Title: Parallel Computing using CUDA

Problem Definition: Vector and Matrix Operations- Design parallel algorithm to:

1. Add two large vectors
2. Multiply Vector and Matrix
3. Multiply two N × N arrays using n2 processors

Learning Objectives:

- Learn parallel decomposition of problems.
- Learn parallel computing using CUDA

Learning Outcomes:

I will be able to decompose problems into subproblems, to learn how to use GPUs, to learn to solve sub problems using threads on GPU cores.

Software Packages and Hardware Apparatus Used:

- Operating System : 64-bit Ubuntu 18.04
- Browser : Google Chrome
- Programming Language : C++, Python 3
- Jupyter Notebook Environment : Google Colaboratory

Concepts related Theory:

Dividing a computation into smaller computations and assigning them to different processors for parallel execution are the two key steps in the design of parallel algorithms. The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel, is called decomposition.

Tasks are programmer-defined units of computation into which the main computation is subdivided by means of decomposition. Simultaneous execution of multiple tasks is the key to reducing the time required to solve the entire problem.

Tasks can be of arbitrary size, but once defined, they are regarded as indivisible

units of computation. The tasks into which a problem is decomposed may not all be of the same size. In addition of two vectors, we have to add ith element from first array with ith element of the second array to get ith element of the resultant array. We can allocate this each addition to a distinct thread. Same thing can be done for the product of two vectors.

There can be three cases for addition of two vectors using CUDA.

1. n blocks and one thread per block.
2. 1 block and n threads in that block.
3. m blocks and n threads per block.

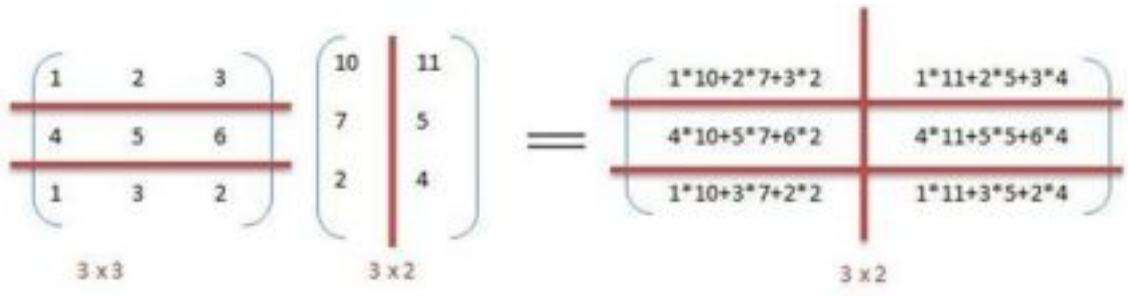
In addition of two matrices, we have to add (i,j)th element from first matrix with (i,j)th element of the second matrix to get (i,j)th element of resultant matrix.. We can allocate this each addition to a distinct thread.

There can be two cases for addition of two matrices using CUDA.

1. Two dimensional blocks and one thread per block.
2. One block and two dimensional threads in that block.

Same cases can be considered for multiplication of two matrices.

Parallelism in matrix:



A collection of blocks from a grid (1D, or 2D) – Built-in variable gridDim specifies the size (or dimension) of the grid.

– Each copy of the kernel can determine which block it is executing with the built-in variable blockIdx.

Threads in a block are arranged in 1D, 2D, or 3D arrays.

Built-in variable blockDim specifies the size (or dimensions) of the block.

– threadIdx index (or 2D/3Dindices) thread within a block – maxThreadsPerBlock: The limit is 512 threads per block

Algorithm:

Let S be the system set:

S = {s; e; X; Y; Fme;Ff;DD;NDD; Fc; Sc}

s=start state

e=end state

X=set of inputs

X = {X1,X2,X3,X4,X5,X6}

where X1, X2, X3 = Arrays

where X4, X5, X6 = Matrices

Y= Output Set

Y = {Y1,Y2,Y3} where

Y1 = X1 +

X2 Y2 = X3 x

X4 Y3 = X5 x

X6

Fme is the set of main functions

Fme = {f0} where

F0 = output display function

Ff is the set of friend functions

Ff = {f1,f2,f3,f4,f5,f6,f7,f8} where

f1 = Kernel Function to add Arrays

f2 = Kernel Function to multiply array and matrix

f3 = Kernel Function to multiply 2 matrices

f4 = Host Function to initialize array/matrix

f5 = Host Function to display array/matrix (Overload insertion operator) f6 = Host Function to add Arrays

(Overload + operator)

f7 = Host Function to multiply array and matrix (Overload * operator)

f8 = Host Function to multiply 2 matrices (Overload * operator)

DD = Deterministic Data

Input Array X1,X2,X3

Input Matrices X4,X5,X6

NDD=Non-deterministic data

No non deterministic data

Fc =failure case:

No failure case identified for this application

Test cases:

Sr No	Input	Actual Output	Expected output	Status
1.	Vector Addition: A = [3,6,7,5,3,5,6,2] B = [9,1,2,7,0,9,3,6]	C = [12,7,9,12,3,14,9,8]	C = [12,7,9,12,3,14,9,8]	Pass
2.	Vector and Matrix Multiplication Vector = [7,9,2] Matrix : [1, 3, 4, 8] [6, 10, 3, 3] [9, 10, 8, 4]	Result = [79,131,71,91]	Result = [79,131,71,91]	Pass

Conclusion:

We have successfully divided problems into subproblems, learnt how to use GPUs and learnt how to solve sub problems using threads on GPU cores.