ECE5755 / CS5754 Modern Computer Systems and Architecture, Fall 2025
Assignment 5: GPT2

1. Introduction

This lab assignment asks you to put everything together to complete the implementation of gpt2 and optimize it. You are tasked with aggressively optimizing your implementation using the microarchitectural principles you've learned throughout the semester. You may use **any optimization techniques you wish**.

2. Materials

We provide the implementation of gpt2 in C in `gpt2.c` with a few lines commented out in the `block` function. Your task is to first complete the code and then optimize the code. When testing the code, using the following command to compile:

```
gcc gpt2.c -o gpt2 -lm
```

3. Objective

Your first objective is to complete the implementation of `gpt2.c`. You can use OpenAI's implementation of gpt2 as a reference. Reading `src/model.py` in their repo should be sufficient for this task. Here is a tutorial on gpt2 implementation, which might also be helpful.

For the second objective, you can choose to either optimize or analyze the performance of the code. If you choose to do optimization, aim for an optimization that achieves 2x faster speed than the `gcc gpt2.c` baseline without any compiler optimization.

If you choose to do analysis, you need to analyze all the functions in detail. You should be able to identify the bottlenecks of the program and apply the techniques you've learned to implement an efficient neural network. You can use any combination of optimizations that you learned in class, any techniques that you research on your own, and any of the techniques in the following Additional Techniques section.

4. Additional Techniques

Not all of the following techniques will necessarily improve your program's performance. The first four techniques listed (cachegrind, gprof, gcov, SIMD) will likely be useful so try them first; the other techniques may or may not improve performance (depending on applicability and implementation) and may or may not be worth the effort, but they are listed here as options for you to explore.

- Profile cache misses using **cachegrind**.

    - Reference: https://valgrind.org/docs/manual/cg-manual.html
- Profile your code using **gprof** to see the execution time breakdown
    - Reference: https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html
- Profile your code using **gcov** to see which functions are getting called the most often
    - Reference: https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
- Vectorize your code with **SIMD instructions** such as SSE, AVX, AVX2, AVX512 intrinsics
    - Reference: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#
- **Pipeline the forward pass**. Not all elements need to finish one stage of the forward pass before the next stage starts processing data. E.g. convolution output elements that are ready early can be processed by the linear layers before all convolution output elements are ready.
- **Embed assembly code**. Compilers have to be conservative about the assumptions that are made regarding what optimizations are allowed in order to guarantee correctness. You as the programmer know more about what is and what is not allowed and can optimize more aggressively.
    - Reference: https://gcc.gnu.org/onlinedocs/gcc/extensions-to-the-c-language-family/how-to-use-inline-assembly-language-in-c-code.html
    - Instruction set reference: https://cdrdv2.intel.com/v1/dl/getContent/671110
- **Inline functions** that are called often. Note that this can increase your code size if the inlined function is called from multiple places.
- **Align data structures** to cache line boundaries.
- Try **different compiler optimizations**. Here are the GCC ones: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. Keep in mind -O3 is not necessarily going to give you better performance compared to -O2; test it out as it varies case by case. You also don't have to use GCC as your compiler if you feel like another compiler can provide more performance. The most popular C compiler besides GCC is Clang.
- Speed up convolution for larger inputs using **FFT methods**
    - Reference: https://arxiv.org/abs/1312.5851
- Improve runtime complexity of matmul using **Strassen's Method**
    - Reference: https://www.geeksforgeeks.org/strassens-matrix-multiplication/

5. Deliverables

Please submit your assignment on Canvas by **Tuesday, Dec. 9 at 11:59 pm**. Additional 10 points will be deducted for each late day after you have used your 3 free late days for the whole semester. Note that we will **NOT** grade assignments that are submitted more than 3 days after the deadline. You are expected to submit your code, report, and scripts in a single zip file named **lab5.zip**.

The lab will be marked out of a **total of 100 marks.**

- **40 marks** for completion and correctness of the `gpt2.c` code
- **20 marks** for performance improvement and/or profiling. If doing performance improvement then must get 2x speedup for full credit. -5 for 1.75x; -10 for 1.5x; -15 for 1.25x; - 20 for no speedup
- **40 marks** for report (named **lab5_report.pdf**)
    - 20 marks for profiling and runtime data (present as graph or chart or table)
        - 10 marks for profiling and runtime data on baseline
        - 10 marks for profiling and runtime data on optimized version
    - 10 marks for explaining how you used profiling data to help optimize your program
    - 10 marks for discussion of optimization techniques (with at least 1 technique not covered in labs 2-4)