This is the documentation of the source code of the paper
"A system architecture for parallel analysis of flux-balanced metabolic pathways"

For simplicity there are 3 files:

1- kernel.cu => contains the "main" function and the basic functions used in "main"
2- structs.h => contains all used structures
3- funcs.cpp => contains all inside functions of basic functions

The following variables should be set by user in the file before compiling and running the program (This will be improved in an upcoming versions by a better user interface)

in kernel.cu ::

```
inFile1 = "inputs\\cho.txt";        // input stoichiometric matrix in "input" folder
outFile1 = "outputs\\cho.txt";      // output :: set of candidates and their fluxes after
run, the last column states that the row is an EFM or not

int firstRec = 0;                   // the index of the boundary reaction
```

in structs.h ::

```
#define       NumberOfMETABOLITEs       12       // number of metabolites
#define       NumberOfREACTIONSs        18       // number of reactions
#define       NumberOfREACTIONSsPlus    19       // number of reactions+1
#define       NumberOfCandidates        48       // number of EFM candidates
#define       MaxNumOfRecInOut          4        // Maximum number of input/output
reactions of a metabolite
#define       MaxNumOfMetInOut          4        // Maximum number of input/output
metabolites of a reaction

#define       MaxDepth                  4        // Maximum depth search to find
dependent reactions over a metabolite
```

The algorithm starts with the main function and the steps are as follows:

Step ONE: Parsing the given Stoichiometric matrix from the input file and store the Matrix in S
```
S = readFileS(*fin);
```

Step TWO: Creating the network structure based on the given S matrix
```
CreateNetwork_array(S);
```

Step THREE: running the main function for exploring EFMs
```
cudaStatus = EFM();
```

Step FOUR: Printing out candidates in the output file
```
printEFMs(*fout);
```

As Step THREE is the main step of the algorithm we go through this step::

Step THREE: running the main function for exploring EFMs

```
cudaStatus = EFM();
```

-- Setting cuda device parameters and preparing it:

```
cudaDeviceReset();
cudaStatus = cudaSetDevice(0);
cudaStatus = cudaMemGetInfo(&free_byte, &total_byte);        // check memory usage
```

-- Allocating GPU memory for network (input) and EFM candidates (output)

```
cudaStatus = cudaMalloc((void**)&d_network, sizeN);
cudaStatus = cudaMalloc((void**)&d_EFMs, sizeE);
```

-- Copying the required information to the allocated space in GPU from CPU

```
cudaStatus = cudaMemcpy(d_network, &network, sizeN, cudaMemcpyHostToDevice);
```

Then we call GPU three times with three different functions and three different thread and block number:

FIRST => To find dependent reactions as stated in Algorithm 2 of the paper

```
dim3 dimGrid0(NumberOfMETABOLITEs);           //each block == one metbolite
float x = MaxNumOfRecInOut; float y = MaxDepth;
int maxPathNumber = pow(x, y);
dim3 dimBlock0(maxPathNumber);
//for each metabolite create all possibilities on threads
depthFUNC<<<dimGrid0, dimBlock0>>> (d_network);
```

Second => To initialize first reactions in all EFM candidates

```
dim3 dimGrid1(NumberOfCandidates);
dim3 dimBlock1(NumberOfREACTIONSs);
MetaINIT<<<dimGrid1, dimBlock1>>>(d_network, d_EFMs);
```

Third => To run the META function on each metabolite until all candidates ready to report their result as stated in Algorithm 3 of the paper

```
dim3 dimGrid2(NumberOfCandidates);      //NumberOfMETABOLITEs = NumberOfBlocks
dim3 dimBlock2(NumberOfMETABOLITEs);    //NumberOfThread = NumOfCandidates
METAx<<<dimGrid2, dimBlock2>>>(d_network, d_EFMs);
```

-- Copying the required information from GPU to CPU

```
cudaStatus = cudaMemcpy(&EFMs[0], &d_EFMs[0],
sizeE,cudaMemcpyDeviceToHost);
```

-- deAllocating GPU memory

```
cudaFree(d_network); cudaFree(d_EFMs);
```

Details of the algorithms are provided in the manuscript.

Now you are ready to use the Code!! :D