DAT278 Lab Manual

November 2023

Hari Abram, Mateo Vázquez Maceiras

Preface

One major pillar of sustainable computing is maximizing energy efficiency, i.e., reducing the amount of energy consumed to carry out your prescribed work. To start you journey in sustainable computing, you will form groups of three, and explore multiple ways to improve the energy efficiency of running image recognition application. To join a group, go to `Canvas > People > Lab Groups`, and add yourself to one of the groups. If all the groups are taken, or you cannot find a group partner, please contact the Teaching Assistants (TAs) for help.

There are four different labs for you to do, and five sessions, as shown in Table I. Each lab is designed to take one session, but the fifth session can be used as extra time. If you have time enough, you can aim to get bonus points by doing the bonus lab.

TABLE I: Lab Schedule

Date	Task
13th Nov	Lab 1: Lab Introduction & Programming Languages
20th Nov	Lab 2: Hardware Selection
27th Nov	Lab 3: Sparse Data
4th Dec	Lab 4: Frequency & Voltage
11th Dec	Extra Time + Bonus Lab

The labs work as follow: before each lab, you need to do a preliminary task, and reply the questions asked in the corresponding Canvas assignment. Please make sure to upload the answer to this preliminary task before coming to the lab, as the deadline for the assignment will be set at the time the corresponding session starts. By the start of the session, the TA will unlock in Canvas the resources required for the corresponding lab if needed. Once in the lab, follow the instructions given in this manual. During the lab session the TAs will ask you some questions, in a demo-like fashion. For that, for each lab a representative will be randomly selected among the members of the group (although we will make sure that everyone gets to be a representative). This representative will be the one in charge of responding to the TA's questions. After this questioning, please make sure that the TA has marked this demo as completed in Canvas. Finally, fill and submit the post-lab assignment in Canvas.

I. Lab 1: Lab Introduction & Programming Languages

General Matrix-Matrix Multiplication (GEMM) is one of the most extensively used kernels across multiple domains and applications. One such example are Convolution Neural Networks (CNNs), which can be used for tasks like image recognition. In this lab, you will focus on optimizing image recognition over the ImageNet dataset [1] using AlexNet [4], one of the most well-known CNN models.

TABLE II: Inference results for the provided images

#images	Statistic	Accuracy
1	Top 1	0%
	Top 5	100%
10	Top 1	20%
	Top 5	70%
100	Top 1	59%
	Top 5	85%

To run AlexNet, first make sure you have the right paths, as explained in the *README.md* file. Then you can run AlexNet with `python3 main.py`. When running this command, the program will do as many inferences (i.e., will recognize as many images) as set up in the constant `BATCH_SIZE`. We have provided 100 images in which you can infer in the *data* folder. The results for inferring these images with AlexNet are as shown in Table II.

To measure CPU power, we read the power and energy counters using Intel's RAPL. For that, IT has provided a binary that reads them and prints them to a file. To use this binary, run `sudo /chalmers/sw/sup64/phc/b/binh/rapl-read` in a terminal, and then launch AlexNet in a second terminal. When AlexNet finishes, go back to the terminal where power was being measured and kill the process with `Ctrl+C`. Then run `power_tools/energy_parser.py` with the start and end times you obtained from running AlexNet, and average power and total energy consumption will be printed.

A. Pre-Lab Assignment

Read the paper [6] and estimate how much energy (in percentage) you could save by moving the computation of GEMM from Python to C++.

In addition to that, once you get to the lab, start running AlexNet with batch size 1 as soon as you can with `python3 main.py`. The baseline version takes quite a lot of time to run, so you can leave that in the background while reading the code and understanding the problem. To set up the batch size, change the corresponding value in *main.py*.

B. Lab Assignment

As explained before, once you get to the lab try to get AlexNet running with batch size 1. The base version you were provided uses a manual implementation of GEMM developed in Python, and you will see the problems of it.

However, you do not need to run everything with Python. You offload parts to the more efficient C++ using Pybind11 [3]. In *src/cpp/gemm.cpp*, you have the template for doing so. Write your GEMM implementation in C++ there, and then build the library with `./compile_library.sh`. Then load this library in *gemm.py*, as use your C++ function instead of the Python one. How much energy are you saving with this change? Then, add parallelism in your C++ implementation with OpenMP. Are you saving energy with this new change? Why (not)? If you change the number of threads assigned, do you see any difference?

