Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине: «Программирование графических процессоров»

Выполнили:

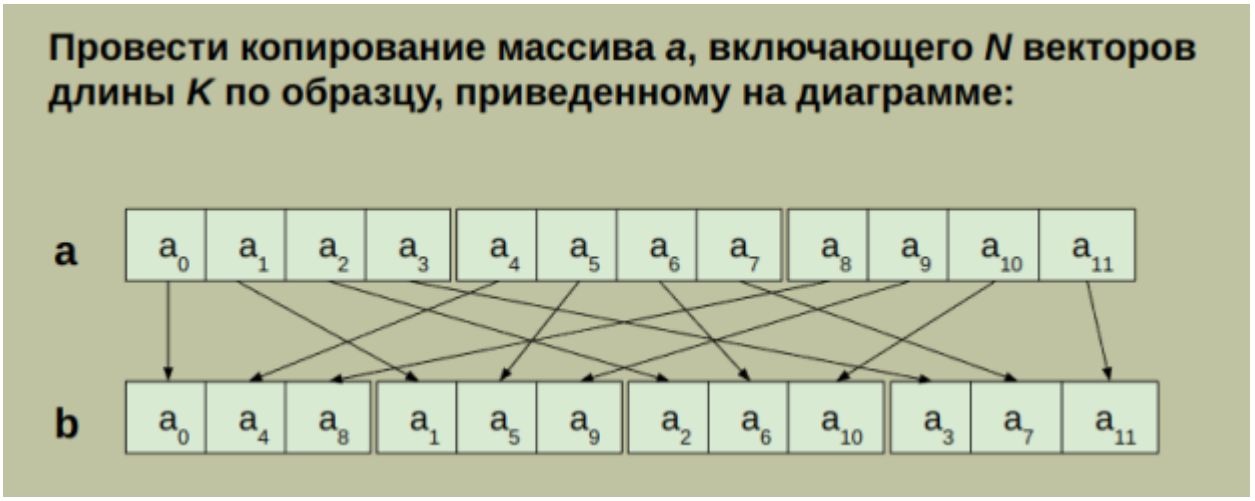
Студенты 3 курса группы ИП-211
Оганесян А.С.
Лацук А.Ю.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2025

Задание:

1. 

   С помощью метрик ncu:

   1.1. Определите время выполнения соответствующих ядер на GPU.

   1.2. Определите для обоих случаев пропускную способность при загрузке из глобальной памяти и при сохранении в глобальной память.

2. Эмулируйте недостаток регистров (большой размер локальных переменных в ядре) и, используя метрики ncu, определите использование локальной памяти.

Цель:

Выполнение работы:

1. По формулам из лекции напишем программу для транспонирования матрицы

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

#define N 3
#define K (1 << 2)

__global__ void gInit(float *a, float *b) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    if (i < N * K) {
        a[i] = (float)i;
        b[i] = 0.0f;
    }
}

__global__ void copyKernel(float *a, float *b) {
```

```
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < N * K) {
        int groupIdx = idx / K;
        int offset = idx % K;
        b[offset * N + groupIdx] = a[idx];
    }
}

int main() {
    size_t size = N * K * sizeof(float);

    float *A = (float*)malloc(size);
    float *B = (float*)malloc(size);

    float *d_A, *d_B;

    cudaMalloc((void**)&d_A, size);
    cudaMalloc((void**)&d_B, size);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N * K + threadsPerBlock - 1) /
threadsPerBlock;

    gInit<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B);
    cudaDeviceSynchronize();

    copyKernel<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B);
    cudaDeviceSynchronize();

    cudaMemcpy(A, d_A, size, cudaMemcpyDeviceToHost);
    cudaMemcpy(B, d_B, size, cudaMemcpyDeviceToHost);

    printf("First %d elements of the original and transposed
arrays:\n", N * K);
    for (int i = 0; i < N * K; i++) {
        printf("a[%2d] = %6.1f, b[%2d] = %6.1f\n", i, A[i], i,
B[i]);
    }
```

```
    free(A);
    free(B);
    cudaFree(d_A);
    cudaFree(d_B);

    return 0;
}
```

Листинг 1 – программа main.cu

Часть вывода профилировщика:

