МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №*1*
по курсу «Параллельная обработка данных»
Сортировка чисел на GPU. Свертка, сканирование, гистограмма.

Выполнила: *К.О. Михеева*
Группа: *8О-407Б*
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

Цель: Ознакомление с фундаментальными алгоритмами GPU: свертка (reduce), сканирование (blelloch scan) и гистограмма (histogram). Реализация одной из сортировок на CUDA. Использование разделяемой и других видов памяти. Исследование производительности программы с помощью утилиты nvprof.

Вариант 2: Сортировка подсчетом. Диапазон от 0 до 2^{24}-1.

Программное и аппаратное обеспечение

Name:	Tesla T4
Compute capability:	7.5
Dedicated video memory:	15835 MB
Shared memory per block:	49152 bytes
Register per block:	65536 bytes
Total constant memory:	65536
Max threads per multiprocessor:	2048
Max threads per block:	1024
Multiprocessors const:	40

AMD RYZEN 5 5500U

Architecture:	Ryzen 5 Zen2(Lucienne)
Processor Technology for CPU Cores:	TSMC 7nm FinFET
CPU Cores:	6
Thread(s):	12
CPU Socket	FP6
CPU MHz:	3600
SMP # CPUs:	1
L2 cache: L3 cache:	3MB 8MB
RAM	16GB
SSD	512GB

OS: Windows 10 - 64-Bit Edition, Ubuntu x86 64-Bit;
IDE: VS code;
Compiler: nvcc

Метод решения

В рамках выполнения лабораторной работы был реализован параллельный алгоритм сортировки слиянием на GPU. Процесс включает разбиение данных на блоки для параллельной обработки, вычисление гистограммы для каждого блока, применение префиксной суммы для определения смещений чисел в блоках, и окончательное формирование отсортированного массива. Этот подход позволяет эффективно использовать вычислительные ресурсы GPU, ускоряя процесс сортировки с использованием гистограмм и префиксных сумм.

Описание программы

Данная программа состоит из одного главного файла. В функции main происходит считывание исходных данных для дальнейшей обработки, а после завершения работы с ней выводит новые обработанные данные.

В функции ***Histogram*** создается гистограмма, где каждый элемент представляет собой количество вхождений определенного значения в массиве.

Далее в функции ***Scan*** выполняется сканирование (построение префиксной суммы) для гистограммы с использованием алгоритма префиксной суммы на GPU. Этот этап помогает эффективно распределить элементы в массиве.Алгоритм использует разделяемую память (*sharedMemory*) для эффективного совместного доступа к данным блока. В начале происходит загрузка данных из глобальной памяти в разделяемую память, затем выполняется итеративное вычисление префиксной суммы. После каждой итерации значения суммируются попарно, при этом используются конфликтов избегающие индексы. Затем значения префиксной суммы сохраняются в массив *prefix_sums*. После завершения вычислений происходит обновление исходных данных с использованием вычисленных префиксных сумм.

Дополнительно, есть функция ***updateDataUsingPrefixSum***, которая обновляет исходный массив данными из массива префиксных сумм, применяя вычисленные смещения.

Эта функция *scan* представляет интерфейс для выполнения параллельного вычисления префиксной суммы на GPU. Внутри функции происходит выделение памяти для массива префиксных сумм, вызов ядра *Scan* для расчета префиксной суммы, и рекурсивный вызов функции для обработки массива префиксных сумм. Затем применяется ядро *updateDataUsingPrefixSum* для обновления исходного массива с использованием вычисленных префиксных сумм. Если размер массива позволяет обработку в одном блоке, процесс завершается, и освобождается выделенная память.

Функция ***sort*** выделяет память для массивов *Hist* и *Result* на устройстве, копирует данные из исходного массива *data* в *Hist*, и запускает ядро *Histogram* для вычисления гистограммы. Затем вызывается функция scan для расчета префиксной суммы на массиве *Result*. Далее, ядро ***CountSort*** используется для сортировки массива *Hist* на основе вычисленной гистограммы и префиксной суммы. Наконец, отсортированные данные копируются обратно в массив *data*.

