Санкт-Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Акиньшин Вадим Витальевич

Выпускная квалификационная работа бакалавра

Методы технологии CUDA обработки цифровых изображений на графических процессорах

Направление 01.03.02
Прикладная математика и информатика

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор — Андрианов С. Н.

Научный руководитель,
кандидат физ.-мат. наук,
доцент — Степенко Н. А.

Рецензент,
кандидат физ.-мат. наук,
доцент — Гришкин В. М.

Санкт-Петербург
2022

Содержание

Введение

В современном мире во многих отраслях деятельности человека используется обработка изображений и сигналов. Например, системы мониторинга и наблюдения с помощью видеокамер, техническое и компьютерное зрение, разнообразные автономные системы, которые анализируют поступающую видеоинформацию и так далее. Уровень развития компьютерных технологий растет каждый день и значимую роль, в том числе, стали играть алгоритмы и методы обработки и улучшения качества видеоинформации. Благодаря им упрощается общее восприятие графической информации человеком и становится возможным дальнейшее распознавание образов и объектов с целью их последующего использования в автономных системах.

Самой распространенной областью использования алгоритмов обработки изображений стали системы, анализа в реальном времени. Например, городские и дорожно-транспортные системы наблюдения. С нарастающим объемом поступающих данных возникла острая необходимость в оптимизации методов и улучшении их быстродействия. Самым эффективным решением поставленной задачи стало внедрение технологии GPGPU (General-purpose graphics processing units), которая стала незаменимым инструментом для осуществления параллельных вычислений. С развитием этой технологии компания NVIDIA сделала огромнейший шаг вперед и разработала архитектуру CUDA, которая как потомок сохранила в себе все лучшие особенности предка, и стала еще быстрее и удобнее. CUDA (Compute Unified Device Architecture) - архитектура и программная модификация для реализации параллельных вычислений, позволяющая проводить расчеты с помощью GPU NVIDIA со значительным увеличением скорости. Сегодня с ее помощью решают широкий спектр научно-исследовательских задач.

Обзор литературы

Книга [1] является отличным практическим руководством по реализации программного обечеспечения с использованием технологии CUDA 4 версии от компании NVIDIA. В первой части подробно изложены основы программной модели CUDA на языках C и Fortran, информация о типах памяти и методах эффективного применения разделяемой памяти на примере различных вычислительных алгоритмов. Вторая часть книги обозревает прикладные математические библиотеки и языковые надстройки на базе CUDA. Последние разделы книги посвящаются деталям профессиональной разработки, а именно средствам отладки, анализа и диагностики. Также рассматриваются способы применения нескольких графических процессоров на распределенных кластерных системах и рабочих станциях. В заключительной части книги расположено несколько статей о применениях технологии CUDA в задачах мат. моделирования, гидродинамики и компьютерной графики. Данная книга предназначена для разработчиков, а также исследователей, которые применяют параллельные вычисления.

В книге [2], написанной старшими членами команды по созданию и разработке CUDA, новая технология представлена от лица программиста. Тут рассматриваются все аспекты создания программ на CUDA с примерами. После короткого введения в архитектуру и небольшого обзора языка C идет большой блок с подробным обсуждением каждой функциональной особенности и тех компромиссов, с которые столкнется разработчик. Также там даются советы, как добиться наибольшей производительности программы.

Работа [3] посвящена программированию на GPU с помощью CUDA. В первой части подробно разбирается технология, архитекутра поддерживаемых видеочипов и способы оптимизации. Далее разбирается большой класс различных алгоритмов, реализоавнных на CUDA.

Для получения дополнительной углубленной информации и упрощения формулировок и терминов более понятным и лаконичным языком были использованы следующие интернет ресурсы: В научно популярных статьях [4]-[8] простым языком отлично описана история создания технология CUDA, способы применения всех имеющихся инструментов, а также приведены наглядные примеры. Для изучения библиотеки CUDAfy было ис-

пользовано самое подробное руководство пользователя [9]. Для получения углубленной информации о методах обработки изображения и используемого математического аппарата были изучены интернет ресурсы [10] - [12].

Глава 1. Вычисления на графических процессорах, технология CUDA

1.1. Определения и основные понятия

CUDA – это программно-аппаратная архитектура параллельных вычислений, с помощью которой можно существенно увеличить скорость вычислений используя графические процессоров NVIDIA. При использовании данной технологии необходимо знать следующие понятия:

Устройство (device) – видеокарта, графический процессор, GPU – исполняет команды центрального процессора.

Хост (host) – центральный процессор (CPU) – запускает команды, выделяет память, и тд.

Ядро (kernel) – функция (задание), которая будет выполняться на видеочипе.

API (Application Programming Interface или интерфейс программирования приложений) — это инструменты и функции в виде интерфейса для создания новых приложений.

RunTime - низкоуровневый API языка программирования C и принадлежит к его языковой библиотеке.

"Архитектура CUDA позволяет разработчикам на свое усмотрение организовывать процесс доступа к набору различных инструкций GPU и управлять памятью."

Технология поддерживает некоторые основные языки программирования, такие как Java, Python, C++ и другие. Рассмотрим запуск программы на графическом процессоре:

1. Хост выделяет требуемое количество памяти на устройстве.
2. Хост копирует данные из своей памяти в память устройства.
3. Хост запускает задание(ядро) на устройстве.
4. Устройство исполняет это задание(ядро).
5. Хост копирует результаты из памяти устройства в свою память.

