190905104

Parth Shukla

1. Write a program in CUDA to multiply two Matrices for the following specifications:

a. Each row of resultant matrix to be computed by one thread.

b. Each column of resultant matrix to be computed by one thread.

c. Each element of resultant matrix to be computed by one thread.

d. Perform matrix multiplication using 2D Grid and 2D Block.

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

__global__ void MatrixMulKernel_a(int *d_M, int *d_N, int *d_P, int wa, int wb)
{
    int ridA = threadIdx.x;
    int sum;
    for (int cidB = 0; cidB < wb; cidB++)
    {
        sum = 0;
        for (int k = 0; k < wa; k++)
        {
            sum += d_M[ridA * wa + k] * d_N[k * wb + cidB];
        }
        d_P[ridA * wb + cidB] = sum;
    }
}

__global__ void MatrixMulKernel_b(int *d_M, int *d_N, int *d_P, int ha, int wa)
{
    int cidB = threadIdx.x;
    int wb = blockDim.x;
```

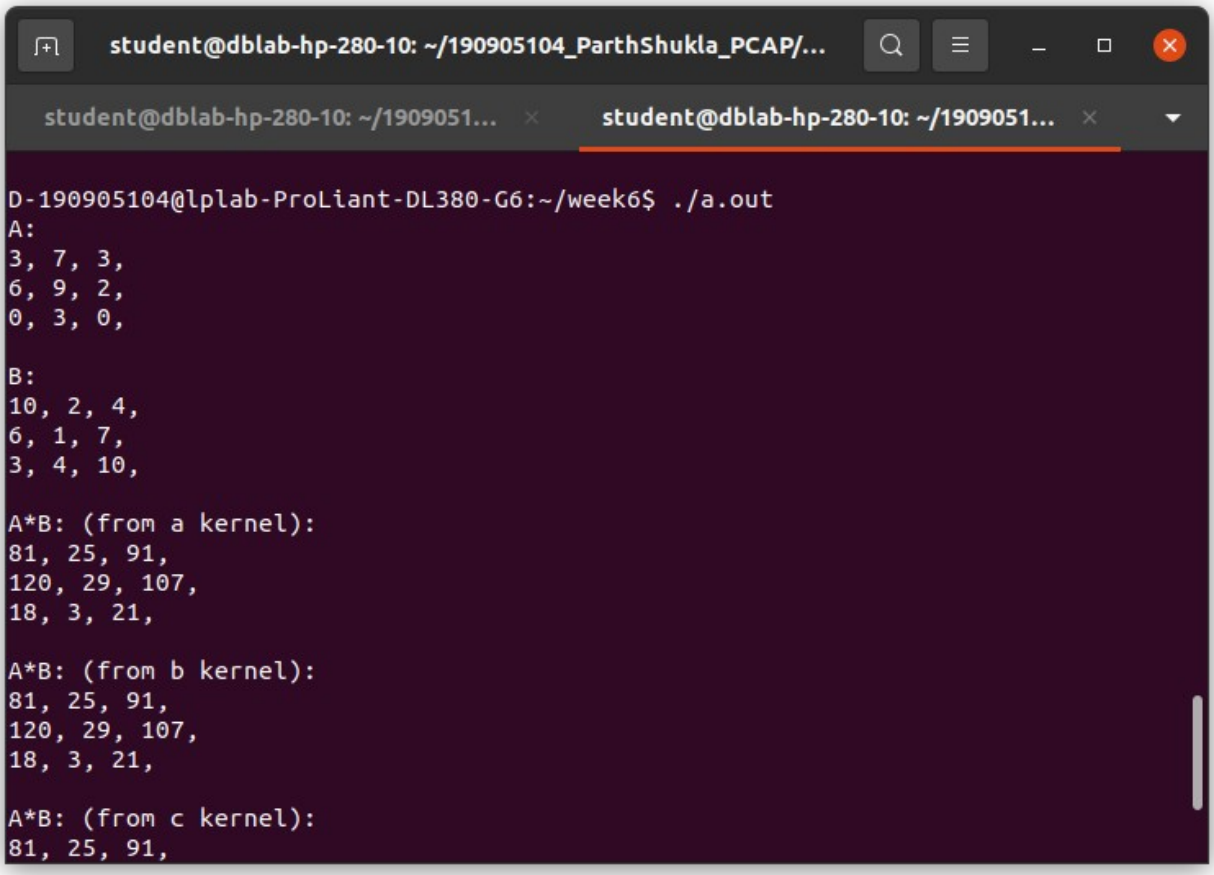
```
    int sum;
    for (int ridA = 0; ridA < ha; ridA++)
    {
        sum = 0;
        for (int k = 0; k < wa; k++)
        {
            sum += d_M[ridA * wa + k] * d_N[k * wb + cidB];
        }
        d_P[ridA * wb + cidB] = sum;
    }
}

