CS 410 Final Project

Text Titans - Heterogenous text inverting with GPUs

Andrew Vamos (avamos2@illinois.edu)
Misha Ryabko (mryabk2@illinois.edu)
Benjamin De Vierno (bid2@illinois.edu)

Background

The internet continues to expand with Cisco estimating 5.3 billion users for 2023, already up from 3.9 billion in 2018. [1] As such, the internet continues to grow at a more rapid pace, making it more challenging to index this information for retrieval. One such approach used by search engines is the inverted index (covered in this course), where a database is generated that maps from words to their location and frequency in documents. While not an algorithmically complex task in isolation, the expansive amount of text information available on the internet makes the process of generating an inverted index highly compute intensive. Here, we assess various methods to generate an inverted index, comparing them to a baseline of a simple Python implementation.

The motivation for this project comes from the paper by Silveira et al. in which they proposed a heterogeneous parallel architecture using CPU and GPU in a cluster. [2] Here, we sought to recreate their architecture shown in Figure 1 below. We originally sought to replicate the multithreaded parallelization demonstrated, but ran into complications and were not able to complete this prior to the end of the semester. Here, we compare performance of various inverted index implementations, including GPU-sorting as in Figure 1. Also, this paper used the *WT10G* text collection curated for research purposes distributed by the University of Glasgow, but a £500 fee is required for access to this text. [3] Instead, we wrote a script to generate pseudo-documents of representative term frequency using the Python NLTK Reuters Corpus, a dataset totalling 10,788 documents totaling 1.3 million words, and this process is included in the code deliverables.

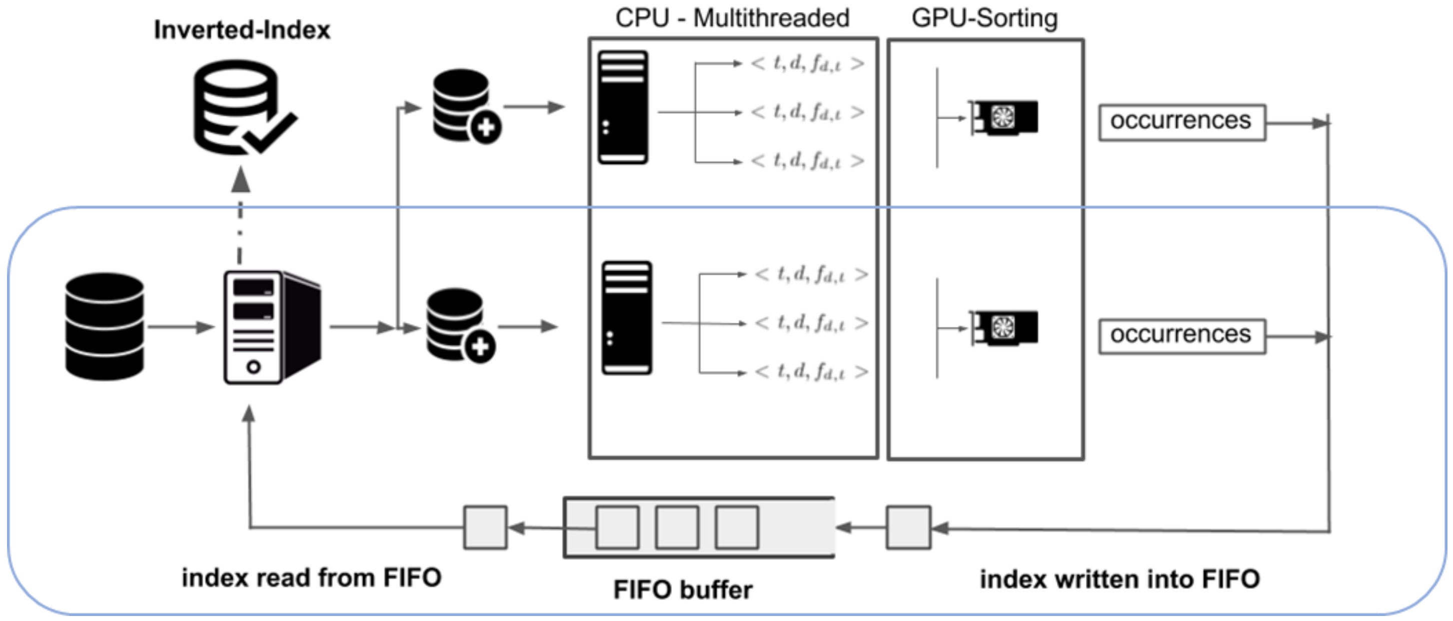

Figure 1. Architecture proposed by Silveria et al. Blue box demonstrates what we implemented.