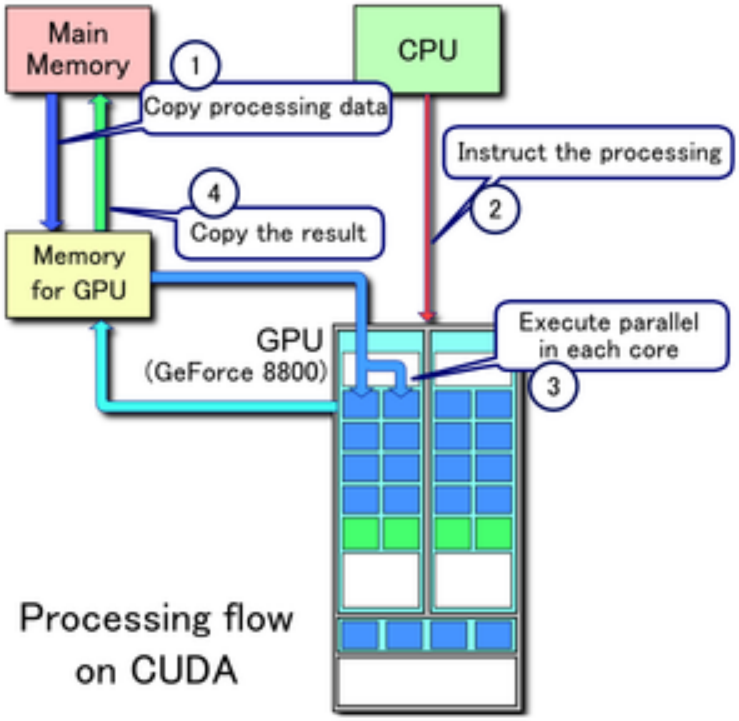

Рис. 1: Шаги запуска программы, кроме первого

Центральный процессор взаимодействует с графическим через CUDA Runtime API, CUDA Driver API и CUDA Libraries. Runtime и Driver API отличаются уровнем абстракции. Runtime API упрощает управление кодом устройства, предоставляя неявную инициализацию, управление контекстом и управление модулями. Это приводит к упрощению кода. Driver API имеет более сложное строение кода, зато предлагает более детальный контроль, особенно над контекстами и загрузкой модулей. Однако важно помнить, что если время, затраченное на работу ядер, будет меньше времени, затраченного на запуск этих ядер, а также на выделение памяти, то в результате мы получим нулевую эффективность использования GPU. Именно поэтому технология параллельных вычислений не используется для решения не сложных и быстрых задач, с которыми отлично справляется CPU.

1.2. Архитектура CUDA, библиотека CUDAfy.NET

Библиотека Cudafy.NET используется для программирования на видеокартах и представляет с собой набор библиотек и инструментальных

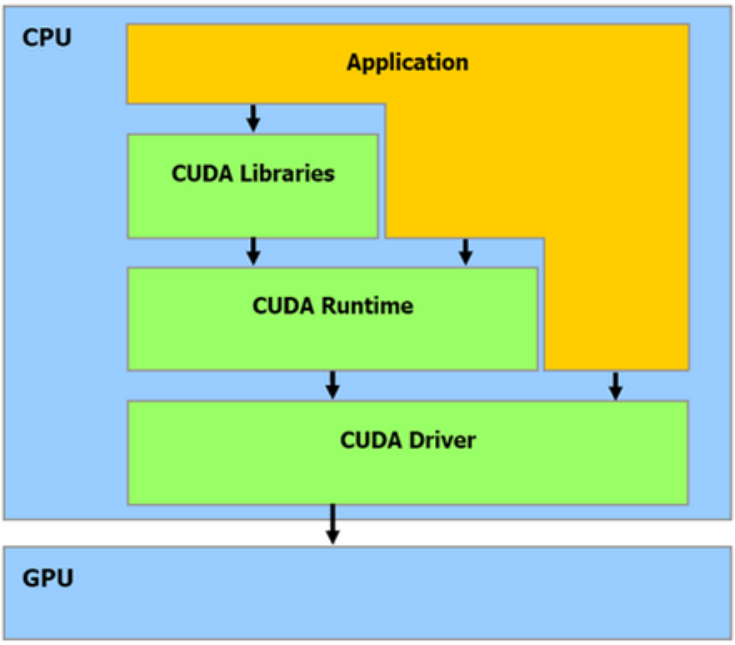

Рис. 2: Взаимодействие CPU и GPU

средств, с помощью которых можно разрабатывать и исполнять программы на устройствах, поддерживающих платформу NVIDIA CUDA или OpenCL.

Архитектура графического процессора имеет ряд существенных отличий от архитектуры центрального процессора. На раннем этапе развития технологии видеочипов применялись исключительно для графических вычислений, допускающих параллельную обработку данных. Графический процессор нужен для запуска большого числа элементарных параллельных процессов (потоков). Распараллеливание вычислений достигается использованием принципов SIMD и многопроцессорной технологии.

SIMD (single instruction, multiple data, то есть одиночный поток команд, множественный поток данных) — принцип компьютерных вычислений. Обеспечивает параллелизм на уровне данных. Проще говоря, одна инструкция одновременно обрабатывает множество данных.

Мультипроцессор – многоядерный SIMD процессор, который в каждый момент времени выполняет на всех ядрах только одну инструкцию.

Для наглядности графически представим архитектуру CPU и GPU:

На изображении видно, что в GPU есть много простых небольших арифметически-логических устройств (АЛП), объединенных в несколько групп и обладающих общей памятью. Благодаря этому помогает повышается продуктивность в вычислительных заданиях, но усложняется процесс

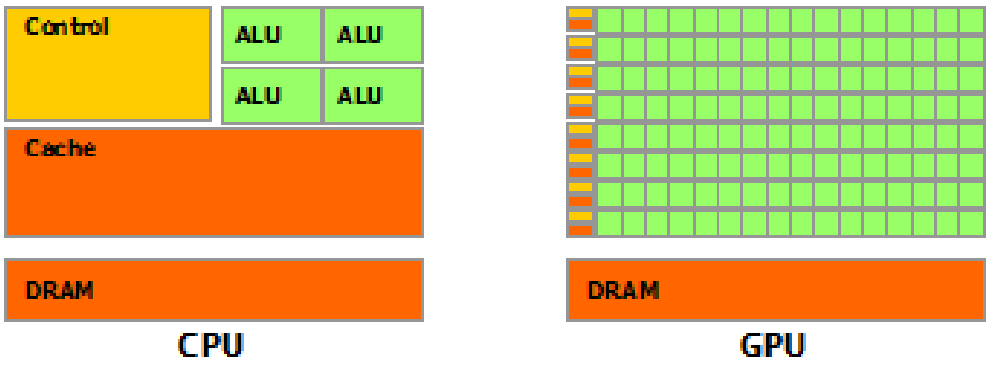

Рис. 3: Схематичное сравнение CPU и GPU

программирования. «Для достижения лучшего ускорения необходимо продумывать стратегии доступа к памяти и учитывать аппаратные особенности.»

GPU ориентирован на выполнение программ с большим объемом данных и расчетов. Он представляет собой массив потоковых процессоров (Streaming Processor Array).Каждый потоковый процессор состоит из кластеров текстурных процессоров (Texture Processor Clusters, TPC). TPC в свою очередь состоит из набора мультипроцессоров (SM – Streaming Multiprocessor), каждый из которых содержит несколько потоковых процессоров (SP – Streaming Processors) или ядер. В современных процессорах количество ядер превышает 1024. В каждом мультипроцессоре набор ядер работает по принципу SIMD, однако имеются некоторые различия, о которых будет сказано далее.