Результаты

Время исполнения программы в мс при различных данных

Размеры входных массивов:

В малом тесте - 10^2

В среднем тесте - 10^5

В большом тесте - 10^9

	Малый тест	Средний тест	Большой тест
<<<1, 32>>>	2090.22	2101.67	2093.09
<<<32, 32>>>	167.13	189.55	201.06
<<<128, 128>>>	167.31	150.23	187.78
<<<256, 256>>>	190.18	189.35	209.56
<<<512, 512>>>	309.22	289.95	306.98
CPU	1709.09	1899.45	4798.65

Изучение производительности программы было осуществлено при помощи инструмента NVPROF:

Информация из профилирования включает в себя:

1. Активность GPU: Время и статистику вызовов функций, выполняющихся на GPU. Это включает среднее, минимальное и максимальное время выполнения. Функция updateLines заняла большую часть времени выполнения.
2. Вызовы API: Статистика по вызовам функций в CUDA API, таких как cudaMalloc, cudaStreamSynchronize, cudaMemcpy и других.

```
==12993== NVPROF is profiling process 12993, command: ./lab5
==12993== Profiling application: ./lab5
==12993== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   58.59%  2.3285ms         3  776.15us  7.9030us  2.3090ms  Scan(unsigned int*,
unsigned int*, unsigned int)
                  23.46%  932.23us         2  466.12us  4.9280us  927.31us
updateDataUsingPrefixSum(unsigned int*, unsigned int*, unsigned int)
                  10.33%  410.61us         1  410.61us  410.61us  410.61us  CountSort(int*, unsigned int*,
unsigned int)
                   7.28%  289.31us         1  289.31us  289.31us  289.31us  [CUDA memset]
                   0.24%  9.4080us         1  9.4080us  9.4080us  9.4080us  Histogram(unsigned int*, int*,
unsigned int)
                   0.06%  2.2400us         1  2.2400us  2.2400us  2.2400us  [CUDA memcpy DtoH]
                   0.05%  1.8560us         1  1.8560us  1.8560us  1.8560us  [CUDA memcpy HtoD]
      API calls:   98.35%  255.78ms         5  51.156ms  4.4260us  255.49ms  cudaMalloc
```

1.25%	3.2599ms	3	1.0866ms	9.1740us	2.3157ms	cudaFree
0.17%	455.06us	2	227.53us	24.899us	430.16us	cudaMemcpy
0.11%	281.24us	1	281.24us	281.24us	281.24us	cudaDeviceSynchronize
0.06%	152.85us	101	1.5130us	197ns	60.741us	cuDeviceGetAttribute
0.03%	73.805us	7	10.543us	6.1300us	19.951us	cudaLaunchKernel
0.01%	32.088us	1	32.088us	32.088us	32.088us	cudaMemset
0.01%	28.877us	1	28.877us	28.877us	28.877us	cuDeviceGetName
0.00%	7.8750us	1	7.8750us	7.8750us	7.8750us	cuDeviceGetPCIBusId
0.00%	2.3830us	3	794ns	348ns	1.6710us	cuDeviceGetCount
0.00%	1.8810us	2	940ns	306ns	1.5750us	cuDeviceGet
0.00%	972ns	1	972ns	972ns	972ns	cuDeviceTotalMem
0.00%	627ns	1	627ns	627ns	627ns	cuModuleGetLoadingMode
0.00%	352ns	1	352ns	352ns	352ns	cuDeviceGetUuid

Выводы

В данной лабораторной работе представлено задание сортировки подсчетом, охватывающим диапазон 0 до 2^{24}-1, заключающееся в эффективной упорядоченной обработке больших массивов целых чисел.

Сортировка подсчетом, как и другие алгоритмы сортировки, находит применение в различных областях информатики и вычислительной техники. В данном случае, применение данного задания, которое включает в себя параллельную сортировку подсчетом на GPU с использованием CUDA, может быть обнаружено в следующих контекстах:

- **Большие базы данных** при работе с большими объемами данных, например, в базах данных, эффективные алгоритмы сортировки могут улучшить производительность запросов и операций.
- **Машинное обучение и анализ данных** в задачах обработки данных для машинного обучения, анализа данных или обработки сигналов, алгоритмы сортировки могут играть важную роль в улучшении производительности.
- **Обработка изображений** в приложениях обработки изображений, где необходимо обработать большое количество пикселей, параллельные алгоритмы сортировки могут использоваться для оптимизации некоторых процессов.

В данной лабораторной работе возникли сложности с заданием нужного размера гистограммы, чтобы программа эффективно и правильно считывала и сортировала данные. Пришлось завести размер гистограммы намного больше заданного диапазона. Также проблемы возникли при сканировании, где тоже неправильно сканировались данные.