УДК 004.424.5

ОПТИМИЗАЦИЯ ОБРАБОТКИ BIG DATA С ПОМОЩЬЮ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ

С.А. Байчик
Студент 4-ого курса БГУИР ФКП, по специальности Проектирование и производство программно-управляемых электронных средств
s.bajchik@bsuir.by

С.Н. Нестеренков
Кандидат технических наук, доцент, декан факультета компьютерных систем и сетей БГУИР, доцент кафедры ПОИТ
s.nesterenkov@bsuir.by

И.С. Тарасюк
Инженер-программист ОИТ, ассистент кафедры ЭВМ
i.tarasiuk@bsuir.by

С.А. Байчик
Студент 4-ого курса БГУИР ФКП, по специальности Проектирование и производство программно-управляемых электронных средств.

С.Н. Нестеренков
Кандидат технических наук, доцент, декан факультета компьютерных систем и сетей Белорусского государственного университета информатики и радиоэлектроники, доцент кафедры программного обеспечения информационных технологий. Автор публикаций на тему машинного обучения, алгоритмов принятия решений, искусственных нейронных сетей и автоматизации.

И.С. Тарасюк
Окончил БГУИР в 2021 году по специальности «Вычислительные машины, системы и сети», магистрант первого года обучения по специальности «Системы и сети инфокоммуникаций» БГУИР.

Аннотация. Данная статья посвящена оптимизации обработки больших объемов данных с использованием графических процессоров (GPU). В статье будет представлено общее представление о технологии GPU, ее особенностях и преимуществах при обработке больших данных. Основным объектом исследования статьи является анализ производительности и оптимизации обработки больших данных на GPU с использованием различных методов и техник, таких как параллельное программирование, оптимизация ядер и алгоритмов, использование специальных библиотек. Целью данной работы является представление и анализ различных подходов к оптимизации обработки больших данных на GPU, а также оценка эффективности этих подходов и их применимости в реальных условиях.

Ключевые слова: Big Data, графический процессор, производительность.

Введение.

Уже несколько лет дата-центры многих компаний работают с вычислениями с ускорением на GPU (Graphics Processing Unit). Такой тип вычислений становится все более востребованным. Так, вычисления с ускорением на GPU можно (и нужно) использовать для ускорения требовательных к ресурсам приложений, созданных для работы в таких сферах, как глубокое обучение, аналитика и проектирование. Этот метод используют в дата-центрах крупных компаний, в лабораториях научно-исследовательских организаций, на предприятиях. Благодаря ускорению на GPU работают многие сервисы, обеспечивающие работу нейронных сетей или обрабатывающие данные, поступающие с умных автомобилей. Плюсом такого метода является то, что ресурсоемкая часть приложения, которая требует высокой вычислительной мощности, обрабатывается на GPU, а все остальное выполняется на центральном процессоре (CPU). В последние несколько лет стали появляться комбинированные решения, на основе которых создаются высокоскоростные базы данных. Такие решения стоит использовать, например, для визуализации крупных массивов данных.

Архитектура и особенности работы с GPU.

Современные графические процессоры используют конвейерную архитектуру (graphic pipeline) для выполнения операций с максимальной эффективностью компьютерной графики. Эта структура позволяет достичь высокой производительности при работе с большими объемами данных благодаря потоковой параллельности.

Входные данные для GPU представляются набором вершин, которые поступают из приложения. Затем вершины обрабатываются в вершинных процессорах, которые классифицируются как MIMD. Для работы с вершинными процессорами используется программа, называемая вершинным шейдером. После обработки результат работы вершинного шейдера поступает на сборку примитивов, таких как полигоны или линии. Затем выполняются стандартные операции компьютерной графики, такие как тесты видимости отсечения. После этого данные поступают на растеризацию, где объемное изображение проецируется на плоский экран и масштабируется согласно параметрам окна приложения. Результатом этой операции является текстура, представляющая из себя прямоугольный массив пикселей. Пиксель может быть представлен 1-4 числами, например, в формате RGBA, т.е. red, green, blue, alpha (коэффициент прозрачности).

Текстура обрабатывается в процессорах пикселей, которые, согласно классификации многопроцессорных систем, могут быть отнесены к категории SIMD. После выполнения дополнительных операций данные передаются в буфер кадра, который и является изображением, видимым на экране для потребителя.

