МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №*5***

**по курсу «Программирование графических процессоров»**

***Сортировка чисел на GPU. Свёртка, сканирование, гистограмма.***

Выполнил: *А.Е. Аксенов*

Группа: *8О-408Б*

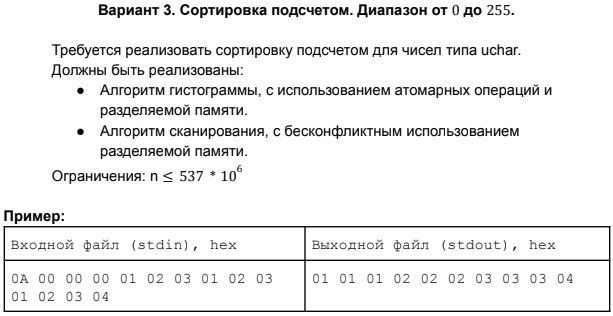
Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2023

**Условие**

Цель работы. Ознакомление c фундаментальными алгоритмами GPU: свёртка (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. Все входные-выходные данные являются бинарными и считываются из stdin и выводятся в stdout.

Входные данные. В первых четырёх байтах записывается целое число n – длина массива чисел, далее следуют n чисел типа заданного вариантом. Выходные данные. В бинарном виде записывают n отсортированных по возрастанию чисел.

**Программное и аппаратное обеспечение**

|  |  |
| --- | --- |
| Графический процессор | |
| GPU | NVIDIA RTX 2070 |
| Compute Capability | 7.5 |
| Объем видеопамяти | 8192 МБ |
| Тип видеопамяти | GDDR6 |
| Разделяемая память | 48 КБ |
| Константная память | 64 КБ |
| Число регистров в блоке | 65636 |
| Максимальное число блоков | 16 |
| Максимальное число нитей | 1024 |
| Количество мультипроцессоров | 34 |
| Центральный процессор | |
| СPU | Intel Core i5 9600K |
| Число физических ядер | 6 |
| Максимальное число потоков | 6 |
| Базовая частота | 3.6 GHz |
| Частота Turbo Boost | 4.5 GHz |
| Объем кэша L3 | 9 МБ |
| Оперативная память | |
| Тип оперативной памяти: | DDR4 |
| Частота памяти | 2666MHz |
| Объём | 32 ГБ |
| Постоянная память | |
| Тип постоянной памяти | SSD |
| Объём постоянной памяти | 1024ГБ |
| Программное обеспечение | |
| Операционная система | Microsoft Windows 11 |
| Версия операционной системы | 2004 |
| Среда разработки | Microsoft Visual Studio 2022 |
| Версия среды разработки | 16.7.3 |
| NVIDIA CUDA Toolkit | 12.2 |

**Метод решения**

1. Считываем числа входного массива.
2. Выделяем ресурсы и копируем входные данные.
3. Для массива строится гистограмма.
4. Для гистограммы строится скан.
5. Скан подаётся на вход сортировке подсчётом.
6. Вывод результатов в стандартный вывод.
7. Освобождение ресурсов и завершение работы.

**Описание программы**

Весь цикл разработки и отладки выполнялся в среде Microsoft Visual Studio с установленными CUDA SDK. Программа написана на языке C++, используются библиотеки. Для наибольшей эффективности для части алгоритмов используется разделяемая память. Особое внимание уделено исследованию паттернов доступа к разделяемой памяти и последующая реализация бесконфликтных обращений:

\_\_device\_\_ void sweep\_down(uint32\_t\* shmem, int32\_t tid, uint32\_t stride){ // фаза спускания сканирования

    uint32\_t r = offset(stride \* tid + stride – 1);

    uint32\_t l = offset(stride \* tid + (stride >> 1) – 1);

    shmem[r] += shmem[l];

