GPU - Programming
Homework - 6

Topic - Convolution filter
Code -

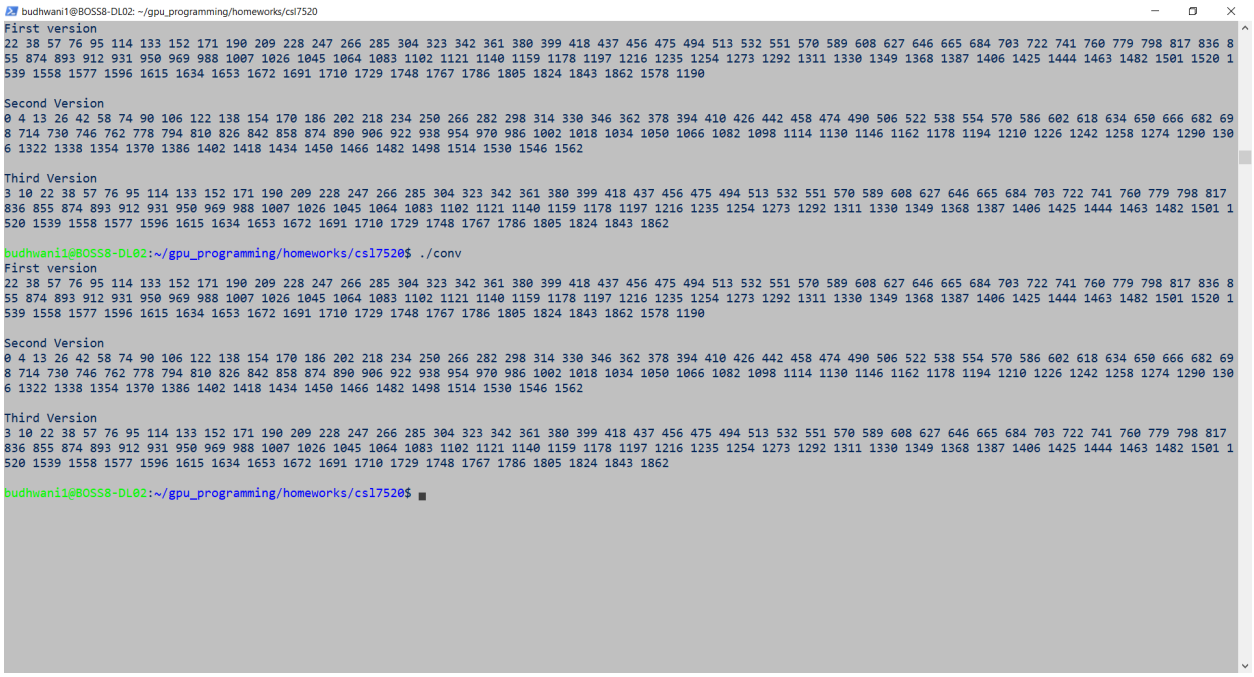
```
#include <stdio.h>
#include <cuda.h>


#define N               100
#define BLOCKSIZE       32

__global__ void init(int *input) {
        unsigned id = blockDim.x * blockIdx.x + threadIdx.x;
        if (id < N) input[id] = id + 1;
}
__global__ void print(int *output) {
        for (unsigned ii = 0; ii < N; ++ii)
                printf("%d ", output[ii]);
        printf("\n");
}

// defined in class by instructor
__global__ void convolutionV1(int *input, int *filter, int *output, int fsize) {
        unsigned id = blockDim.x * blockIdx.x + threadIdx.x;
        if (id >= N) return;

        //int *filteroutput = (int *)malloc(fsize * sizeof(int));
        int sum = 0;
        int halff = fsize / 2;
        int istart = id - halff, iend = id + halff + 1;
        int fstart = 0, fend = fsize;

        if (istart < 0) {
                fstart -= istart;
                istart = 0;
        }
        if (iend > N) {
                fend -= (iend - N);
                iend = N;
        }
```

```
	for (unsigned ii = fstart; ii < fend; ++ii) {
		// filteroutput[ii] = input[id + ii] * filter[ii];
		// sum+=filteroutput[ii];
		sum += input[istart + ii - fstart] * filter[ii];
	}
	output[id] = sum;
}

// defined by me using two loops for convolution and considers starting elements also thus output is not
//the same as version 1.
__global__ void convolutionV2(int *input,int *filter,int *output,int fsize)
{
	int tid = blockDim.x*blockIdx.x+threadIdx.x;
	int step = blockDim.x*gridDim.x;
	int convlength = N+fsize-1,n,k;
	for(n=tid;n<convlength;n+=step)
	{
		int sum = 0;
		int limit = min(N,n);
		for(k=0;k<limit;k++)
		{
			if(k<N && n-k<fsize)
			{
				sum+=input[k]*filter[n-k];
			}
		}
		output[n]=sum;
	}


}

// this version gives the same output as version 1 but provides 2 more outputs at the beginning and end.
__global__ void convolutionV3(int *input,int *filter,int *output,int fsize)
{
	int tid = threadIdx.x;
	int blocksize = blockDim.x;
	int i,j;
	for(i=tid;i<N;i+=blocksize)
	{
		int end,sum=0;
		if(i<fsize)
			end = i+1;
		else
```

```
            end = fsize;
        for(j=0;j<end;j++)
        {
            sum+=input[i-j]*filter[j];
        }
        output[i]=sum;
    }


}
int main() {
    int *input, *filter, *output1,*output2,*output3;
    int hf[] = {3, 4, 5, 4, 3};
    int fsize = sizeof(hf) / sizeof(*hf);

    if (fsize % 2 == 0) {
        printf("Error: Filter size (%d) is even.\n", fsize);
        exit(1);
    }
    cudaMalloc(&input, N * sizeof(int));
    cudaMalloc(&filter, fsize * sizeof(int));
    cudaMalloc(&output1, N * sizeof(int));
    cudaMalloc(&output2, (N+fsize-1)*sizeof(int));
    cudaMalloc(&output3, N*sizeof(int));
    cudaMemcpy(filter, hf, fsize * sizeof(int), cudaMemcpyHostToDevice);

    int nblocks = (N + BLOCKSIZE - 1) / BLOCKSIZE;
    init<<<nblocks, BLOCKSIZE>>>(input);

    printf("First version\n");
    convolutionV1<<<nblocks, BLOCKSIZE>>>(input, filter, output1, fsize);

    print<<<1, 1>>>(output1);
    cudaDeviceSynchronize();
    printf("\n");
    printf("Second Version\n");
    convolutionV2<<<nblocks, BLOCKSIZE>>>(input, filter, output2, fsize);

    print<<<1, 1>>>(output2);
    cudaDeviceSynchronize();
    printf("\n");
    printf("Third Version\n");
    convolutionV3<<<nblocks, BLOCKSIZE>>>(input, filter, output3, fsize);
```

```
    print<<<1, 1>>>(output3);
    cudaDeviceSynchronize();
    printf("\n");

    return 0;
}
```

Outputs -



Result:

Version 1 and 3 provide the same output of filters just a few elements more, and version 2 gives different outputs.