__global__ void
MatrixMulKernel_c(int *d_M, int *d_N, int *d_P, int Width)
{

    int col = threadIdx.x;
    int row = threadIdx.y;
    int k = 0;
    for (int i = 0; i < width; i++)
    {
        k += A[row * width + i] * B[col + i * width];
    }
    C[row * width + col] = k;
}

__global__ void MatrixMulKernel_d(const int *a, const int *b, int *c, int m, int n, int
o)
{
    // row and col calculations
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    c[row * o + col] = 0;
```

```
    // calculating one element
    for (int k = 0; k < n; k++)
    {
        c[row * o + col] += a[row * n + k] * b[k * o + col];
    }
}

__host__ void clearMatrix(int *A, int width)
{
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < width; j++)
        {
            A[i * width + j] = 0;
        }
    }
}

__host__ __device__ void printMatrix(const char *string, int *A, int width)
{
    printf("%s\n", string);
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < width; j++)
        {
            printf("%d, ", A[i * width + j]);
        }
        printf("\n");
    }
    printf("\n");
}

void multiplyMatrix(int *h_A, int *h_B, int *h_C, int width)
```

```
{
    int *d_A, *d_B, *d_C;
    int size = width * width * sizeof(int);
    cudaMalloc((void **)&d_A, size);
    cudaMalloc((void **)&d_B, size);
    cudaMalloc((void **)&d_C, size);
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_C, h_C, size, cudaMemcpyHostToDevice);
    dim3 dimBlock(1, 1, 1);
    dim3 dimGrid(1, 1, 1);
    dimBlock.x = 1;
    dimBlock.y = width;
    dimBlock.z = 1;
    MatrixMulKernel_a<<<1, width>>>(d_A, d_B, d_C, width, width);
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    printMatrix("A*B: (from a kernel): ", h_C, width);
    clearMatrix(h_C, width);
    cudaMemcpy(d_C, h_C, size, cudaMemcpyHostToDevice);
    dimBlock.x = width;
    dimBlock.y = 1;
    dimBlock.z = 1;
    MatrixMulKernel_b<<<1, width>>>(d_A, d_B, d_C, width, width);
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    printMatrix("A*B: (from b kernel): ", h_C, width);
    clearMatrix(h_C, width);
    cudaMemcpy(d_C, h_C, size, cudaMemcpyHostToDevice);
    dimBlock.x = width;
    dimBlock.y = width;
    dimBlock.z = 1;
    MatrixMulKernel_c<<<(1, 1), (width, width)>>>(d_A, d_B, d_C, width);
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

```
    printMatrix("A*B: (from c kernel): ", h_C, width);
    clearMatrix(h_C, width);
    cudaMemcpy(d_C, h_C, size, cudaMemcpyHostToDevice);
    dimBlock.x = width;
    dimBlock.y = width;
    dimBlock.z = 1;
    MatrixMulKernel_d<<<dimBlock, dimGrid>>>(d_A, d_B, d_C, width, width,
width);
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
    printMatrix("A*B: (from d kernel): ", h_C, width);
    clearMatrix(h_C, width);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
}
int main()
{
    int *A, *B, *C;
    int width = 3;
    int size = width * width * sizeof(int);
    A = (int *)calloc(width * width, sizeof(int));
    B = (int *)calloc(width * width, sizeof(int));
    C = (int *)calloc(width * width, sizeof(int));
    int k = 1;
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < width; j++)
        {
            A[i * width + j] = rand() % 10;
            B[i * width + j] = rand() % 11;
            k++;
        }
    }
```

```
    printMatrix("A:", A, width);
    printMatrix("B:", B, width);
    multiplyMatrix(A, B, C, width);
    return 0;
}
```



2. Write a program in CUDA to read a sentence with equal length words. Count the number of times a given word is repeated in this sentence. (Use Atomic function).

