CS F422 - Parallel Computing Assignment

Team Members:

- Suryavir Kapur (2022A7PS0293U)
- Ronit Dhansoia (2022A7PS0168U)

Assignment Overview

This project involves processing a large dataset of Amazon Electronics reviews using different parallel computing techniques (CUDA and OpenMP) and comparing their performance against sequential implementations. Tasks include finding top-rated products, performing sentiment analysis, and identifying elaborate reviewers.

Dataset

- **Reviews:** Amazon Electronics Reviews (5-core subset) - `electronics.json`
- **Sentiment Lexicon:** VADER Lexicon - `lexicon.txt`
- Downloaded using the `download.sh` script.

File Descriptions

Source Code

- `download.sh`:
  A bash script to download the Amazon Electronics review dataset and the VADER sentiment lexicon. It handles extraction and renaming.
- `cuda_toprated.cu` **(Task a)**:
  Reads reviews using RapidJSON (CPU), calculates average ratings per product (CPU), and then uses a CUDA kernel (Bitonic Sort) to sort the products by average rating on the GPU to find the top 10.
- `cuda_toprated_opt.cu` **(Task b)**:
  Similar implementation to `cuda_toprated.cu` for finding top-rated products using CPU aggregation and GPU sorting (Bitonic Sort). Uses CUDA events for more granular timing of GPU operations.
- `cuda_reviewanalysis.cu` **(Task d)**:
  Reads reviews (RapidJSON) and the VADER lexicon. Performs sentiment analysis *sequentially on the CPU* by looking up words in the lexicon. Minimal CUDA usage (device check/sync), primarily CPU-bound despite the `.cu` extension.
- `cuda_reviewanalysis_opt.cu` **(Task e)**:
  Reads reviews (RapidJSON) and lexicon. Implements CUDA-accelerated sentiment analysis using batch processing. Preprocesses text to word IDs (CPU), transfers data, and runs a CUDA kernel (`sentimentKernel`) for parallel score calculation.
- `c_elborate.cpp` **(Task g)**:
  A sequential C++ program using RapidJSON. Reads reviews, counts how many reviews with >= 50 words each reviewer has written, and identifies reviewers with at least 5 such "elaborate" reviews.
- `c_elaborate_openmp_cpu.cpp` **(Task h)**:
  Reads reviews sequentially like `c_elborate.cpp`. Then, uses OpenMP directives (`#pragma omp parallel for`, `#pragma omp critical`) to parallelize the *filtering* step (checking review counts >= 5) across CPU cores.

Results Files

- `results/a.txt`:
  Output from `cuda_toprated.cu`. Lists the top 10 products found and provides a timing breakdown for different stages (CPU read, CPU avg, GPU sort, etc.).
- `results/b.txt`:
  Output from `cuda_toprated_opt.cu`. Similar to `a.txt`, showing the top 10 products and timing breakdown, potentially with slightly different timing measurements due to CUDA event usage.

- `results/d.txt`:
  Output from `cuda_reviewanalysis.cu`. Shows the counts of positive, negative, and neutral reviews based on the CPU sentiment calculation, along with total execution time.
- `results/e.txt`:
  Output from `cuda_reviewanalysis_opt.cu`. Shows sentiment counts from the CUDA-accelerated version and total execution time. Note potential minor differences in counts compared to `d.txt`.
- `results/g.txt`:
  Output from `c_elborate.cpp`. Reports the number of elaborate reviewers found using the sequential method and provides timing for CPU reading/aggregation and filtering.
- `results/h.txt`:
  Output from `c_elaborate_openmp_cpu.cpp`. Reports the number of elaborate reviewers found using OpenMP parallel filtering and the time taken specifically for the parallel filtering step.

Other Files

- `.gitignore`:
  Specifies intentionally untracked files that Git should ignore, such as the large dataset files (`Electronics_5.json`, `lexicon.txt`).

Performance Comparison (Answers for c, f, i)

- (c) `cuda_toprated` vs `cuda_toprated_opt`:
  Comparing `results/a.txt` and `results/b.txt`, the total execution times are very similar (~156.7s vs ~154.8s). The GPU kernel execution time for sorting is nearly identical (~8ms). The "optimization" primarily involves more precise timing via CUDA events rather than a significant algorithmic change impacting overall runtime drastically for this workload. The bottleneck remains the CPU-based JSON parsing and aggregation (~155s).
- (f) `cuda_reviewanalysis` vs `cuda_reviewanalysis_opt`:
  Comparing `results/d.txt` and `results/e.txt`, the CPU-based version (`d.txt`, ~323.6s) is slightly *faster* than the CUDA-accelerated version (`e.txt`, ~329.6s). This suggests that for this task, the overhead associated with CPU preprocessing (batching, word-to-ID mapping), data transfers (Host-to-Device, Device-to-Host), and kernel launches in the `_opt.cu` version outweighs the benefits of parallel computation on the GPU compared to the simpler, albeit misnamed, CPU-bound approach in `cuda_reviewanalysis.cu`.
- (i) `c_elaborate` vs `c_elaborate_openmp_cpu`:
  Comparing `results/g.txt` and `results/h.txt`:
  - The sequential version (`g.txt`) takes ~225 seconds total, with the filtering step taking only ~121 milliseconds.
  - The OpenMP version (`h.txt`) parallelizes *only* the filtering step, which takes ~273 milliseconds (0.273 seconds) according to its output.
  - The vast majority of the time (~225 seconds) is spent on sequential reading/parsing in both versions.