```
 copyKernel(float *, float *) (1, 1, 1)x(256, 1, 1), Context 1,
Stream 7, Device 0, CC 8.6
    Section: GPU Speed Of Light Throughput
    ----------------------- ----------- ------------
    Metric Name             Metric Unit Metric Value
    ----------------------- ----------- ------------
    DRAM Frequency                  Ghz         5.84
    SM Frequency                    Ghz         1.21
    Elapsed Cycles                cycle         2873
    Memory Throughput                 %         0.95
    DRAM Throughput                   %         0.64
    Duration                         us         2.37
    L1/TEX Cache Throughput           %        14.84
    L2 Cache Throughput               %         0.95
    SM Active Cycles              cycle        80.88
    Compute (SM) Throughput           %         0.04
    ----------------------- ----------- ------------
 gInit(float *, float *) (1, 1, 1)x(256, 1, 1), Context 1,
Stream 7, Device 0, CC 8.6
    Section: GPU Speed Of Light Throughput
    ----------------------- ----------- ------------
    Metric Name             Metric Unit Metric Value
    ----------------------- ----------- ------------
    DRAM Frequency                  Ghz         5.94
    SM Frequency                    Ghz         1.21
```

```
    Elapsed Cycles                    cycle         2394
    Memory Throughput                     %         1.32
    DRAM Throughput                       %         1.32
    Duration                             us         1.98
    L1/TEX Cache Throughput               %        23.22
    L2 Cache Throughput                   %         1.13
    SM Active Cycles                  cycle        51.69
    Compute (SM) Throughput               %         0.05
    ----------------------- ----------- ------------
```

Время выполнения ядра copyKernel - 2.37 us, a gInit 1.98 us

2. Посмотрим метрики dram__bytes_read.sum и dram__bytes_write.sum для просмотра пропускной способности:

```
  gInit(float *, float *) (1, 1, 1)x(256, 1, 1), Context 1,
Stream 7, Device 0, CC 8.6
    Section: Command line profiler metrics
    ---------------------- ----------- ------------
    Metric Name            Metric Unit Metric Value
    ---------------------- ----------- ------------
    dram__bytes_read.sum          Kbyte         2.94
    dram__bytes_write.sum          byte            0
    ---------------------- ----------- ------------

  copyKernel(float *, float *) (1, 1, 1)x(256, 1, 1), Context 1,
Stream 7, Device 0, CC 8.6
    Section: Command line profiler metrics
    ---------------------- ----------- ------------
    Metric Name            Metric Unit Metric Value
    ---------------------- ----------- ------------
    dram__bytes_read.sum          Kbyte         2.69
    dram__bytes_write.sum          byte            0
    ---------------------- ----------- ------------
```

3. Напишем программу, эмулирующую недостаток регистров:

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

#define N 1024
#define K 1024

__global__ void copyKernelWithRegs(float *a, float *b, int N,
int K) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    float reg[64];

    if (idx < N * K) {
        for (int i = 0; i < 64; i++) {
            reg[i] = a[idx] + static_cast<float>(i);
        }

        float sum = 0.0f;
        for (int i = 0; i < 64; i++) {
            sum += reg[i];
        }

        b[idx] = sum;
    }
}

int main() {
    float *a, *b;
    float *d_a, *d_b;

    size_t size = N * K * sizeof(float);

    a = (float *)malloc(size);
    b = (float *)malloc(size);
    for (int i = 0; i < N * K; i++) {
        a[i] = static_cast<float>(i);
    }
```

```
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);

    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N * K + threadsPerBlock - 1) /
threadsPerBlock;
    copyKernelWithRegs<<<blocksPerGrid, threadsPerBlock>>>(d_a,
d_b, N, K);

    cudaError_t error = cudaGetLastError();
    cudaDeviceSynchronize();
    if (error != cudaSuccess) {
        printf("CUDA error: %s\n", cudaGetErrorString(error));
    }

    cudaMemcpy(b, d_b, size, cudaMemcpyDeviceToHost);

    free(a);
    free(b);
    cudaFree(d_a);
    cudaFree(d_b);

    return 0;
}
```

Через nvprof получим метрики об использовании локальной памяти:

```
Local Load Transactions | 0 |
Local Store Transactions | 0 |
```

Вывод:

В ходе лабораторной работы было исследовано использование глобальной памяти GPU с помощью CUDA. Мы анализировали время выполнения ядер, пропускную способность при работе с памятью и влияние нехватки регистров на использование

локальной памяти. С помощью инструментов профилирования удалось лучше понять, как эффективно управлять памятью на GPU и как это влияет на общую производительность.