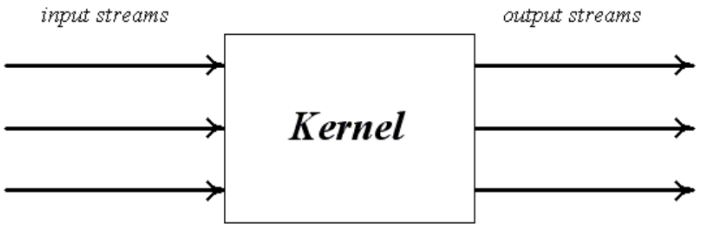

Рис. 4: Устройство GPU

В итоге GPU стал устройством, которое реализует потоковую вычислительную модель (stream computing model): есть несколько потоков входящих и исходящих данных, которые состоят из одинаковых элементов. Каждый элемент может быть обработан независимо от остальных.

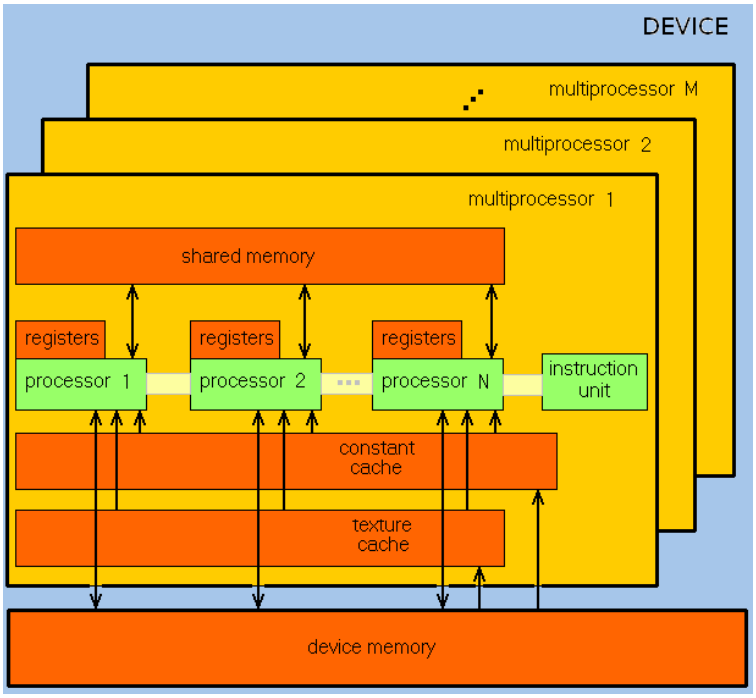

Рис. 5: Архитектура GPU

Технология CUDA использует огромное количество раздельных потоков для произведения расчетов. Все они объединяются в конкретную иерархию – grid / block / thread (сетка, блок, нить).

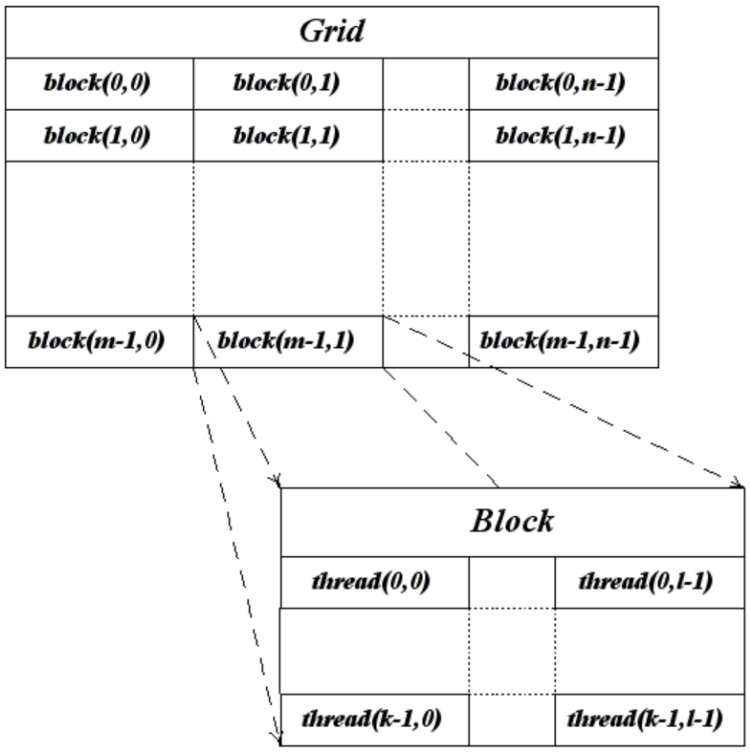

Рис. 6: Структура блоков

Первый уровень - сетка(grid) - связан непосредственно с ядром. Сетка объединяет все потоки, выполняемые ядром, представляя собой массив блоков (block) одномерный или двумерный. Каждый из блоков это обособленный набор когерентных потоков (нитей - threads). Поэтому нити из разных блоковво время выполнения программы никак не связаны между собой. В сетке у всех блоков имеется конкретный адрес - одно, два неотрицательных целых числа (индекс блока в сетке). Аналогично, каждая нить в блоке также имеет свой собственный уникальный адрес. Только в этом случае он представляет собой от одного до трех неотрицательных целых чисел.

Теперь давайте перейдем к различию между архитектурой GPU и чистой концепцией SIMD. Определим термин warp (будем далее упоминать как варп). В варпе 32 потока (бывают исключения). Только потоки из одного варпа могут по-настоящему выполняться одновременно. Потоки разных варпов осуществляются последовательно на различных этапах программы. Этот принцип компьютерых вычислений называется SIMT (Single Instruction - Multiple Theads). Контроль за варпами происходит на уровне программной архитектуры устройства или компьютера.

1.3. Управление пямятью в CUDA

Между собой потоки взаимодействуют с помощью использования различных видов памяти. В CUDA выделяют 6 основных видов:

- Регистры;
- Локальная;
- Глобальная;
- Разделяемая;
- Константная;
- Текстурная.

Подобное разнообразие обусловлено в первую очередь особенностью работы видеокарты и ее изначальным предназначением. Рассмотрим подробнее все виды памяти.

Регистры(Registers). Все доступные регистры совместно используются потоками блока во время компиляции. Каждый из потоков в полной мере использует несколько регистров, доступных для чтения и записи во время работы этого ядра. В этом случае поток не имеет доступа к регистрам других потоков. Регистры расположены в потоковом мультипроцессоре и поэтому имеют самую высокую скорость доступа.

Локальная память (Local Memory) - расположена в DRAM (Dynamic Random Access Memory - Динамическая память с произвольным доступом) и используется для поиска данных локального потока в случае, если существующих регистров недостаточно.

Глобальная память(Global Memory) - это обычная память DRAM, выделяемая с помощью специальных функций процессора. Основным отличием глобальной памяти является произвольная адресация, то есть возможность чтения из любой ячейки и записи в произвольную ячейку. Но следует отметить, что общая память значительно ниже по скорости и не может быть кэширована.