Code Deliverables

All code is provided in our project git repo: https://github.com/Reisande/CS410_Project_2023

For ease of use for reviewers, select files for you to test have been uploaded to Google Colab. Note, you need to access it via your UIUC email. Also, if it seems like any other reviewers are running code blocks, please just review the output instead of interrupting anything. Drive/Colab Folder here, read only accessible with an illinois.edu email: https://drive.google.com/drive/folders/1QKTteA65yyul-cZVhEc8RCKiHSH9mAKa

Colab files you can easily run:

inverter_comparison_light.ipynb

numba_inverter_comparison_light.ipynb

generate_data.ipynb:

This notebook demonstrates the pseudo document generation process used to create the evaluation datasets. To be brief, the NLTK Reuters Corpus was used as a language model to sample from, generating pseudo-documents for inputs into the text inverter based on the word frequency within the corpus. Example term frequency and common words are demonstrated here in Figure 2 below.

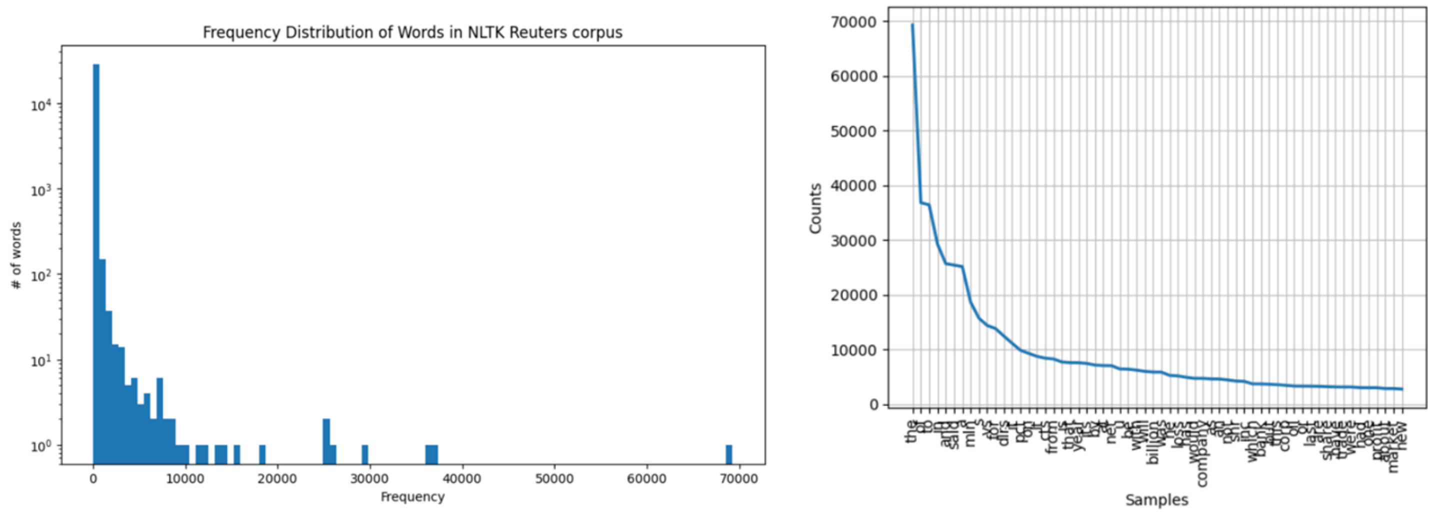

Figure 2. NLTK Reuters word frequency distribution and example common words. As you would expect, many stopwords and some abbreviations common to news articles from Reuters make up the most frequent term list. Note that no stop word filtering was done as we are only interested in speed of the inverter, not relevance of words it indexes.

inverter_comparison_light.ipynb:

This notebook demonstrates the text inversion process with a straightforward python implementation. Three of our smaller evaluation files (`term_doc_freq_10.txt, term_doc_freq_100.txt, term_doc_freq_1000.txt`) are inverted and the execution time is measured. Each of these files contains, 10, 100, and 1,000 documents of variable length between 100 and 10,000 words long generated from the NLTK Reuters document as demonstrated above. Three functions are used to perform the text inversion process. generate_doc_files processing sorts the dataset so it is organized by document ID. generate_term_files then organizes the triples by term ID, and then finally merge_indices rebuilds the sorted term IDs into a single list. The block at the end of the notebook allows you to see this process and get a sense of how long it takes to run via simple python implementation. Note, this notebook version was modified to remove file IO by using Python's dictionaries so it could be easily run on Google Colab.

