Sparse Matrix-Vector Multiplication (SpMV)

Objective: In this assignment, you will optimize the performance of Sparse Matrix-Vector Multiplication (SpMV). You will apply your optimisations (you can assume that you have unlimited time for this), measure the performance of the SpMV with the modified matrix, and compare the results. You are not allowed to change other parts of the code.

Instructions: The provided C++ code reads a sparse matrix from a binary file using the Compressed Sparse Row (CSR) format - you will add your code to the specified place.

- The matrix is then used to perform SpMV over multiple iterations.
- Your task is to modify the matrix/vector structures such that when the same SpMV routine runs, it produces the same result but faster.

Sparse matrices are matrices in which most of the elements are zero. Storing these matrices in a typical 2D array wastes memory, so we use representations like CSR.

Compressed Sparse Row (CSR) Format: CSR is one of the most common formats for storing sparse matrices, as it efficiently compresses the matrix by only storing the non-zero elements and their positions. It uses three arrays:

- **`row_ptrs`**: This array has `n + 1` elements (where `n` is the number of rows). Each entry `row_ptrs[i]` indicates the index in the `values` and `col_ids` arrays where the non-zero elements for row `i` start. The last element of `row_ptrs` (`row_ptrs[n]`) equals the total number of non-zero elements, `nnz`.
- **`col_ids`**: This array has `nnz` elements. Each entry `col_ids[j]` stores the column index of the non-zero value located at `values[j]`.
- **`values`**: This array has `nnz` elements. Each entry `values[j]` contains a non-zero value from the matrix.

Example: Consider the following 4x4 sparse matrix:

```
10  0   0   0
 0  0   3   0
 0  5   0   0
 7  0   9   1
```

The CSR representation would be:

- `row_ptrs = [0, 1, 2, 3, 6]`
    - Row 0 has 1 non-zero element starting at index 0.

    - Row 1 has 1 non-zero element starting at index 1.
    - Row 2 has 1 non-zero element starting at index 2.
    - Row 3 has 3 non-zero elements starting at index 3.
    - The last entry (`row_ptrs[4] = 6`) is the total number of non-zero elements.
- `col_ids = [0, 2, 1, 0, 2, 3]` – The column indices of the non-zero values.
- `values = [10, 3, 5, 7, 9, 1]` – The actual non-zero values from the matrix.

Let vector be **`x`**: `[1,2,3,4]`. Then one **SpMV** iteration is **(`y = A * x`)**:

1. **Row 0**: `y[0] = 10 * x[0] = 10 * 1 = 10`
2. **Row 1**: `y[1] = 3 * x[2] = 3 * 3 = 9`
3. **Row 2**: `y[2] = 5 * x[1] = 5 * 2 = 10`
4. **Row 3**: `y[3] = 7 * x[0] + 9 * x[2] + 1 * x[3]`

   `= 7 * 1 + 9 * 3 + 1 * 4 = 7 + 27 + 4 = 38`

The binary file **`/data/sparse_matrix.bin`** stores the matrix data. All the input read code is there. You will not modify this part.

- This explanation is just for clarity:
    - First, an integer `n` represents the number of rows/columns in the matrix.
    - Next, an integer `nnz` represents the number of non-zero values.
    - `n + 1` integers for `row_ptrs` (CSR row pointers).
    - `nnz` integers for `col_ids` (CSR column indices).
    - `nnz` doubles for `values` (non-zero values of the matrix).

You need to add new code to optimize the SpMV operation by only changing the matrix structure. Then you need to check correctness by ensuring that the values printed values are correct and measure the time taken by the optimized SpMV implementation. You need to write a report (in PDF format) that includes, (1) your approach to optimizing the SpMV operation, (2) a performance comparison of the original and optimized execution with explanations and (3) any challenges you faced during the process. Use **perf** or other tools to explain what was the bottleneck and what your optimizations fixed.

Do not change the output structure in the codes.

Submit your source code files (`.cpp`) and the report (`.pdf`) in a zip file to SUCourse. Name the zip file with **name_surname.zip**

Happy coding!