Korea Advanced Institute of Science and Technology

CS492 Spring 2020

Project 3

Seunghyo Kang

shkang@casys.kaist.ac.kr

Due: Jun. 2

Rules

- This project is a team project.
- Do not copy from others. You will get huge penalty according to the University policy. Sharing of code between teams is viewed as cheating.
- Every code was written and will be tested in the provided docker environment. You need to implement your code in the given environment.
- Read carefully not only this document but the comment in the skeleton code. This is the version 1.0 document which means not perfect. Any specifications or errors can
- be changed during the project period with announcement. Stay tuned to Piazza.

Introduction

You are going to implement same features of DNN graph in the previous project with some improvement using C language and vectorization. C language is known to be faster than python language. So, you will rewrite some of the code in C and connect it to python code. There are several libraries that boosts looping by parallelizing. In this project, you will learn (1) how to offload python code using C language, (2) how to use various BLAS libraries and vectorized APIs, and (3) how to test your code in Amazon EC2 instance.

Files

This project contains the following files:

- `examples/`: example files for various libraries.
- `src/`: reference codes of project 2
- `sample.jpg, sample_out.jpg`: Sample input, output image file.

Strategy

Ctypes

Ctypes is used to run a code from C in python. The C code should be compiled in a shared object form to use it in the python code. Following command will create `libexample.so` from `example.c`.

```
$ gcc -shared -fPIC -o libexample.so example.c
```

Check the example in example/ctypes, how it is compiled, and how it is imported in python code.

Use BLAS libraries

There are several libraries that support Basic Linear Algebra Subprograms (BLAS). Useful function in these libraries is `sgemm`(dgemm) which stands for single(double)-precision general matrix-matrix multiplication. The function automatically vectorizes matrix multiplication by one function call.

OpenBLAS is one of BLAS libraries which supports matrix multiplication using CPU. The arguments LDA, LDB, and LDC in cblas_dgemm would be important in calculating convolution. The LDX represents the first dimension of matrix X. We can convolve some portion of a matrix A and filter B to project it to resulting matrix C.

There is also BLAS library that uses GPU, cuBLAS. Use cublasSgemm for multiplication. Check example codes, run it and how the result came out.

AVX & pthread

AVX is an instruction level parallelism supported by CPU. It collects several variables (e.g. eight single-precision floats) in one data type to compute them in a single instruction. To compiled AVX code, you need to add -mavx2 flag. Check Makefile in examples/AVX. Also, pthread is well-known parallelizing library with spawning threads. With these techniques, you can achieve massive parallelism in computation.

CUDA

CUDA is a computing platform that enables parallel programing with GPU. We set the behavior of one computation unit in GPU, then the whole computation will be done in parallel automatically. Check the usage of CUDA library in examples/CUDA. See how the inputs are set up before they are fed to GPU and how the `add` function is defined and called.

Implementation

Overview

Strat from the baseline code in src. The code is the reference solution of project 2. You can start from yours if you want. There must be some computation that consume most of times (or you may have improved already). Improve dnn.py by offloading some code to C and using the introduced libraries. You must use specified libraries at a time. So, make four versions of dnn.py and corresponding C files. The names of the files must be dnn_openblas.py, dnn_openblas.c, dnn_avx.py, dnn_avx.c, dnn_cublas.py, dnn_cublas.cu, dnn_cuda.py, dnn_cuda.cu. You will get full points if your code is faster than the baseline code by offloading some computations to C code.

(1) CPU parallelization

  (a) Library-supported parallelization - OpenBLAS (**5 pt**)

  (b) Manual parallelization – AVX & pthread (**20 pt**)

(2) GPU parallelization

  (a) NVIDIA-provided high-level library – cuBLAS (**5 pt**)

  (b) Only CUDA primitives (**20 pt**)

Running Time Contest (5 pt each)

We will measure the running time of two versions of your code; AVX and CUDA. They will be tested in the Amazon EC2 instance. The first-place team will get 5 points, the second team gets 4 points, the third team gets 3 points, 4-6th get 2 points, and 7-11th get 1 point. The maximum point for this contest is 5 points for each version.

Report (10 pt)

Write a report that describes your works. Your report must include (1) key functions for parallelization (e.g. OpenBLAS – cblas_dgemm) and how they work, (2) parallelization strategies: which parts in dnn.py and how you parallelized them, and (3) performance gain from offloading the code and analysis for each parallelization scheme. The running time should be measured in the EC2 instance. The purpose of the report is to check your understanding. Please write the answer clear.

Test Your Code

Amazon EC2 instance

Amazon provides a cloud service that is well-isolated between the instance in a same machine. It guarantees performance to be consistent. Before you submit, you can test your code in the same environment to the contest machine. You will learn how to launch and use the instance in Amazon EC2. Follow the EC2 Tutorial.pdf in KLMS. The docker is already set up in the AMI so you can just launch the docker to use the same environment to your department machine.

There are credits that is consumed when a machine is up. There is fixed amount of credits, so you are allowed to use up to 50 hours. To track your usage, log your start time and termination time of the instance in the Google Spread Sheet in tab of your team number. You must terminate, not stop, the instance not to pay for it. So, you need to implement your code first in provided VM and test only in the EC2 instance. If you are using machine without logging, TA can shut your machine down manually.

Handin

Your final outcome should be a single tar file that contains 8 files and a report in pdf format. The name of the file should be [team number].tar. Upload your tar file to KLMS.

Reference

Ctypes - https://docs.python.org/3/library/ctypes.html

OpenBLAS - https://github.com/xianyi/OpenBLAS/wiki/

cuBLAS - https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf

AVX - https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

CUDA - https://developer.download.nvidia.com/books/cuda-by-example/cuda-by-example-sample.pdf