Архитектура GPU отличается от архитектуры CPU тем, что она специализирована на выполнении параллельных вычислений. GPU состоит из большого числа ядер (обычно от нескольких сотен до нескольких тысяч), которые могут работать независимо друг от друга. Это позволяет GPU эффективно обрабатывать множество задач одновременно.

GPU, аналогично CPU, обладает своими регистрами и кэшами для быстрого доступа к данным в процессе вычислений. В дополнение к этому у GPU есть своя основная память - графическая память. Для того, чтобы программист мог выполнять вычисления на GPU, ему необходимо предварительно передать данные из оперативной памяти CPU в память GPU. Эта операция традиционно считается дорогостоящей в плане производительности из-за относительно низкой скорости передачи данных между памятью приложения и графической памятью, однако современные шины PCI Express и специальные чипы на материнской плате (например, NV3 и NV4) заметно ускорили этот процесс.

В отличие от памяти CPU, у памяти GPU есть некоторые ограничения, и доступ к ней возможен только через некоторые абстракции графического программного интерфейса. Каждая из этих абстракций может рассматриваться как поток со своим собственным набором правил доступа. Программисту доступны четыре таких потока: поток вершин, поток фрагментов, поток буфера кадров и поток текстур.

Одна из главных особенностей работы с GPU — это необходимость оптимизации кода под параллельную обработку. Это достигается использованием таких технологий, как CUDA (Compute Unified Device Architecture) от NVIDIA, OpenCL (Open Computing Language) и других. Технология CUDA представляет собой платформу для параллельных вычислений, которая позволяет программистам использовать графические ускорители NVIDIA в целях обработки больших объемов данных с высокой скоростью. Она предоставляет набор инструментов и библиотек для разработки приложений, использующих параллельные вычисления на графических процессорах. Кроме того, для эффективной работы с GPU требуется умение распараллеливать задачи и разбивать их на подзадачи, которые могут выполняться параллельно [1, 2].

При работе с GPU есть ряд преимуществ и ограничений, которые следует учитывать.

Преимущества работы с GPU:

1. Высокая скорость обработки данных. GPU может обрабатывать большое количество данных параллельно, что позволяет сократить время выполнения задачи в несколько раз по сравнению с использованием только центрального процессора.

2. Высокая энергоэффективность. GPU потребляет меньше энергии, чем CPU при выполнении тех же задач.

3. Поддержка технологий глубокого обучения. GPU используется для обучения нейронных сетей, так как он может параллельно обрабатывать множество операций с большими объемами данных.

4. Наличие специализированных библиотек и фреймворков для работы с GPU, таких как CUDA и OpenCL.

Ограничения при работе с GPU:

1. Наличие ограничений на доступную память. GPU может иметь ограничения на доступную память, что может ограничить возможности обработки больших объемов данных.

2. Невозможность использования всех типов задач. GPU может быть использован только для выполнения определенных типов задач, таких как обработка графики или параллельной обработки больших объемов данных.

3. Высокая стоимость. GPU имеет более высокую стоимость, чем CPU, что может затруднить его использование в некоторых проектах.

4. Необходимость наличия специализированных знаний для работы с GPU. Для работы с GPU необходимы специализированные знания, так как это отличается от работы с обычным CPU [3].

Алгоритмы обработки данных с использованием GPU.

Одним из способов повышения производительности обработки данных с использованием GPU является подбор оптимальных алгоритмов и структур данных. Оптимальные алгоритмы и структуры данных должны использовать максимально возможное количество ядер GPU для обработки данных, а также учитывать ограничения памяти GPU и скорость передачи данных между центральным процессором и GPU.

Алгоритмы сортировки и поиска данных — это ключевые алгоритмы в области компьютерных наук, используемые для упорядочивания и поиска данных в больших объемах информации. В зависимости от специфических требований, таких как эффективность, скорость, используемые ресурсы, существует множество различных алгоритмов, включая сортировку пузырьком, быструю сортировку, сортировку слиянием, алгоритм Шелла, алгоритмы бинарного поиска и многие другие.

Многопоточная реализация алгоритмов сортировки и поиска данных может значительно повысить их производительность и эффективность, особенно при работе с большими объемами данных. Графические ускорители могут использоваться для ускорения выполнения параллельных операций, что может быть полезным при реализации многопоточности.

