COL380 A3

Sarthak Panda (2022CS51217)

March 2025

1 Brief Discussion on the Current Implementation

The current implementation returns a naive solution to addresses the given problem statement by ensuring that when the elements of the output matrix are written in row-major order, they appear in sorted order. To achieve this, a frequency array is created to store the count of each distinct element of input matrix. An exclusive prefix sum is then computed over this frequency array, where each prefix value p_i is defined as:

$$p_i = \sum_{j=0}^{i-1} a_j,$$

with a_j being the frequency of the element j. Finally, the matrix is filled in row-major order such that for each value i, the elements between the indices p_i and p_{i+1} in the output array are set to i.

2 Optimizations Performed

2.1 Optimization in Frequency Array Calculation

In the frequency array calculation, the input matrix is divided among CUDA blocks such that each block processes a chunk of the matrix. The number of blocks is computed as the ceiling of

$$\frac{\text{num\_elements in matrix}}{1024}$$

Each block is launched with 1024 threads & Each thread within a block is responsible for one position of the matrix. The corresponding frequency count is updated using atomic increments to avoid race conditions. This approach effectively parallelizes the counting step while handling variable matrix sizes and ensuring correctness.

2.2 Optimization of Prefix Sum Calculation

2.2.1 Blelloch Prefix Sum Algorithm

The prefix sum (or scan) is computed using the Blelloch algorithm, a parallel algorithm that operates in $O(\log n)$ time using two main phases: the upsweep (reduce) phase and the downsweep phase. The pseudocode for these phases is given below.

Upsweep Phase:

Algorithm 1 Blelloch Upsweep

1: **for** $d = 0$ **to** $\log_2(n) - 1$ **do**
2: **for all** indices i in parallel such that $i \mod 2^{d+1} = 0$ **do**
3: `data[`$i + 2^{d+1} - 1$`] += data[`$i + 2^d - 1$`]`
4: **end for**
5: **end for**

Downsweep Phase:

Algorithm 2 Blelloch Downsweep

```
data[n-1] = 0
for d = log2(n) − 1 downto 0 do
    for all indices i in parallel such that i mod 2^(d+1) = 0 do
        temp = data[i + 2^d − 1]
        data[i + 2^d − 1] = data[i + 2^(d+1) − 1]
        data[i + 2^(d+1) − 1] += temp
    end for
end for
```

2.2.2 Cyclical Prefix Computation in a Single Block

The prefix sum computation is executed on a per-matrix basis using a single CUDA block. This choice makes common case efficient which utilizes the fact that the frequency array size (after padding to the next power of two) is relatively small. By using one block, the algorithm can perform efficient synchronization (using `__syncthreads()`) during both the upsweep and downsweep phases. This minimizes inter-block communication overhead and simplifies memory management within the kernel.

2.3 Optimizations in Write-back

The write-back phase leverages the computed prefix sum to reconstruct the sorted matrix. In this phase, each thread in the CUDA kernel processes a subset of the prefix array.

- Each thread reads the starting and ending indices for a given value from the prefix sum array.
- It then writes the corresponding value into the positions of the matrix between these indices.

This approach fills the matrix (in 1-D array i.e. flattened in row major way) in a sorted order in a parallelized approach.