МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и цифровых технологий

Кафедра КБ-14 «Цифровые технологии обработки данных»

Платформы анализа больших данных

Лабораторная работа 1

Вариант 6. Применить гамма-коррекцию к изображению

Выполнил:

Студент группы БСБО-09-22

Шутов Кирилл Сергеевич

Проверил:

Кашкин Евгений Владимирович

Москва, 2025

Постановка задачи

В рамках лабораторной работы необходимо реализовать алгоритм гамма-коррекции изображения на GPU с использованием технологии CUDA.

Описание кода и выполненных действий

1. Время работы kernel-функции, конфигурация пространства потоков, количество используемых регистров

Время выполнения: 40,06 мкс

Конфигурация пространства потоков:

- Grid Size: (53, 41, 1) — общее количество блоков в сетке по осям X, Y, Z
- Block Size: (16, 16, 1) — количество потоков в блоке по осям X, Y, Z

Количество регистров на поток: 22 регистра

2. Арифметическая интенсивность

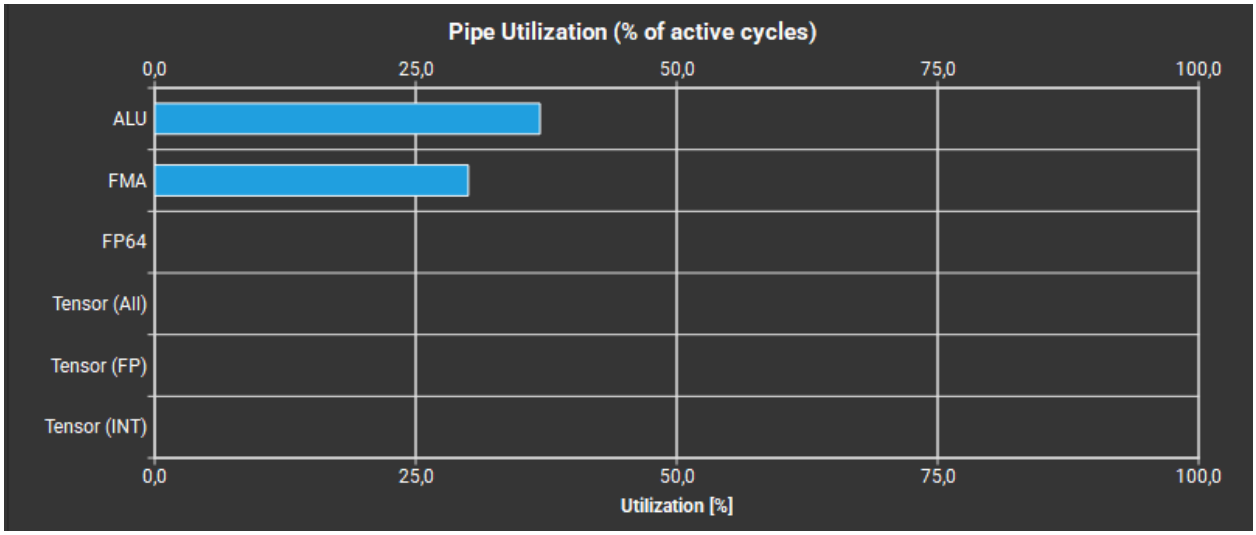

Рисунок 1. Арифметической интенсивности

Формула для арифметической интенсивности:

$$AI = \frac{\text{Compute Throughput}}{Memory\ \text{Throughput}}$$

Подставляем значения из таблицы:

$$\frac{63{,}94}{70{,}90} = 0.9$$

Арифметическая интенсивность меньше 1 → программа ограничена пропускной способностью памяти (memory-bound). Это значит, что узким местом является доступ к памяти, а не вычисления.

3. Occupancy (заполняемость) и объединение блоков

Практическая заполняемость определяется количеством регистров, размером блока и ограничениями на количество активных потоков на мультипроцессор.

Регистров используется 22 на поток → это хороший показатель, который не ограничивает сильно заполняемость.

Размер блока (16, 16, 1) = 256 потоков на блок.

Максимальное количество потоков на мультипроцессор в большинстве архитектур CUDA — 2048 потоков.

$$\frac{2048}{256} = 8 \text{ блока}$$

Значит, одновременно на одном мультипроцессоре может выполняться до 8 блоков, если не ограничивает использование регистров и общего разделяемого объема памяти.

4. Оценка с размером блока 32

Если сделать размер блока (32, 32, 1), получится **1024 потока на блок**:

$$\frac{2048}{1024} = 2 \text{ блока}$$

Ограничение по количеству потоков: при размере блока 32 (1024 потока) заполняемость снизится из-за ограничения на максимальное количество потоков на мультипроцессор.

Оптимальный размер блока обычно — **128–256 потоков** для лучшего баланса между заполняемостью и арифметической интенсивностью.

5. Соотношение между «Issue Active» и «Inst Executed»

Issue Active: 11

Inst Executed: 2709520

Inst Executed / Issue Active = 246 320

6. Соотношение между инструкциями доступа к данным (Lsu), арифметикой (Fma, Alu) и инструкциями ветвления (Adu)

Lsu: 28.56

Fma: 30.05

Adu: 7.16

Alu: 36.91

7. Память.