numba_inverter_comparison_light.ipynb:

This notebook explores Numba which is a Just-In-Time (JIT) compiler for Python. Numba translates sections of code into machine code, which can improve performance. In order to support Numba compilation, some modifications to our original python code were made. This includes the addition @njit decorators for Just-In-Time compilation with Numba, replacing Python lists with Numba lists, removal of explicit type conversions (e.g., int()). A key limitation during performance testing was that we were not able to successfully use the out of the box 'sorted()', instead utilizing a basic selection sort algorithm instead. For parity during testing we updated the **inverter_comparison_light.ipynb** to utilize selection sort as well. Our findings from Numba, found that JIT compilation improved performance by 25% over the straightforward python implementation which utilizes selection sort algorithm. However, due to optimization using the Python sorted() function which is a variation of TimSort, this is about 10x less performant.

Inverted.py

This python file contains the invertext text generator using Python's sorted(), which is an implementation of the TimSort algorithm. [4] This sort algorithm is much more efficient than selection sort, so speed improvements are expected.

Inverted.cpp

This file contains the inverted text generator via CPP. The program starts by reading an input file line by line based off a command line argument, then converts them into an in memory data structure(triple). Then, it uses the std::sort method which uses either quick sort or merge sort(depending on the compiler), with a custom comparator lambda in order to do the 2 pass sorting required for the text generation. Finally, the results are written to a file. All file operations are via the fstream lib.

Inverted.cu

This file contains the inverted text generator via CUDA. It uses a third party library called Thrust, also distributed by Nvidia, to do a majority of the labor. [5] The Program starts by reading an input file line by line based off a command line argument, then converts them into an in memory data structure(triple). Then, it moves the data off the CPU into the GPU using thrust::copy in assignment. Then, it uses the Thrust::sort method which uses bitonic sort on the GPU, with a custom GPU lambda in order to do the 2 pass sorting required for the text generation. Finally, the program copies over the data back from the GPU into the CPU, and writes the results to a file. All file operations are via the fstream lib.

Usage documentation:

CUDA: for CUDA install the relevant cuda driver for your GPU (the author used an NVIDIA GeForce RTX 3060 Ti). Then follow the instructions here:

https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html# to install the cuda toolkit for the GPU. This code was compiled with nvcc version 12.3, V12.3.103 and Nvidia GPU driver 545. Then install Thrust via

```
git submodule init ; git submodule update
```

Then build using

```
nvcc ./inverted.cu -std=c++20 -I./thrust/ -I./thrust/dependencies/libcudacxx/
-I./thrust/dependencies/cub --extended-lambda -Xcompiler=-fno-gnu-unique
--gpu-architecture=compute_86 --gpu-code=compute_86,sm_86 -O3 -o ./cuda.out # (for SM 86, your GPU
will vary) ; ./cuda.out <file>
```

Replacing with the relevant information where necessary.

C++: build and run with:

```
g++ ./inverted.cpp -std=c++11 -O3 ; ./a.out <file>
```

Replacing <file> with your expected input file.

Local Python: build and run with:

```
Rm -rf terms/* docs/* index.txt ; python3 ./inverted.py <file>
```

Replacing <fiile> with your expected input file

Two Jupyter Notebooks have been created to support reviewers reproducing our findings. The links to those notebooks can be found below. There are two cell that have to be run, and the execution time for the text inversion process will be displayed as an output.

Results

The results were roughly in line with what we expected. The Python versions both were much slower than the C++ and Cuda versions, and the C++ version was a little slower than the CUDA version. The difference in the C++ and the Cuda version would likely be more pronounced when the GPU memory size is larger than the CPU memory pool. We did not run the python version on the 500 MB dataset due to time constraints.

In terms of correctness the Python, C++, and Cuda versions gave the same results, and were validated via the diff command. The timing for each program was done via the linux time utility, and we used the total experienced time since the GPU memory allocation was recorded as system time and the CPU allocations were recorded as user time.