Разделяемая память(Shared Memory) расположена в потоковом мультипроцессоре, но выделяется на уровне блоков: каждый из блоков получает одинаковый объем общей памяти для своего использования. Все потоки в блоке имеют доступ на чтение и запись к этой памяти.

Постоянная память и текстурная память (Constant and Texture Memory) также находятся в динамической памяти. Однако они не кэшируются и поэтому имеют высокую скорость доступа. Обратите внимание, что эти типы памяти немедленно доступны всем потокам в сетке, но только для чтения. Операция записи на них может быть выполнена только процессором с помощью специальных функций.

Ниже более наглядно представлена архитектура CUDA с точки зрения взаимодействия нитей и блоков с памятью внутри сетки:

1.4. Практическое использование CUDA

Для программиста люывателя, графический конвейер это несколько стадий обработки. Сначала геометрический блок создает треугольники, затем растеризующий блок отрисовывает пиксели на мониторе.

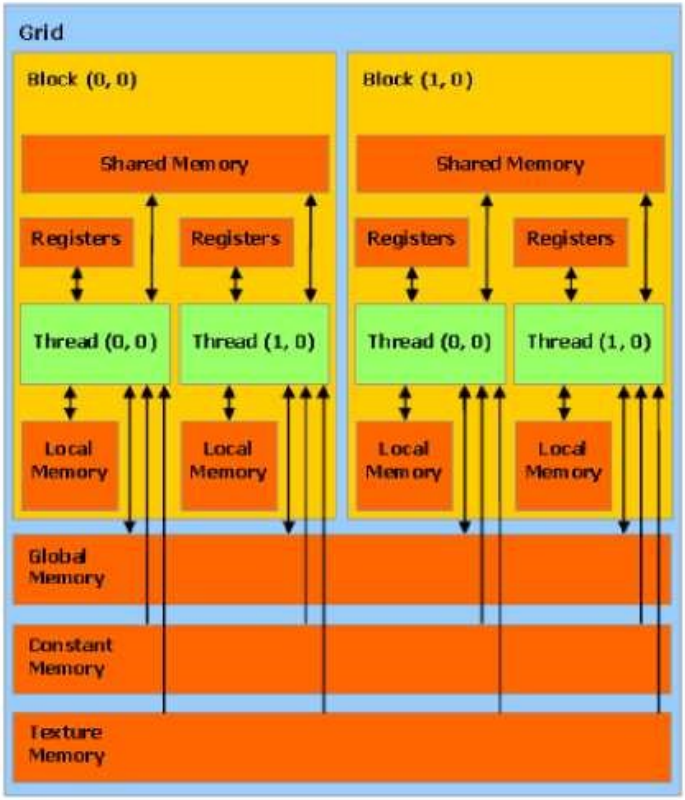

Рис. 7: Архитектура технологии CUDA

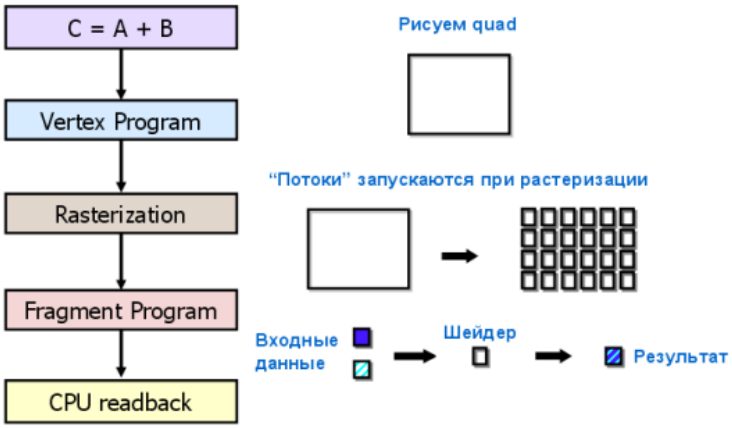

Рис. 8: Стандартная модель программирования GPGPU

Главным преимуществом архитектуры CUDA от компании NVIDIA по сравнению с предшествующими ей версиями GPGPU является полная функциональность для написания с ее помощью программ на языке Си с соответствующим этому языку синтаксисом, указателями и минимальным требованием к каким либо дополнительным расширениям для обеспечения доступа к ресурсам GPU. CUDA смогла обособиться от графических

API и имеет ряд ключевых особенностей, которые позволяют эффективнее проводить вычисления общего назначения на графических процессорах.

Теперь более подробно разберем основные преимущества и недостатки технологии CUDA. Преимущества CUDA:

1. В качестве основы для создания интерфейса программирования приложений CUDA был выбран популярный язык программирования Си. Поэтому весь процесс обучения и интегрирования технологии в программу очень сильно упростился;

2. Доступность разделяемой памяти в 16 килобайт на мультипроцессор. Ее используют для работы кэша с более широкой пропускной способностью;

3. Улучшение эффективности и скорости обмена между памятью графического процессора и памятью системы;

4. Полный отказ от использования графических API;

5. Процесс записи возможен на произвольные адреса;

6. Поддержка операций с битовыми цепочками и целочисленными значениями.

Недостатки CUDA:

1. Не поддерживается рекурсия для исполняемых функций;

2. 32 потока - минимальная ширина блока;

3. Архитектура закрытая и принадлежит Nvidia.

Благодаря всем вышеперечисленным преимуществам, технология CUDA стала передовой в своей сфере, внесла неотъемлемый вклад в развитие и внедрение высокопроизводительных параллельных вычислений. С помощью нее можно отлично оптимизировать фильтры обработки изображений, а также эффективно решать широкий спектр задач из разных областей.

Глава 2. Методы цифровой обработки изображений и их реализация

2.1. Степенные преобразования

Разработано и придумано много методов, повышающих качество изображения. Выделяют две основные группы: методы обработки в пространственной и частотной областях. Подходы первой группы основаны на прямом управлении пикселями. Подходы второй группы основаны на модификации сигнала, полученого применением преобразования Фурье к изображению.

Градационные преобразования изображений (степенные) - подгруппа пространственных методов обработки. Они оперируют величинами пикселей и задаются уравнением:

$$s(x, y) = T[r(x, y)]$$

где r и s — значения яркостей входящего и исходящего изображений в каждой точке (x, y); T — оператор преобразования над начальным изображением, который задается в окрестности точки (x, y). s(x, y) зависит исключительно от величины яркости r(x, y) изначального изображения, а в качестве оператора T принимается функция градационного преобразования яркостей (функция преобразования интенсивностей).