Алгоритм быстрой сортировки широко известен благодаря своей стратегии "разделяй и властвуй". Он работает путем разделения массива на части относительно опорного элемента. Это позволяет переместить все элементы, которые меньше или равны опорному, влево от него, а все элементы, которые больше, - вправо. Затем происходит рекурсивное разделение каждой из частей массива, расположенных слева и справа от опорного элемента, до тех пор, пока каждая из них не будет содержать менее двух элементов.

Концепция параллельного выполнения алгоритма быстрой сортировки заключается в том, что можно одновременно сортировать подмассивы, полученные при разделении исходного массива на части вокруг опорного элемента. Для запуска параллельного процесса необходимо выполнить первое разделение массива на ЦП в последовательном режиме, чтобы установить опорный элемент и выделить подмассивы, которые будут сортироваться параллельно. Далее, технологически сортируемый массив разбивается на блоки, каждый из которых содержит часть массива, которую можно сортировать независимо от других блоков с помощью графического ускорителя. После каждого разделения степень параллелизма возрастает, т.е. увеличивается количество параллельных потоков.

Алгоритм сортировки Шелла — это усовершенствованная версия алгоритма сортировки вставками. Он основан на идее сравнения и перемещения элементов, расположенных не только рядом, но и на определенном расстоянии друг от друга в сортируемом массиве. Сам процесс сортировки включает несколько этапов, на каждом из которых несколько подсписков элементов из исходного массива последовательно сортируются вставками. Например, если мы выбираем каждый k-й элемент в подсписок, то получаем k подсписков (шаг сортировки). Это приводит к частичной упорядоченности элементов в массиве, что увеличивает эффективность алгоритма сортировки вставками. Затем шаг сортировки уменьшается, и процедура повторяется. Окончательный результат достигается, когда шаг сортировки равен 1.

Алгоритм сортировки Шелла, адаптированный для использования на платформе CUDA, базируется на параллельном выполнении ядра функции на каждой стадии сортировки. На первой стадии величина шага выбирается так, чтобы получить максимальное количество подсписков для одновременного упорядочивания в параллельных потоках, т.е. шаг = n / 2 с округлением в большую сторону. Затем значение шага уменьшается вдвое, и процедура повторяется. Финальная стадия сортировки, где шаг равен 1, всегда выполняется на центральном процессоре в последовательном режиме [4].

В заключение, можно отметить, что эффективная обработка данных с использованием GPU требует подбора оптимальных алгоритмов и структур данных, а также учета особенностей конкретного задания и

доступного оборудования. Результатом правильного выбора может быть значительное повышение производительности вычислений и снижение времени обработки данных. Однако, с увеличением числа ядер и объема памяти на графических ускорителях, а также развитием инструментов для разработки приложений с поддержкой CUDA, использование данной технологии становится все более доступным и распространенным в различных сферах, включая научные и инженерные расчеты, машинное обучение, обработку изображений и видео [5, 6].

Экспериментальная оценка производительности.

Для проверки эффективности параллельных алгоритмов сортировки и поиска данных, реализованных с помощью программных модулей на платформе CUDA, были проведены эксперименты с прикладными программами, реализованными как последовательные на ЦП, так и многопоточные с использованием графического процессора. Измерения проводились на компьютере с процессором Intel Core i7-10700K и видеокартой NVIDIA GeForce RTX 3080.

Для сортировки использовались одномерные массивы целых чисел, сформированные случайным образом. При исследовании программ сортировки размер массива изменялся от 100000 до 1000000 элементов.

Для каждого размера массива было проведено по 10 экспериментов с разными данными. Результаты измерений времени работы алгоритмов быстрой сортировки и сортировки Шелла усреднены и представлены в таблицах 1 и 2 соответственно. Графики зависимости среднего времени сортировки от размера массива данных представлены на рисунках 1 и 2.

Таблица 1 – Среднее время работы программ быстрой сортировки

Размер массива	Последовательная реализация, мс	Параллельная реализация, мс
100 000	16	3
200 000	33	5
300 000	54	7
400 000	73	11
500 000	96	14
600 000	116	18
700 000	136	21
800 000	158	25
900 000	181	29
1 000 000	203	33