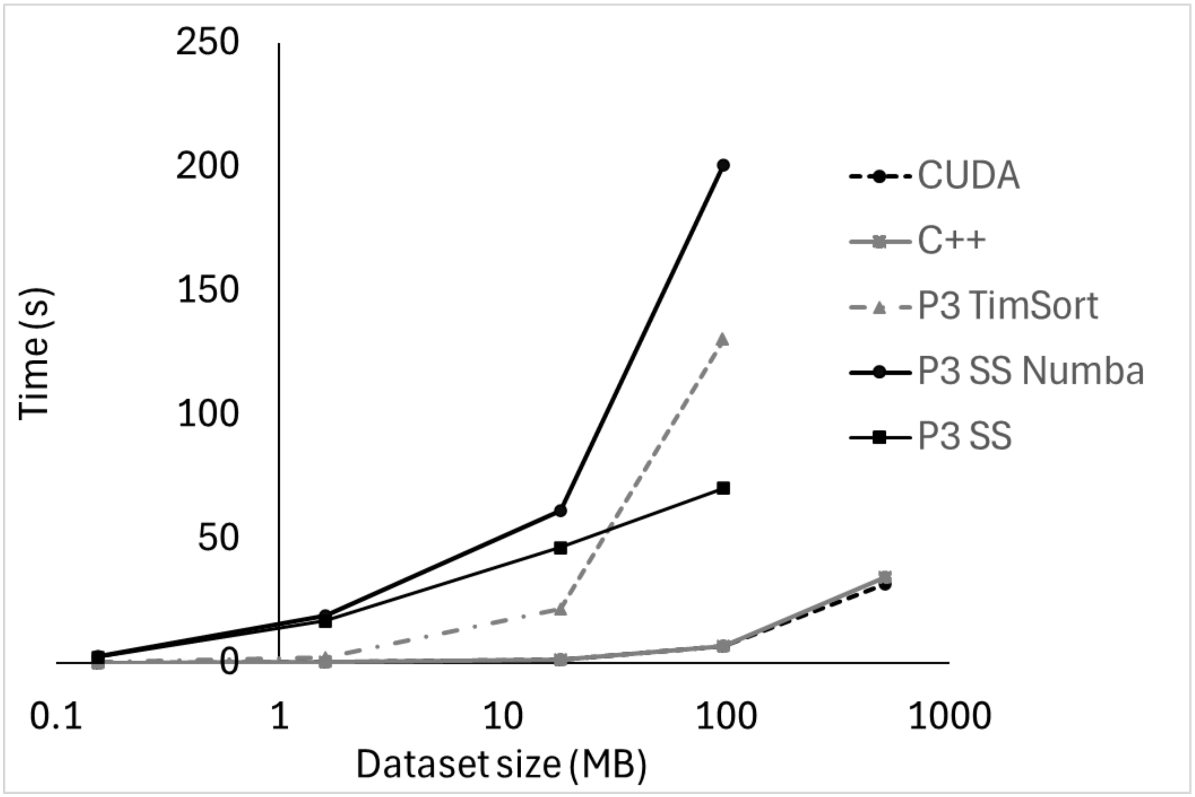

Figure 3. Results from the different implementations. For the Python selection sort versions, the 500 MB files were omitted due to lengthy runtimes.

Contributions

Misha Ryabko: Implemented and tested the CUDA version. Did technical work with regards to setting up the GPU. Implemented the local python version. Researched and disqualified the cloud implementation due to operating costs and usage quotas from Azure.

Andrew Vamos: created pseudo-document generator, researched/tested NVIDIA Triton + CUDA tutorials, Python selection sort, documentation

Benjamin De Vierno: Implemented and tested the Numba variations and selection sort

References

1. "Cisco Annual Internet Report (2018–2023) White Paper." Cisco, Cisco Systems, December 2023, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html#Executivesummary.
2. da Silveira, Tiago, et al. "Heterogeneous Parallel Architecture for Inverted Index Generation." 2019, doi:10.5753/wscad.2019.8664.
3. "Access to Data." Information Retrieval Group, University of Glasgow, December 2023, https://ir.dcs.gla.ac.uk/test_collections/access_to_data.html.
4. Peters, Tim. "Sorting." *Python-Dev Mailing List*, 20 July 2002, mail.python.org.

5. "Thrust." *CUDA Toolkit Documentation*, version 12.3, NVIDIA Corporation, December 2023, https://docs.nvidia.com/cuda/thrust/index.html.