Обозначая зависимость по формуле, можно построить несколько простых и эффективных алгоритмов пространственной обработки изображений. Так как улучшение каждого компонента изображения обуславливается только яркостью этого компонента, то такие алгоритмы нередко называют процедурами поэлементной обработки. На (рис. 11) показаны три главных типа преобразований для усовершенствования изображений: линейное (тождественное преобразование и негатив), логарифмическое (обычный и обратный логарифм) и степенное (n-я степень и корень n-й степени). Тождественное преобразование это примитивный случай, при котором яркость на входе и выходе одинаковы. Оно приведено на графике для полноты рас-

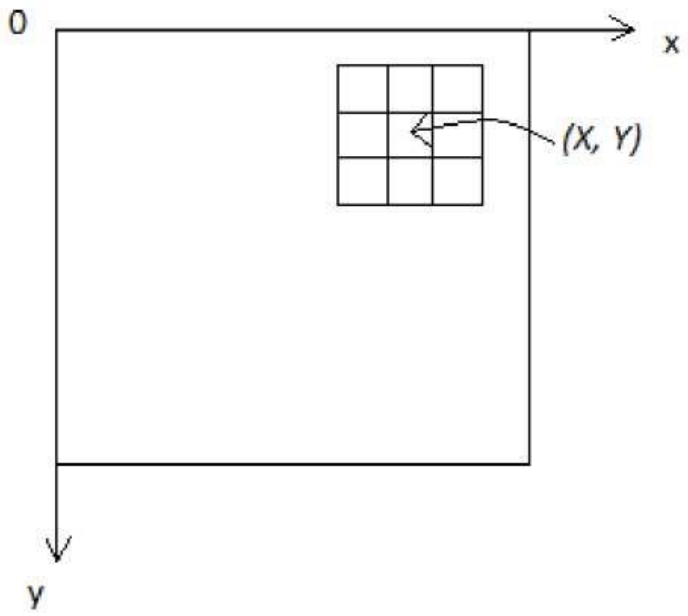

Рис. 9: Окрестность 3 x 3 вокруг точки (x, y) изображения

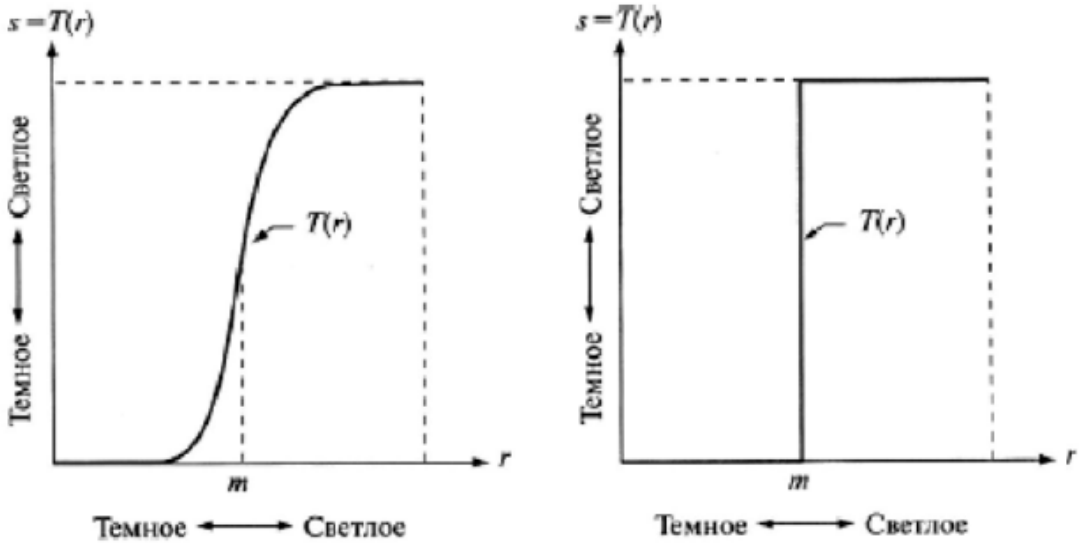

Рис. 10: Градационное улучшения контраста

смотрения.

Рассмотрим другие виды степенных преобразований.

Негативное преобразование

Благодаря негативному преобразованию можно перевести изображение в негатив. В таком случае значения яркости пикселей будут в диапазоне от 0 до L-1. Это преобразование задается формулой:

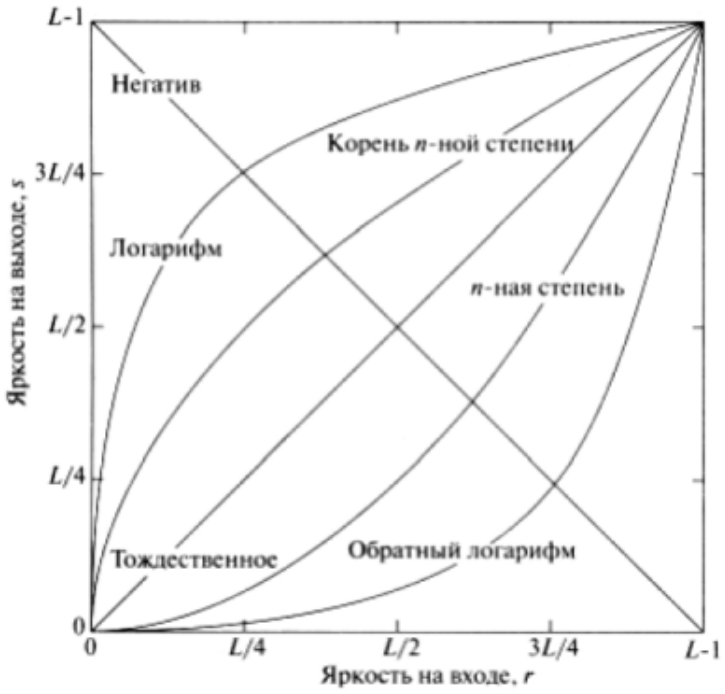

Рис. 11: Функции градационных преобразований

$$s(x, y) = (L - 1) - r(x, y)$$

Такое инвертирование соотвествует реальному фотографическому эффекту негатива. С его помощью усиляют светлые детали на преобладающих темных областях фотографии.

Прямое и обратное логарифмическое преобразование

Общий вид логарифмического преобразования:

$$s(x, y) = clg(1 + r(x, y))$$

где c — константа и r > 0.

Форма кривой логарифмической свидетельствует о том, что небольшой диапазон низких величин яркости на входном изображении превращается в более широкий выходных величин, а широкий диапазон высоких значений яркости превращается в более узкий небольших значений. При использовании обратного логарифмического преобразования диапазон темных пикселей сжимается, а диапазон светлых пикселей растягивается. Любая кривая, похожая на показанную на (рис. 12), вызовет такое увеличение

или уменьшение разброса яркостных величин на изображении. Использование прямого преобразования более целесообразно в тех случаях, когда большая часть деталей изображения сосредоточена в темных областях. Использование обратного логарифмического преобразования более целесообразно в тех случаях, когда большая часть деталей изображения сосредоточена в светлых областях. Несмотря на то, что степенные преобразования более универсальны, чем логарифмические преобразования, последние обладают чрезвычайно важной особенностью, позволяющей сжимать динамический диапазон пикселей в изображениях с широким диапазоном значений яркости.

