E0 243: Programming assignment: Part 1

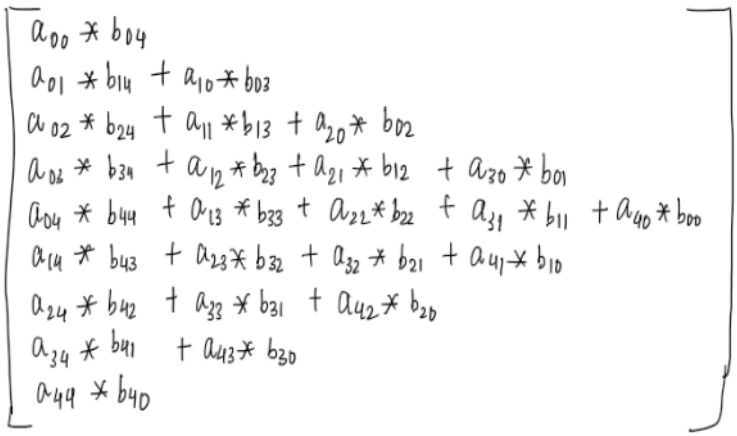
Name: Aman Sachan
S.R No.: 18094

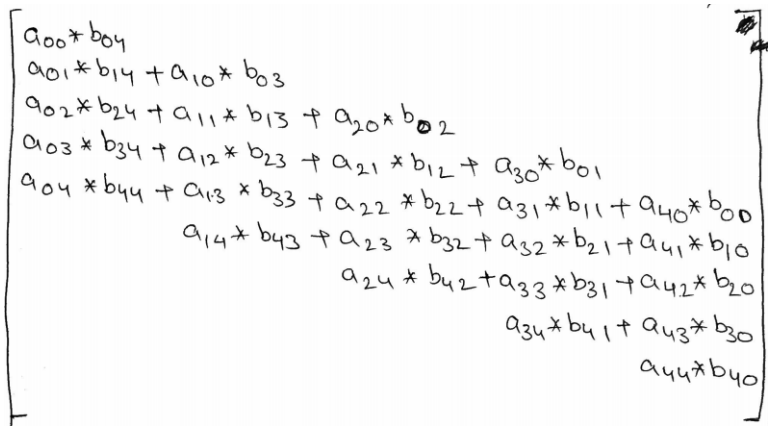
Due Date: 1^{st} Feb
Program: MTech-D

Part B (Diagonal matrix multiplication in CUDA)

I have used some CUDA API's like `cudaMalloc(), cudaFree(), cudaMemcpy()` that are equivalent to the C `malloc(), free(), memcpy()` respectively.

To understand the actual working of code, visualize the output matrix (as described in PartA) as :





Using `cudaMalloc()` we have allocated the memory for matrix A, B and output (device memory). After that using `cudaMemcpy()` we have transferred the initial data of the matrices (i.e, from host) to the newly allocated memory (to device) as well as the computed results by the thread (i.e, from

device) to the output matrix (to host). And atlast `cudaFree()` is used to free the memory allocated for GPU processing (device memory).
I modified the code in such a way that for every index of output a thread is assigned and it is computing the complete result for that particular index. So, in total there will be $2 * N - 1$ threads because output matrix is of $2 * N - 1$ size. I also ran the code with different threads per block and result is optimal in 256.

The below code is executed by every thread :

```
__global__ void gpuMultiply(int n, int *A, int *B, int *O){
        int i = threadIdx.x + blockDim.x*blockIdx.x;
        int k = i%n;
        int l = i/n;
        for(int j = l*k ; j <= min(i,n-1) ; j++){
                O[i] += A[j*n + i-j] * B[(i-j)*n + n-j-1];
        }
        __syncthreads();
}
```

`__syncthreads()` is used to synchronize the threads. Integers k and l are used to identify the first row of matrix A used for producing the result of Output matrix.

We can further optimize the execution time by *tiling* (blocking) i.e, making the blocks and executing blocks by blocks such that miss rate will be decreased also we can use the concept of *loop unrolling* to optimize the code.