High Performance Computing – Iteration 2(Progress Report)

Description: Implementing GPT4's variant of Byte Pair Encoding in CUDA C++.
Byte pair encoding is a way of sub word tokenization used by LLMs (large language models) to convert text into numerical tokens in a manner that is representative of linguistic nuances.

Planned Solution:
Input: Directory containing N files with text data
Output: Directory containing output, one corresponding file for each input file
Software: A compiled executable file that takes the input and output directory paths

Tasks Breakdown

- Data acquisition
  - Get raw text in ascii format from the internet.
    - Common crawl: https://commoncrawl.org/latest-crawl
    - Other datasets: https://github.com/niderhoff/nlp-datasets

- Data handling
  - Inference
    1. Create a fixed thread pool of size T
    2. Read files, split each file virtually into X chunks. Hence, at any point in time, X>=T
    3. In each thread, read the assigned file chunk and pass it onto the GPGPU for tokenization and wait until completion
       Chunk size is fixed (to be finalized based on observed performance, start with 4MB)
    4. Tokenize the text
       4.1 Load ***tokenToTokenIdHashMap*** – *use it as readonly*
       4.2 replace tokens with tokenId using map - ***cuda_kernel-INF1***
    5. In each thread, write the file chunk into a new file (output_dir/file_name-part_n.txt)
    6. After the file is processed, concatenate all file parts into single file

  - Training (Same inference steps except step 4)
    4.1 Create the following data structures as thread-safe and share across all threads
    - ***tokenIdToTokenOrderedMap***
    - ***bytePairFrequencyMaxHeap***

    4.2 Run the BPE algorithm, collect updates and update the common data structures at once
    4.3 Repeat for N times
    4.4 Save ***bytePairFrequencyMaxHeap, tokenIdToTokenOrderedMap*** to file
    4.5 Convert ***tokenIdToTokenOrderedMap*** into ***tokenToTokenIdHashMap*** and save it to file

- BPE Algo - Encoding:
  - Sentence splitter – ***cuda_kernel-BPE1***
    - Split each sentence using following regex
      GPT4_SPLIT_PATTERN = "'(?i:[sdmt]|ll|ve|re)|[^\r\n\p{L}\p{N}]?+\p{L}+|\p{N}{1,3}| ?[^\s\p{L}\p{N}]++[\r\n]|\s*[\r\n]|\s+(?!\S)|\s+"*
  - For each pair of adjacent letters create a frequency table – ***cuda_kernel-BPE2***
    - Merge the frequency tables and then update ***bytePairFrequencyMaxHeap***
  - Take the most frequent pair and assign them a new token and update the data structures
  - Replace all occurrences of byte pair with new token – ***cuda_kernel-BPE3***
  - Repeat until N iterations

- BPE Algo - Decoding
  - Load ***tokenIdToTokenOrderedMap***
  - Replace tokenIds with their corresponding byte-pair & count the number_of_substitutions done- ***cuda_kernel-BPE4***
  - Continue until number_of_substitutions==0

References:

- Reference implementation in python: https://github.com/karpathy/minbpe/blob/master/minbpe/
- Let's build the GPT Tokenizer: https://www.youtube.com/watch?v=zduSFxRajkE