Степенные преобразования

Степенные преобразования задаются формулой:

$$s(x, y) = cr(x, y)^{\gamma}$$

где c и γ — положительные константы.

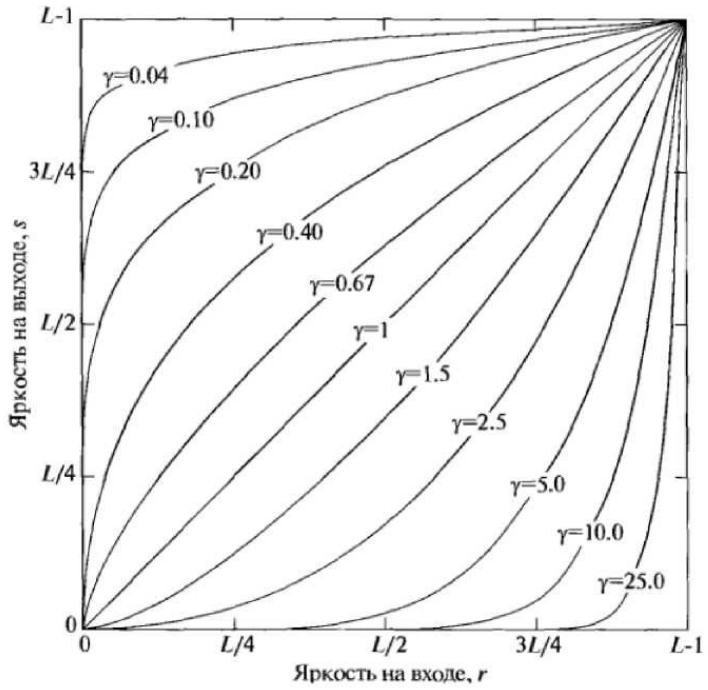

Рис. 12: Графики степенных преобразований

При небольших значениях переменной гамма степенные преобразования создают тот же эффект, что и логарифмические. Однако за счет изменения параметра γ тут получается большое семейство кривых. Кри-

вые с параметром $\gamma > 1$ и $\gamma < 1$ дают противоположные эффекты. При $\gamma = \ = 1$ преобразование становится тождественным. Данная процедура поправки степенного параметра называется гамма-коррекцией.

Гамма-коррекция используется в качестве алгоритма обработки, когда требуется максимальная точность при отображении изображения на мониторе. Неверно подправленное изображение может выглядеть блеклым или чересчур темным. Однако верное воспроизведение цветов требует точных знаний о гамма-коррекции, поскольку в результате меняется не только яркость, но и пропорции между красным, зеленым и синим.

2.2. Использование медианного фильтра для восстановления изображения

Помимо полезного сигнала, цифровое изображение часто содержит различные шумы. Поэтому, работая с изображениями, часто возникает неоюходимость в их устранении. Существует множество различных типов шумов. Мы сосредоточимся на импульсном шуме — искажение сигнала большими всплесками коротких импульсов случайных значений. Импульсный шум в сигнале возникает из-за ошибок в работы пикселя цифровой матрицы, повреждения фрагмента памяти (при отсутствии контроля ошибок) или ошибок в каналах обмена сообщениями. Пусть процесс искажения изображения (рис. 13) заключается в действии оператора искажения H на начальное изображение $r(x, y)$. Далее, после создания аддитивного шума $\mu(x, y)$ получается искаженное изображение $s(x, y)$. Наша задача восстановить изображение. Для этого построим аппроксимацию $\tilde{r}(x, y)$ из имеющегося искаженного изображения $s(x, y)$, информации об искажающей функции H и аддитивном шуме $\mu(x, y)$. Будем считать, что оператор H является тождественным оператором и что искажение изображения вызвано только наличием шума соль-перец. Перечно-солевой шум характеризуется тем, что поврежденные пиксели принимают только максимальные или минимальные значения в динамическом диапазоне 1.

На сегодняшний день выработано много разных алгоритмов, которые способны удалять шумы. Рассмотрим один из самых популярных пространственных фильтров — медианный фильтр.

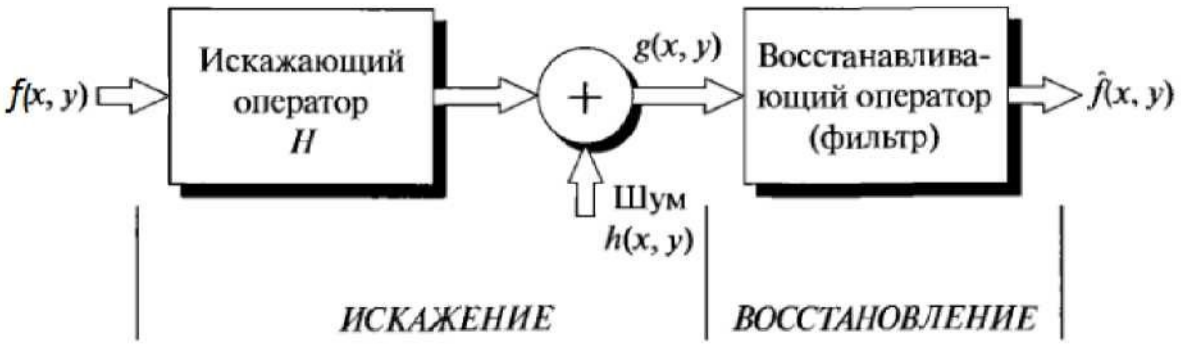

Рис. 13: Искажение и восстановление изображений

Медианная фильтрация

Алгоритм медианной фильтрации был разработан в 1974 году Джоном Уаилдером Тьюк. Он относится к подгруппе оконных фильтров, которые постепенно перемщаются по массиву исходного цифрового сигнала и перезаписыают на каждом шаге один элемент из окна. При фильтрации выходящий сигнал y_k ширины n для текущего пикселя формируется из полученного ряда значений $..., x_{k-1}, x_k, x_{k+1}, ...$ по формуле:

$$y_k = med(x_{k-(n-1)/2}, ..., x_k, ...x_{k+(n-1)/2})$$

где $med(x_1, ..., x_n) = x_{(n+1)/2}$ — элементыпо возрастанию их значений $x_1 = min(x_1, ..., x_n) \leq x_2 \leq x_3 \leq ... \leq x_n = max(x_1, ..., x_n)$. Ширина для медианной фильтрации подбирается для того, чтобы она смогла подавить импульс ширины $(n-1)/2$ при условии, что n нечетное.