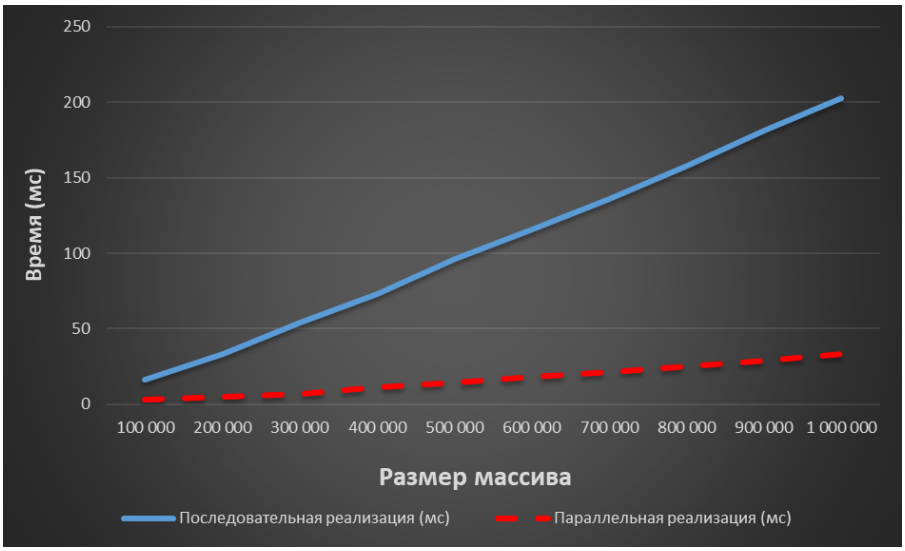

Рисунок 1. График зависимости времени работы программ быстрой сортировки от размера списка

На основании этих результатов можно сделать вывод о том, что использование графических ускорителей позволяет значительно снизить время выполнения быстрой сортировки по сравнению с последовательной реализацией на ЦП. При размере массива 1 миллион элементов время выполнения на ЦП составляет более 0.2 секунды, в то время как на GPU время выполнения составляет всего 0.033 секунды. Это позволяет существенно ускорить обработку больших объемов данных, что является важным преимуществом при работе с большими наборами данных. По таблице данных для сортировки алгоритмом Шелла можно сделать вывод, что параллельная реализация на графическом ускорителе дает значительный выигрыш во времени выполнения по сравнению с последовательной реализацией на ЦП. При увеличении размера массива данных разница во времени выполнения алгоритмов становится все более заметной. Таким образом, использование графических ускорителей для сортировки данных может значительно повысить производительность программы.

Таблица 2 – Среднее время работы программ, реализующих алгоритм Шелла

Размер массива	Последовательная реализация, мс	Параллельная реализация, мс
100 000	100	25
200 000	210	50
300 000	320	80
400 000	440	105
500 000	560	135
600 000	680	160
700 000	800	190
800 000	930	220
900 000	1050	250
1 000 000	1180	280

