COL380 Assignment 4

(2024-25 Sem II)

You are given N sparse matrices A_i, $0 \leq i < N$ (of the kind given in Assignment 1 – You may employ the generate_matrix function from assignment 1 for testing as well). Your task is to multiply this sequence of matrices: produce $\prod A_i$ in the same format as the input matrices (all integers are 64-bit long long and all operations are modulo maximum long long). This sequence multiplication is to be parallelized across multiple nodes (< 8), each with multiple CPU cores (up to 16) and 1 GPU. Combine MPI, OpenMP, and CUDA (same versions as you have used in the earlier assignments).

You will need to implement a CUDA kernel for multiplying two sparse matrices. You may re-use the OpenMP parallelization of matrix multiplication. You will distribute these pair-wise multiplications on all available MPI processes.

The command line of your program has one input string: the path of a readable folder, call it `F`. Folder `F` will contain N+1 files, named `matrix1` .. `matrixN`, and `size`, respectively. The file `F/size` has a two integers (separated by white-spaces). The first denotes N and the second denotes *k*, the block size used for all matrices in this folder. Each file `matrix`$_i$ has the following format:

```
<Height:int> <Width:int>
<Number of non-zero blocks: int>
For each non-zero block:
<Block starting row number:int> <Block starting column number:int>
For each non-zero block (in the same order as above)
  k*k <value:int>
```

Correct matrix files ensure that `<Width>` value in `matrix`$_i$ equals `<Height>` value in `matrix`$_{i+1}$. It is possible (if rare) that a matrix's width or height is not divisible by k – in that case, the last block will have fewer than *k* elements.

The output will be a single matrix produces in the execution directory with the name `matrix` in the same format as input matrices. (Note that `F` may be read-only – do not attempt to write there.)

For performance tuning, expect input matrix sizes to be around s*1024 (with a small single digits s). N may be up to a few hundred. The value of *k* would be small (e.g., 4, 16, 32). Fewer than 10% of the values in each matrix would be non-zero.

Note that finding the most efficient order of multiplications in the parallel context is a hard problem. Consider greedy heuristics to reduce the number of operations. For example, if multiplying a pair of consecutive matrices produces fewer non-zero blocks, it may be multiplied first. Remember, the time to compute the heuristic ordering is included in the total multiplication time.

Submission:
Submit any number of source files, but must also include a `Makefile`, which produces an executable `a4`. The tester will call `make`, followed by `a4 foldername` (of course, with appropriate `mpirun`).

In addition to the source file, also submit a report in file `report.pdf` (only for part 2) explaining your parallelization strategy. Include your performance analysis in the report itself. (No csv files are required.) The total execution time will include file IO times (and will be determined by the slowest MPI process). You may also separately analyze the relative performances of your CUDA and OpenMP parts.