В конечном итоге, медианная фильтрация осуществляется в виде функции локальной обработки значений пикселей в скользящем окне $S_{i,j}$, которое содержит в себе конкретное число значений сигнала. Для каждого из положений окна, расположенные в нем элементы сортируются по возрастанию или убыванию. Средний элемент в подобном упорядоченном списке называется медианой. И именно этим элементом заменяется центральный в окне для обрабатываемого изображения. Если количество элементов четное, то медиана вычисляется как среднее арифметическое двух центральных элементов упорядоченного списка. На каждом этапе фильтрации используется текущее значение сигнала.

2.3. Практическая реализация

Теперь с помощью технологии CUDA реализуем два метода обработки изображений: Медианную фильтрацию и гамма-коррекцию. Сперва определим общий принцип реализации задачи, а также используемую память. При медианной фильтрации набор значений, которые попадают в окно $S_{i,j}$, и поиск медианы для каждого пикселя происходят независимо. Поэтому алгоритм можно производить одновременно для всех точек и каждая нить редактирует свой пиксель входного изображения и некоторую его окрестность. Для гамма – коррекции на GPU мы использовали глобальную,

разделяемую и константую память.

Для рассчетов была написана компьютерная программа,состоящая из 2-ух модулей: графического интерфейса и вычислительный блока. Код для модулей написан на языке C# с помощью инструмента .Net Framework.

Графический интерфейс

Запускаем приложение. В начале мы видим главное окно (рис. 14). На нем располагаются несколько вкладок меню. Пункт «Файл» - «Открыть» позволяет загрузить исходное изображение для дальнейшей обработки.

Теперь с помощью пункта «Изменить» можно выбрать один из двух алгоритмов обработки. После завершения работы программы на экране мы увидим скорректированное изображение.

Результаты

Вычисления проводились на восьмиядерном процессоре CPU – Intel® Core i7 11800H (2.30 ГГц) и видеокарте NVIDIA GeForce RTX 3060 Ti. На практических задачах рассмотрим быстродействие обоих алгоритмов на CPU и GPU с помощью CUDA.

Гамма – коррекция. Рассмотрим, как с помощью гамма – коррекции управляют контрастом изображениий. На рисунке 15 наше исходное изображение — панорама леса вокруг трассы. Поскольку общая гамма в основном темная, то желательно провести увеличения уровня яркости. Для этого используем степенное преобразование с показателем степени $\gamma < 1$. Значение γ для рисунка 16, 17 и 18 равны 0.7, 0.5 и 0.2 соответственно.

Рис. 14: Исходное изображение (темный лес и дорога)

Нетрудно заметить, что при изменении значения переменной гамма с 0.7 до 0.5 на изображении проявляется все больше различных деталей.

Рис. 15: Гамма-коррекция $\gamma = 0.7$

Рис. 16: Гамма-коррекция $\gamma = 0.5$

Рис. 17: Гамма-коррекция $\gamma = 0.2$

Последущее уменьшение гаммы до 0.2 усиливает отдельные детали фона, однако возникают артефакты и ухудшается общее качество изображение.

Медианный фильтр. Для проверки правильности работы параллельного алгоритма и программы, которая реализует медианную фильтрацию, был предварительно сгенерирован набор тестовых картинок с наложенным шумом «соль-перец». На рис. 19 - 24 представлены схемы ориентации гироскопов на борту самолета с различным уровнем зашумления: слева (рис. 19) – слабая степень зашумления, (рис. 21) – средняя, (рис. 23) – сильная, а справа (рис. 20, рис. 22, рис. 24) – обработанные фильтром изображения

с окном 3 × 3.