Sample string: Pcap EEFM exam Pcap test Pcap

Word: Pcap

Given word repeated 3 times

```
#include <cuda.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
__global__ void word_count_kernel(char *str, char *key, int *word_indices, int *result)
```

```
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    // get idx'th word
    int si = word_indices[idx];
    int ei = word_indices[idx + 1];
    char word[100];
    int i = 0;
    for (i = 0; i < (ei - si - 1); i++)
    {
        word[i] = str[si + 1 + i];
    }
    word[i] = '\0';
    // compare word and key
    int i1 = 0;
    int i2 = 0;
    int is_equal = 1;
    while (word[i1] != '\0' && key[i2] != '\0')
    {
        if (word[i1] == key[i2])
        {
            i1++;
            i2++;
        }
        else
        {
            is_equal = 0;
            break;
        }
    }
    if (is_equal == 1)
    {
        atomicAdd(result, 1);
    }
}
int main()
{
    char str[100];
    char key[100];
    printf("Enter input String:\n");
    gets(str);
    printf("Enter a Key:\n");
    gets(key);
    int str_len = strlen(str);
    //     Inputed String : Hello Hi Hello Hi Hello
    //                      Key : hi\
// Total occurances of hi\ is 0 D -
    //                          190905087 @lplab - ProLiant - DL380 - G6 : ~$./ atom Enter input String :

    int key_len = strlen(key);
    int word_count = 0;
    for (int i = 0; i < str_len; i++)
    {
```

```
        if (str[i] == ' ')
        {
            word_count++;
        }
    }
    word_count--;
    int *word_indices = (int *)(malloc(word_count * sizeof(int)));
    int wi = -1;
    for (int i = 0; i < str_len; i++)
    {
        if (str[i] == ' ')
        {
            word_indices[++wi] = i;
        }
    }
    int result = 0;
    char *d_str;
    char *d_key;
    int *d_word_indices;
    int *d_result;
    cudaMalloc((void **)&d_str, str_len * sizeof(char));
    cudaMalloc((void **)&d_key, key_len * sizeof(char));
    cudaMalloc((void **)&d_word_indices, (word_count + 1) * sizeof(int));
    cudaMalloc((void **)&d_result, sizeof(int));
    cudaMemcpy(d_str, str, str_len * sizeof(char), cudaMemcpyHostToDevice);
    cudaMemcpy(d_key, key, key_len * sizeof(char),
               cudaMemcpyHostToDevice);
    cudaMemcpy(d_word_indices, word_indices, (word_count + 1) * sizeof(int),
               cudaMemcpyHostToDevice);
    cudaMemcpy(d_result, &result, sizeof(int), cudaMemcpyHostToDevice);
    word_count_kernel<<<1, word_count>>>(d_str, d_key, d_word_indices,
                                         d_result);
    cudaMemcpy(&result, d_result, sizeof(int), cudaMemcpyDeviceToHost);
    printf("Inputed String: %s\n", str);
    printf("Key: %s\n", key);
    printf("Total occurances of %s is %d\n", key, result);
    cudaFree(d_str);
    cudaFree(d_key);
    cudaFree(d_result);
    return 0;
}
```

```
D-190905104@lplab-ProLiant-DL380-G6:~/week6$ ./a.out
Enter input String:
Pcap EEFM exam Pcaptest Pcap
Enter a Key:
Pcap
Inputed String: Pcap EEFM exam Pcaptest Pcap
Key: Pcap
Total occurances of Pcap is 2
```