Final Project Report

GPU Parallel programming for CNN

Team Name: slowestTeam

Teammates: Zonglin Li, Yutao Ma, Tianxi Zhao

ECE 408

November 17, 2017

Introduction

The goal of our project is to apply parallel programming techniques on a convolutional neural network (CNN) for the MNIST dataset. To be more specific, our project focuses on modifying a CNN model implemented under the MXNet framework by using CUDA parallel programming to improve the runtime performance of the forward convolutional step. As we've learned in the lecture, the forward convolutional step convolves the input images with different feature kernels, and this step can be a very good practice to harness the parallel computing power of a Graphic Processing Unit(GPU).

In this report, the Progress section briefly introduces work that has been carried out by our team based on the milestones required by the assignments, and the Result section will illustrate our optimization results by using different implementations. The Conclusion section summarizes the result and our observations of our approaches. And the Teamwork section shows how this project was planned and finished by our team.

Current report is written under Google docs and is only for progress up to finishing milestone 2, so contents shown in this report is only a small part compare to the final report. For final report, we will use Latex instead of Google docs, so the typesetting and formatting will be much better than this preliminary version.

Progress

Milestone 1: First submission

In milestone 1, we managed to execute the example scripts (m1.1.py and m1.2.py) through rai (a client to interact with a cluster of computers) for both CPU and GPU implementation of the CNN. Further, we utilized nvprof to profile the GPU performance.

- The setup process:
  - Cloned the skeleton code from github
  - Downloaded the rai from rai project page (rai version: v0.2.17)
- rai commands executed:
  - rai -p <our project dir>
  - Modified image in rai_build.yml to run with GPU
  - Modified build commands to run different .py files
  - Added **`nvprof python /src/m1.2.py`** to build commands in rai_build.yml to profile GPU execution.

Milestone 2: CPU implementation

In milestone 2, we implemented the CPU version of the convolution operation under the MXNet framework. Specifically, our C++ code is compiled as part of a python module, which will later be loaded and used by MXNet. The convolution layer supports multiple filters and inputs with multiple channels. Here is our workflow:

- Reading documentations and spec
  - We looked up the APIs of Tensor class in the mshadow library from here.
  - We read through the lecture slide “2017FA_ECE408_dl03_CNN01.pptx” to understand how convolutional neural network works.

- CPU Implementation
  - We modified the file “new-forward.h” to implement a sequential CPU code for a forward convolutional layer. The main part of the code for is given below:

```
Void forward(mshadow::Tensor<cpu, 4, DType> &y, const mshadow::Tensor<cpu,    4, DType> &x, 
const mshadow::Tensor<cpu, 4, DType> &k)
{
        const int B = x.shape_[0]; // batch size
        const int M = y.shape_[1]; // Number of output channels
        const int C = x.shape_[1]; // Number of input channels
        const int H = x.shape_[2]; // height of each input channel
        const int W = x.shape_[3]; // width of each input channel
        const int K = k.shape_[3]; // size of the square kernel
        for (int b = 0; b < B; ++b)
          for (int m = 0; m < M; ++m)
            for (int h = 0; h < H-K+1; ++h)
              for (int w = 0; w < W-K+1; ++w){
                y[b][m][h][w] = 0;
                for (int c = 0; c < C; ++c)
                  for(int p = 0; p < K; ++p)
                    for(int q=0; q < K; ++q)
                      y[b][m][h][w] += x[b][c][h + p][w + q] * k[m][c][p][q];
              }
}
```

- Correctness
  - We have tested our implementation on both “ece408-high” and “ece408-low” models. The batch size for both models is 10000. The correctness and operation time for each model are shown below:
    Correctness: 0.8562 Model: ece408-high, Op Time: 9.023215
    Correctness: 0.629 Model: ece408-low, Op Time: 9.019743

  We got the expected correctness for each model.

Milestone 3: GPU implementation

Milestone 4: Further Optimization on GPU implementation

Results

Teamwork

- Team work
    - For milestone 1 and 2, because these checkpoints are short and very important for everyone to get familiar with the project platform, each team member implemented the code and tested its correctness separately. Then we chose one of the three versions to submit.
    - For future milestones, more team work will be included.