the **old boost::visitor** interface:
https://github.com/zoq/mlpack/tree/ann-vtable/src/mlpack/methods/ann/layer/not_adapted
Layers that are not listed here are adapted to use inheritance instead.

Ryan went ahead and made most of the layers bandicoot ready(except for ones in the not adapted directory), i.e. each class has **MatType** as template parameter compared to **eT** previously and some other changes.

If required we can work on other layers, adapt each layer to use inheritance and make them bandicoot ready.

Benchmarking the kernels and optimizing if required

Once we come-up with a kernel for the said function we need to make sure that it is optimized enough to give us a bump.

Plan is to first write the most basic form of the required kernel and then use tools like profiler to understand the bottlenecks and work on them until we get to a satisfactory point.

We can just time them against their CPU counterpart to make sure that they are not bottlenecked. We can follow-up on the same benchmarking strategy that Ryan has followed here.

Writing tests for bandicoot

Each implemented function must be tested individually and in combination with other functions. Some tests are already implemented in bandicoot. We can use the same design and write tests. We might also be able to use test-cases from armadillo.

Benchmarking the implementations

Once we are done implementing all the required kernels, it's time to see the code in action and compare with other and mlpack(CPU) implementations.

- mlpack CPU vs mlpack GPU
- mlpack GPU vs some other library GPU implementation

Comparing it with our own CPU implementation will give us an idea of how much of a performance bump we can get when mlpack runs on GPU.

While going against the other implementation we can figure out how much more we need to speed up the implementation to beat the standards.

Compiling an example to train a network on GPU

Once we reach a high enough score, we can now implement some ready to use examples which can be added in the example repo.

Some small changes in **MakeFile** might be required. Maybe while building this repo we can choose which namespace to use, **arma** or **coot** and hence we can in future have all examples running for both CPU and GPU. This should not be complex and can be discussed in further stages as well.

A web page about mlpack GPU support

Once we have all the testing and benchmarking results we can compile it in the form of a web page and host it on mlpack's website. It's time to show-off the GPU power.

Relevant issues and PR

I've already started working on some parts of the project.

- Getting started with bandicoot: issue
- Basic skeleton for **join_cols:** merge request
- For Neural networks we have to implement a NN outside mlpack and train it with armadillo and ensmallen. Suggested here.

Project Timeline ~350 hours

The coding period is 12 weeks.

12 weeks of time seems fine to me, at max I would like to finish the project in 15 weeks.

- **Pre Coding period(~12th June)**
  Before the official coding period begins I would like to complete all the adoption changes from mlpack's side. I will concentrate on **mlpack/core/data/** directory. Firstly I will start with csv-parser and then to some data manipulation functions like **mlpack::split()**.

  I would also put some more time in research, like reading about kernels, and finding some interesting resources that might help us. Discussion with mentor regarding kernel implementations and more planning to make further development smoother.

- **June 13th - June 19th: Complete join_cols implementation**
  I've already started working on join_cols implementation. Initial idea is to implement it using cudaMemCpy and benchmark it and then decide if we need a kernel implementation or using **cudaMemCpy** is fast enough. Some discussion**.**

- **June 20th - June 26th**
  Will start working on implementing **functions of Matrices.** Let's start by implementing **vectorise()**. Goal would be first coming up with a working kernel and then time it and proceed accordingly.

- **June 27th - July 3rd**
  This week I will start working on **reshape()**. Write the implementation, test it and if it bottlenecks, try working out some more complex strategies.

- **July 4th - July 10th**
  Will start working on **fliplr() and flipud()**.

- **July 11th - July 17th**

Starting this week I would like to start working on some signal and image processing functions. I would start with **fft2 and ifft2()**.

- **July 18th - July 24th**
  I am assuming fft2 and ifft2 might take more than a week even if not we can use the extra time to make sure that we have implemented all the matrix functions that we need to support ResNet.

- **July 25th - July 31st**
  Coming four weeks I will be concentrating on **Decompositions, Factorisations, Inverses and Equation Solvers.** I will start with **solve()** which is used to solve a system of linear equations. On quick thought we can use **QR Decomposition** from **cuSolver** to solve the system. Note that this is a dense system.

- **August 1st - August 7th**
  I will start working on economical QR decomposition i.e. qr_econ(). Implementing this might be a bit complex. If we come-up with a good kernel, I think we can use the same in solve()?

- **August 8th - August 14th**
  We can get started with singular value decomposition. svd() is a bit more complex. We can take help of cuSolver to implement this.

- **August 15th - August 21nd**
  Considering we will not be able to finish deviating from the above timeline, these 2 weeks are for cover-up and remaining work.

- **August 22rd - August 28th**
  Considering we are on time, I will use these 2 weeks to implement some stat functions like median, standard deviation, variance.

- **August 29th - Sept 5th**
  Benchmarking individual kernels will be the main concentration here. Although we will be doing that side by side as well, benchmarking numbers now will be final since we are at the end of the project, if at all any kernel would require more work, we can add it in future work sections.

  I will also be writing tests for bandicoot. We already have some implemented here, so we can follow the same design. I guess we can also use test-cases from armadillo(as inspiration).

- **Sept 6th - Sept 12th(final week)**
  Wrapping up by writing the final report of the project. Will also work on creating a web page to show the GPU support with some speed upgrade graphs.

At the end I will open an issue which will talk about the future work regarding GPU support. Any developer who would like to contribute to bandicoot, this issue can be like a contribution guidelines

Extra Work

Just a list of points I would like to work on if we can complete all the above-mentioned tasks before the end of the coding period.

- Compile a list of functions needed by mlpack to extend GPU support
- Write a short sample on how to contribute to mlpack GPU support
- Making mlpack completely ready Bandicoot

Future Work

After completing GSoC successfully I am also planning to apply for **NumFocus SDG** and continue working on the project.

Even with or without SDG I will continue to contribute to mlpack as I've been doing for the past 2 years.

The scope of this project goes beyond the GSoC and wishes to achieve something that can be a major upgrade to the open-source community.