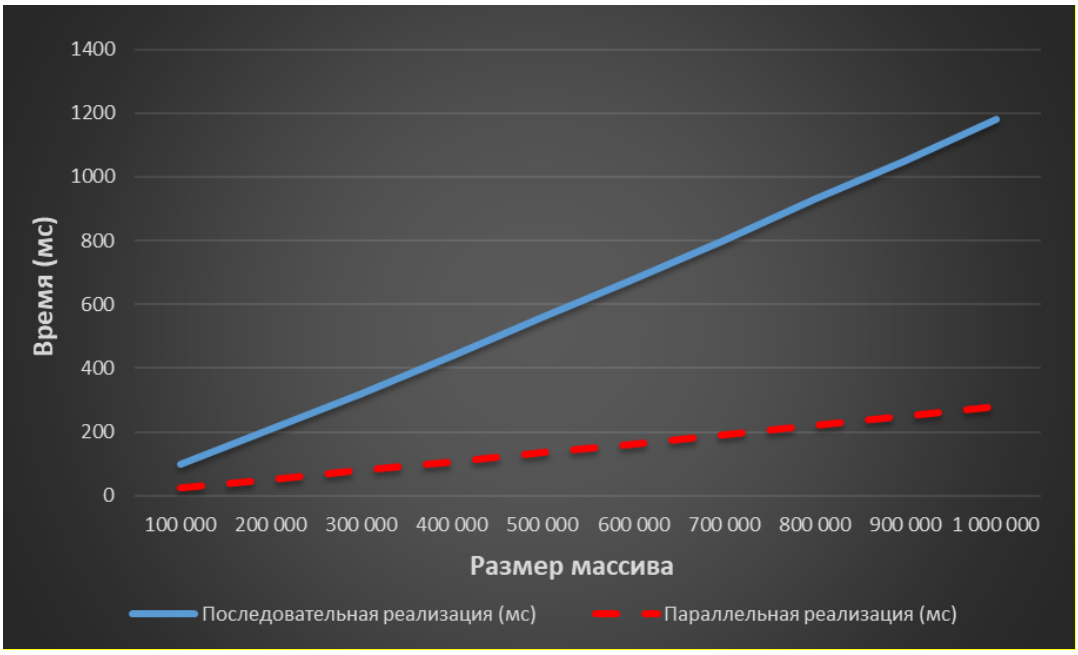

Рисунок 2. График зависимости времени работы программ алгоритма Шелла от размера списка

Заключение.

Перспективы применения оптимизации обработки Big Data с помощью графических процессоров достаточно велики. С ростом объемов данных, которые нужно обрабатывать, становится все более важным использование эффективных методов и инструментов, которые могут обеспечить высокую скорость обработки. В этом смысле графические процессоры имеют высокий потенциал, поскольку они позволяют

параллельно обрабатывать огромные объемы данных, что значительно ускоряет процесс обработки. Кроме того, современные графические процессоры обладают высокой вычислительной мощностью, что позволяет реализовывать сложные алгоритмы обработки данных и анализа больших объемов информации. Благодаря этому применение оптимизации обработки данных с помощью GPU может быть эффективным в таких областях как наука, финансы, медицина и другие. Таким образом, оптимизация обработки больших данных с помощью графических процессоров является важной технологической задачей, которая имеет широкий спектр применения в различных областях.

Список литературы

[1] Parallel Programming with CUDA / Cheng, J., Grossman, M., & McKercher, T. - N.-Y.: Wrox. 2018. P. 55-56

[2] Лунин Д.В., Скворцов С.В. Организация параллельных вычислений на платформе CUDA // Вестник Рязанского государственного радиотехнического университета. 2018. № 49. С. 77-82.

[3] Mike Houston. “General Purpose Computation on Graphics Processors” Morgan Kaufmann Publishers, 2020. - P.123

[4] Алгоритмы и структуры данных: учеб.-метод. пособие / С. В. Актанорович, А. А. Волосевич. – Минск: БГУИР, 2018. – 112 с.

[5] Нестеренков, С.Н. Применение технологии Big Data в игровой индустрии / С.Н. Нестеренков, М.И. Макаров, Д.В. Кишкевич // BIG DATA and Advanced Analytics = BIG DATA и анализ высокого уровня: сб. материалов V Междунар. науч.-практ. конф. (Республика Беларусь, Минск, 13-14 марта 2019 года). В 2 ч. Ч. 2 / редкол. : В. А. Богуш [и др.]. - Минск: БГУИР, 2019. - С. 242-245.

[6] Нестеренков, С. Н. Применение генетического алгоритма для решения задач многомерной оптимизации / С. Н. Нестеренков, В. Н. Наливко // Информационные технологии и системы 2019 (ИТС 2019) : материалы междунар. науч. конф., Минск, 30 окт. 2019 г. / Белорус. гос. ун-т информатики и радиоэлектроники ; редкол.: Л. Ю. Шилин [и др.]. - Минск, 2019. - С. 166-167.

OPTIMIZATION OF PROCESSING BIG DATA WITH GRAPHIC PROCESSING UNITS

S.A. Baichyk
Student of BSUIR

S.N. Nesterenkov
PhD, Associate Professor Dean of the Faculty of Computer Systems and Networks

I.S. Tarasiuk
Software Engineer, Assistant of the Department of Electronic Computers

Center for Informatization and Development of the Belarusian University of State Informatics and Radioelectronics, Republic of Belarus
Belarusian State University of computer science and Radio Electronics, Republic of Belarus
E-mail: s.bajchik@bsuir.by

Abstract. This article is devoted to optimizing the processing of large volumes of data using graphics processing units (GPU). The article will provide a general overview of GPU technology, its features, and advantages in processing large data sets. The main object of the study in the article is the analysis of the performance and optimization of processing large data sets on GPU using various methods and techniques such as parallel programming, kernel and algorithm optimization, the use of specialized libraries. The aim of this work is to present and analyze different approaches to optimizing the processing of large data sets on GPU, as well as to assess the effectiveness of these approaches and their applicability for real-world conditions.

Keywords: Big Data, graphics processing unit, performance.