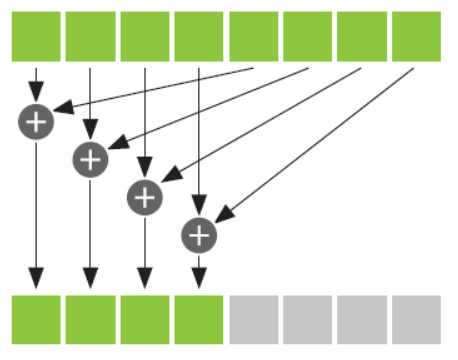
**BCSCCS705R02- Parallel and Distributed Systems Lab**

**Lab Report**

**Experiment-1: Compute Dot Product of two vectors using CUDA Programming.**

* Dot Product is a reduction from vector to scalar. 
* **Example:**

c= (a0, a1, a2, a3) ∙ (b0, b1, b2, b3)

c = a0 b0 + a1 b1 + a2 b2 + a3 b3

* Perform parallel, pair wise multiplications and serial addition. Store each thread result in a shared memory Using ***“\_\_\_shared\_\_\_”.***Synchronize the threads to maintain the order of execution of threads using ***“\_\_syncthreads()”***before the serial addition. No thread executes instructions after ***“\_\_syncthreads ()”*** until all threads have reached the ***“\_\_syncthreads ()”.*** sum these pairwise products from a single thread. Launch ***add ()*** with N threads: **add<<< 1, N >>>()** Use threadIdx.x to access thread’s index. Use (threadIdx.x + blockIdx.x \* blockDim.x) to index input/output. N/THREADS\_PER\_BLOCK blocks and THREADS\_PER\_BLOCK threads. gave us N threads total.

**Experiment-2:**  **Compute the product of two matrices.**

* Matrix multiplication: **P = A.B**
* Map each data into thread using ***“row=blockIdx.x\*blockDim.x+threadIdx.x” and “col=blockIdx.y\*blockDim.y+threadIdx.y”.*** Matrix multiplication between two matrices ‘A’ and ‘B’ is done using the equation ***Px, y = 𝝨 Ax,,k\*Bk,y, for k=0,1,2,....width.***  kernel that implements the above logic is defined as ***“ \_\_global\_\_ void matrixMultiplicationKernel(float\* A, float\* B, float\* P, int width)”*** . Product is done in if condition to make sure that no extra threads don’t do any work. Launch ***“matrixMultiplication<<<blocksPerGrid,threadsPerBlock>>>(A, B, P, N);”***

**Experiment-3: Find the k-th largest element in an unsorted list of n elements initially distributed among p processors** .