II. Lab 2: Hardware Selection

Modern computing systems are heterogeneous, with various hardware resources: we need to use them efficiently. In this lab, you will be offloading the workload to Nvidia GPU and observing the power and performance difference compared to the CPU.

A. Pre-Lab Assignment

Read the paper [5], how does the change in block size affect the power consumption of GPUs? and in this paper what is the the authors' reasoning for the observed changes in power consumption depending on block size?

B. Lab Assignment

Implement the GEMM using CUDA to run on Nvidia GPU, this was previously implemented in C++ using Pybind11 in Lab 1. You have to write your implementation of GEMM in this lab and verify that your implementation works, you can find the template in */src/cpp/gemmCuda.cu*. You can compile the Cuda code using the same method as in lab 1 with `./compile_library.sh`.

Once you implemented the GEMM kernel in GPU, change the BLOCK_SIZE to see if there is any difference in the power consumption depending on the block size. What is the highest and lowest power consumption you observe on the GPU? Compare these results with the CPU power consumption you obtained in lab 1. In lab 1, the OpenMP programming model is used to parallelize the GEMM kernel and comment on the power consumption using OpenMP parallel constructs and GPU power consumption.

III. Lab 3: Sparse Data

When inferring images, CNNs do not give us an exact answer. Rather, for all the different possibilities, i.e. labels it knows, the CNN model gives us an estimation of how confident it is that each label is the right one. Therefore, CNNs are complex models, with big memory and compute requirements, that "only" give us an estimated result. This raises the question: is it possible to simplify our CNN model, will still achieving a similar accuracy with the estimations provided? This is already under research, and one of the approaches proposed is pruning [2].

The idea of pruning is simple: there is a huge set of weights but, are all of them equally relevant? And if they are not, can we discard the ones that are less relevant without paying relevant penalties? Han *et al.* [2] discovered that it is possible to remove up to 90% of the weights with no major impact in the accuracy. If we have our weights stored as matrices, removing 90% of the weights will mean that said matrices will consist on 90% zeroes, i.e, they will be 10% dense.

When working with such matrices, one approach is to store them in different compressed formats, instead of in the traditional 2D array. These formats will spare us from storing the zeroes, thus saving space. Among the most widely used formats we can find Coordinate List (COO) and Compressed Sparse Row (CSR), with its counterpart Compressed Sparse Column (CSC). These formats store only the non-zeroes (NNZs), and have some auxiliary arrays to indicate where each of the NNZs was in the original matrix. Moreover, in order to multiply matrices in these formats, different algorithms that the ones used for the dense matrices shall be used. These algorithms are not as efficient as their dense counterpart. Therefore, sparse formats come with both a memory overhead and a algorithmic penalty. With this in mind, in this lab we aim to understand when the matrix multiplication starts to benefit from using sparse formats, and when we should stick with our dense arrays.

A. Pre-Lab Assignment

Look for more detail information regarding COO and CSR formats. Which auxiliary arrays do they use?

B. Lab Assignment

In *sparsity/sparsity.py* you have a list of dictionaries with the sizes of the matrices in each layer, as well as a list with different densities. Generate random matrices with the given sizes and densities with `rand` from the `scipy.sparse` library. Then, compute GEMM as $C = A * B + C$, with A, B, and C having sizes [M, K], [K, N], and [M, N] respectively. Do so with COO and CSR formats, as well as with dense matrices. You can use *scipy* to convert across different formats, including dense, with the `to____()` methods (e.g., `tocsr()`). Also, for this lab, you can use the available Python libraries for computing the multiplication (you can just use `C = A @ B + C`). How much energy can you save by using sparsity? (Use the same approach for measuring power and energy as in Lab 1) At which point is the trade-off between using dense and sparse matrices?

In addition to saving energy by skipping multiplications by 0, the sparse formats also allow memory savings. Measure how much memory you save compared to storing dense matrices, for different levels of density. How does this compare to the energy savings?

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[2] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[3] W. Jakob, J. Rhinelander, and D. Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[5] L. Lefèvre, A.-C. Orgerie, and D. Boughzala. A macroscopic analysis of GPU power consumption. In *COMPAS2019 : Conférence d'informatique en Parallélisme, Architecture et Système*, Anglet, France, June 2019.

[6] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva. Ranking programming languages by energy efficiency. *Science of Computer Programming*, 205:102609, 2021.