Dram: Cycles Active - 59485.33333333333

8. Оптимизация

Да, проект можно оптимизировать

В процессе оптимизации, скорость, была увеличена на 60%.

Вывод

Оптимизация позволила увеличить скорость выполнения на 60% благодаря следующим мерам. Применение типа данных uchar4 вместо отдельных каналов для более эффективной загрузки данных и снижения нагрузки на память. Использование оптимизированного размера блока (32, 8) для обеспечения оптимального баланса между заполнением потоков и пропускной способностью памяти. Сокращение инструкций загрузки/выгрузки памяти с помощью функции cudaMemcpy2D.

Источники

1. Документация NVIDIA CUDA. [Электронный ресурс] URL: https://docs.nvidia.com/cuda/ Дата обращения: (03.03.2025 г).

Листинг

Листинг 1. Старый код

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

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

__global__ void gammaCorrectionKernel(unsigned char* input, unsigned char* output,
int width, int height, float gamma) {
```

```
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x < width && y < height) {
        int idx = (y * width + x) * 3;
        for (int i = 0; i < 3; i++) {
            float normalized = input[idx + i] / 255.0f;
            float corrected = powf(normalized, gamma);
            output[idx + i] = (unsigned char)(corrected * 255.0f);
        }
    }
}

int main() {
    int width, height, channels;
    unsigned char* h_input = stbi_load("input.png", &width, &height, &channels, 3);
    if (!h_input) {
        fprintf(stderr, "Не удалось загрузить изображение\n");
        return 1;
    }

    size_t imageSize = width * height * 3 * sizeof(unsigned char);
    unsigned char* h_output = (unsigned char*)malloc(imageSize);

    unsigned char* d_input, * d_output;
    cudaMalloc((void**)&d_input, imageSize);
    cudaMalloc((void**)&d_output, imageSize);
    cudaMemcpy(d_input, h_input, imageSize, cudaMemcpyHostToDevice);

    dim3 blockSize(16, 16);
    dim3 gridSize((width + blockSize.x - 1) / blockSize.x, (height + blockSize.y -
1) / blockSize.y);

    float gamma = 2.2f;
    gammaCorrectionKernel << <gridSize, blockSize >> > (d_input, d_output, width,
height, gamma);
    cudaDeviceSynchronize();

    cudaMemcpy(h_output, d_output, imageSize, cudaMemcpyDeviceToHost);

    stbi_write_png("output.png", width, height, 3, h_output, width * 3);

    cudaFree(d_input);
    cudaFree(d_output);
    stbi_image_free(h_input);
    free(h_output);

    return 0;
}
```

Листинг 2. Новый код

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

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

__global__ void gammaCorrectionKernel(const uchar4* __restrict__ input, uchar4*
__restrict__ output, int width, int height, float gamma) {
      int x = blockIdx.x * blockDim.x + threadIdx.x;
```

```
	int y = blockIdx.y * blockDim.y + threadIdx.y;

	if (x < width && y < height) {
		int idx = y * width + x;
		uchar4 pixel = input[idx];

		float3 normalized = make_float3(pixel.x / 255.0f, pixel.y / 255.0f,
pixel.z / 255.0f);
		float3	corrected	=	make_float3(powf(normalized.x,	gamma),
powf(normalized.y, gamma), powf(normalized.z, gamma));

		output[idx] = make_uchar4(
			(unsigned char)(corrected.x * 255.0f),
			(unsigned char)(corrected.y * 255.0f),
			(unsigned char)(corrected.z * 255.0f),
			255
		);
	}
}

int main() {
	int width, height, channels;
	unsigned char* h_input = stbi_load("input.png", &width, &height, &channels,
3);
	if (!h_input) {
		fprintf(stderr, "Не удалось загрузить изображение\n");
		return 1;
	}

	int pitch = (width * 3 + 3) & ~3;
	size_t imageSize = pitch * height;

	unsigned char* h_padded = (unsigned char*)malloc(imageSize);
	for (int y = 0; y < height; y++) {
		memcpy(h_padded + y * pitch, h_input + y * width * 3, width * 3);
	}

	uchar4* d_input, * d_output;
	cudaMalloc((void**)&d_input, imageSize);
	cudaMalloc((void**)&d_output, imageSize);

	cudaMemcpy2D(d_input,	pitch,	h_padded,	pitch,	width	*	3,	height,
cudaMemcpyHostToDevice);

	dim3 blockSize(32, 8);
	dim3 gridSize((width + blockSize.x - 1) / blockSize.x, (height + blockSize.y
- 1) / blockSize.y);

	float gamma = 2.2f;
	gammaCorrectionKernel << <gridSize, blockSize >> > (d_input, d_output, width,
height, gamma);
	cudaDeviceSynchronize();

	cudaMemcpy2D(h_padded,	pitch,	d_output,	pitch,	width	*	3,	height,
cudaMemcpyDeviceToHost);

	for (int y = 0; y < height; y++) {
		memcpy(h_input + y * width * 3, h_padded + y * pitch, width * 3);
	}
	stbi_write_png("output.png", width, height, 3, h_input, width * 3);

	cudaFree(d_input);
	cudaFree(d_output);
	stbi_image_free(h_input);
	free(h_padded);
```

```
	return 0;
}
```