COL380 Assignment 4

REPORT

Akshansh Chahal (2014CS10423)
Amaresh Kumar (2014CS10425)

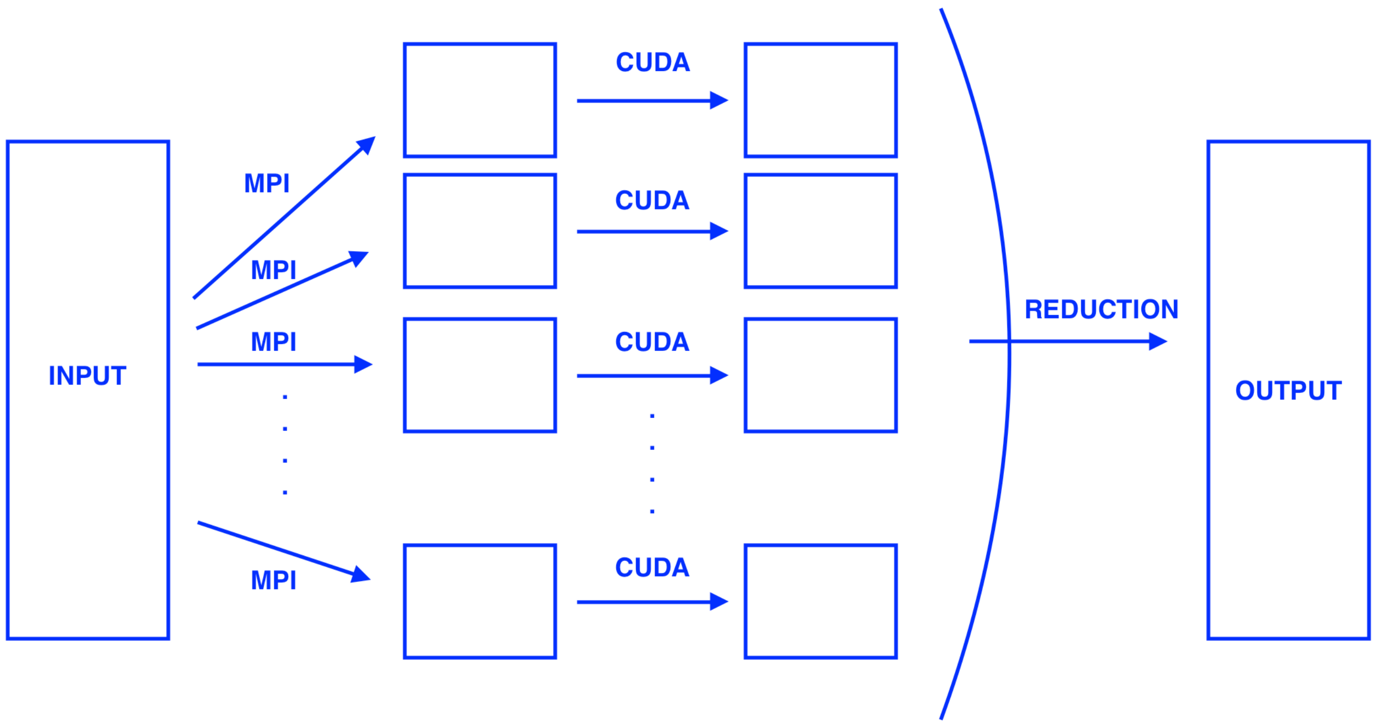
Parallel Algorithm

First of all we obtain the sparse matrix and the vector from input file into our main variables, while adhering to the standard input/output format provided to us.
Then we distribute this data to all the MPI nodes, based on the rank or id of the nodes.
Each node sends its data to its own CUDA device, by using cudaMalloc() and cudaMemcpy() etc

Now there is a __global__ function called add().
It takes 4 arguments :
1. Input matrix
2. Vector (same to everyone)
3. Output matrix
4. No. of elements in the input Matrix

It multiplies the part of matrix it has been assigned to, by the vector and adds, combines those parts appropriately and stores in the output array provided to it.

Post this the MPI reduction is used to get the final answer of the product of matrix and vector.



DESIGN DECISIONS

Made a struct with 3 elements

1. row
2. column
3. value

Now the input matrix is an array of these structs, and hence the output array which each CUDA device prepares locally is also an array of these structs.

Other design choices include the type and number of arguments sent to the __global__ function add() which had to be executed on the device.

The way we did the data scattering and reduction using MPI calls to distribute and combine the data computed by all the nodes was also a major design decision.

PARALLELISATION STRATEGY

Most of this is explained above within the algorithm itself.
The data was divided amongst different nodes in MPI, now it executes in SPMD method hence all the nodes will work in parallel. Each node computes the product on CUDA devices, hence that work also happens in parallel.
So enough parallelism in the algorithm design, except in the starting when we need to take the input from the input file, can't be done in parallel and since in some cases there is huge dataset then the major portion of the time consumed is taken in taking the input only, rather than computing the product of the matrix and vector.

LOAD BALANCING

If the data is perfectly distributable amongst the nodes then the load on each node is equal.
But when it not then the extra elements, that is the remainder left when total elements divided by the no. of nodes are added one by one extra in the nodes starting from first node.
Hence in that case too the load is almost equally split amongst all nodes.

It may also depend the numbers given to a node if they are variable size. Some nodes may take more amount of time if the integers they have are very large as compared to other nodes.