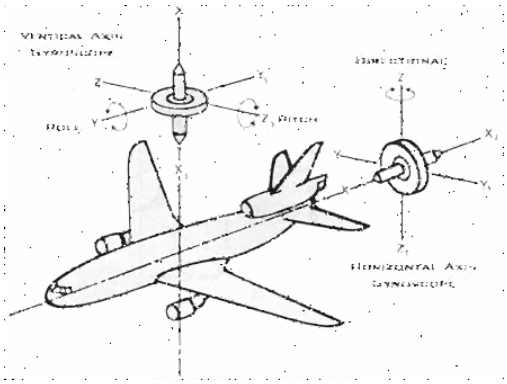

Рис. 18: Слабый шум

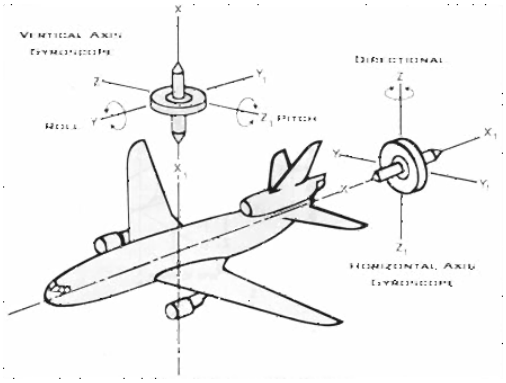

Рис. 19: Результат медианной фильтрации слабого шума

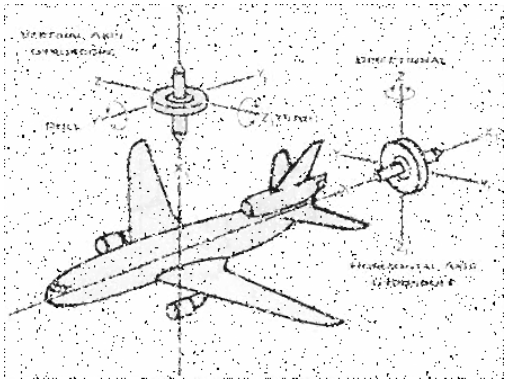

Рис. 20: Средний шум

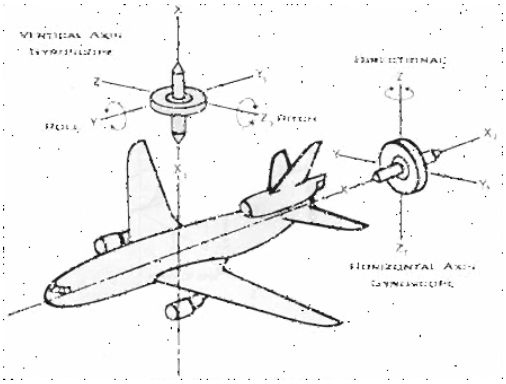

Рис. 21: Результат медианной фильтрации среднего шума

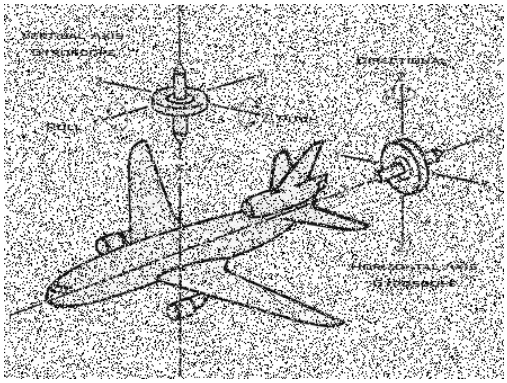

Рис. 22: Сильный шум

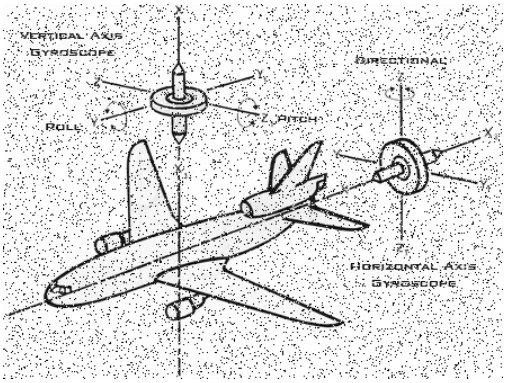

Рис. 23: Результат медианной фильтрации сильного шума

На изображениях видно, что медианная фильтрация неплохо исправляет слабую и среднюю степень шума (рис. 20 и рис. 22), однако при по-

Название операции	Размер изображения в пикселях	Время на CPU, мс	Время на GPU, мс		
			4x4	8x8	16x16
Медианная фильтрация	444x333	24 814	175	79	73
Гамма - коррекция	640x360	17 935	77	70	53

следующем усилении уровня шума фильтр с окном 3×3 не справляется и на выходе возникают ошибки.

Сравнение скорости выполнения алгоритмов на CPU и GPU

Дополнительно увеличить скорость вычислений позволяют манипуляции с размерами блоков. Все полученные данные, а именно скорость выполнения программы на CPU и GPU, а также сравнительные значения времени в зависимости от размера блока занесем в таблицу для большей наглядности.

Проанализировав результаты, данные в таблице, можно сделать вывод, что с увеличением размера блоков время, затрачиваемое на обработку изображения. При этом скорость выполнения алгоритмов на графических процессорах с помощью CUDA значительно выше, по сравнению с запуском алгоритма на центральном процессоре. А время выполнения различается более чем в 140 раз.

Выводы

Программно-аппаратная архитектура CUDA, от компаниии Nvidia для программирования на графических процессорах CUDA прекрасно подходит для решения обширного круга задач с высоким рангом параллелизма. CUDA облегчает программирование на GPU и дает пользователю множество новых возможностей: общая память, синхронизация потоков, вычисления двойной точности и операции с целыми числами.

Технологией CUDA может пользоваться каждый разработчик ПО без исключения. Для ее использование необходимо только знание языка C. Очень вероятно, что из-за большого распространения видеокарт в мире, совершенствование области параллельных вычислений на графических процессорах окажет мощное воздействие на всю отрасль высокопроизводительных вычислений. Данные возможности уже спровоцировали огромнейший интерес в научно-технических кругах и не только. Будущее огромного множества различных вычислений однозначно за параллелизацией алгоритмами.

Разумеется GPU не смогут заменить CPU. В сегодняшнем виде они предназначены для других целей. Но сейчас и видеочипы постепенно двигаются в сторону центральных процессоров, становясь всё более и более многофункциональными (осуществляют расчёты с плавающей точкой, а также целочисленные вычисления), и CPU обзаводятся большим числом ядер, многопоточными технологиями и всё более «параллелизуются». Скорее всего, в ближайшем будущем GPU и CPU сольются в один сверх универсальный процессор, а такие компании как Intel и AMD уже активно работают в этом направлении.

Заключение

Был разработан программный комплекс для коррекции изображений на GPU с параллельной архитектурой. В ходе выполнения работы были изучены некоторые методы улучшения качества цифровых изображений, осуществлена реализация на центральном и графическом процессоре с использованием технологии CUDA и библиотеки CUDAfy. Также произведено их сравнение по следующим параметрам: скорость процедуры обработки и ее зависимость от размера блоков памяти, используемых архитектурой CUDAfy.

На основе полученных результатов были сделаны следующие выводы:

1. Благодаря адаптации алгоритмов для их выполнения на графических процессорах мы можем получить существенное ускорение времени рассчетов. Поэтому параллелизация вычислений с помощью CUDA полностью оправдана.

2. Использование видеокарты (с поддержкой архитектуры CUDA) для проведения вычислений — более выгодное решение экономически и вычислительно по сравнению с использованием центрального процессора.

Список используемых источников

[1] Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учебное пособие. А. В. Боресков и др. Предисл.: В. А. Садовничий Издательство Московского университета, 2012

[2] Технология CUDA в примерах. Введение в программирование графических процессоров Джейсон Сандерс, Эдвард Кэндрот ДМК Пресс, 2011 г.

[3] Боресков А. В., Харламов А. А., Основы работы с технологией CUDA, Москва: ДМК Пресс, 2010.

[4] Краткий курс обработки изображений. -URL: https://hub.exponenta.ru/post/kratkiy-kurs-teorii-obrabotki-izobrazheniy734 (Дата обращения 09.04.2022)

[5] Image filters in CUDA. -URL: https://github.com/jonaylor89/Median-Filter-CUDA#image-filters-in-cuda (Дата обращения 15.03.2022)

[6] Знакомство с программно-аппаратной архитектурой CUDA. -URL: https://proglib.io/p/cuda (Дата обращения: 12.02.2022)

[7] Неграфические вычисления на графических процессорах. -URL: https://www.ixbt.com/video3/cuda-1.shtml (Дата обращения: 21.03.2022)

[8] CUDA: Как работает GPU. -URL: https://habr.com/ru/post/54707/ (Дата обращения: 27.01.2022)

[9] CUDAfy.NET User Guide. -URL: https://manualzz.com/doc/6666114/cudafy.net-user-guide (Дата обращения: 29.03.2022)

[10] Градационные преобразования изображений. -URL: https://bstudy.net/857716/tehnika/gradatsionnye_preobrazovaniya_izobrazheni (Дата обращения: 06.03.2022)

[11] Исследование основных градационных преобразований улучшения изображения. -URL: https://kaf24.mephi.ru/upload/iblock/d3b/d3b6183e35f6a5ec1da6b0f2151126b2.p (Дата обращения: 05.04.2022)

[12] Быстрее быстрого или глубокая оптимизация Медианной фильтрации для GPU Nvidia. -URL: https://habr.com/ru/post/308214/ (Дата обращения: 17.04.2022)