}

\_\_device\_\_ void sweep\_up(uint32\_t\* shmem, int32\_t tid, uint32\_t stride){ // фаза подьема сканирования

    uint32\_t r = offset(stride \* tid + stride – 1);

    uint32\_t l = offset(stride \* tid + (stride >> 1) – 1);

    shmem[r] += shmem[l];

    shmem[l] = shmem[r] – shmem[l];

}

\_\_global\_\_ void scan\_kern\_bs128(uint32\_t\* hist) {                   // ядро сканирования (exclusive) – строит профиксные суммы

    \_\_shared\_\_ uint32\_t shmem[512];                                 // разделяемая память (немного больше тк используются смещения для избежания конфликтов банков памяти)

    shmem[offset(threadIdx.x      )] = hist[threadIdx.x      ];

    shmem[offset(threadIdx.x + 128)] = hist[threadIdx.x + 128];

    \_\_syncthreads();

    for(uint32\_t I = 2; I <= 256; I <<= 1){                         // фаза спуска

        if (I \* threadIdx.x < 256) {

            sweep\_down(shmem, threadIdx.x, i);

        }

        \_\_syncthreads();

    }

    if(threadIdx.x == 0) shmem[offset(255)] = 0; \_\_syncthreads();

    for(uint32\_t I = 256; I >= 2; I >>= 1){                         // фаза подьема

        if (I \* threadIdx.x < 256) {

            sweep\_up(shmem, threadIdx.x, i);

        }

        \_\_syncthreads();

    }

    hist[threadIdx.x      ] = shmem[offset(threadIdx.x      )];

    hist[threadIdx.x + 128] = shmem[offset(threadIdx.x + 128)];     // пишем обратно в DRAM из разделяемой памяти

    \_\_syncthreads();

}

\_\_global\_\_ void hist256\_kern(uint8\_t\* arr, int32\_t n,              // ядро гистограммы (подсчет встречаемости каждого числа в массиве)

                                        uint32\_t\* hist) {

    const int32\_t s = gridDim.x \* blockDim.x;

    for (int32\_t I = blockIdx.x \* blockDim.x

                + threadIdx.x; I < n; I += s) {

        atomicAdd(&hist[arr[i]], 1u);

    }

}

\_\_global\_\_ void sort256\_kern(uint8\_t\* arr, uint8\_t\* res,            // ядро сортировки массива (сортировка подсчетом)

                             int32\_t n, uint32\_t\* hist) {           // использует префиксные суммы которые сформировали ранее

    const int32\_t s = gridDim.x \* blockDim.x;

    for (int32\_t I = blockIdx.x \* blockDim.x

                + threadIdx.x; I < n; I += s) {

        uint8\_t num = arr[i];

        const int32\_t t = atomicAdd(&hist[num], 1);

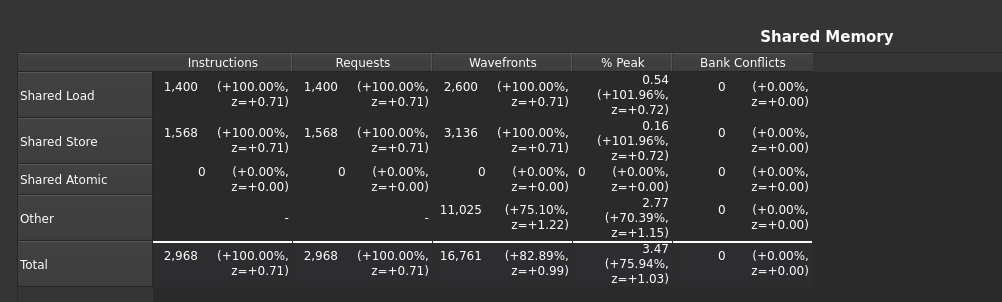
        res[t] = num;

    }

}

**Результаты**

Контрольные замеры выполнены для алгоритма в целом, т.е. учитывается время, затрачиваемое на построение префиксных сумм и саму сортировку. Для сравнения использовалась сортировка std::sort. Были добавлены смещения при обращениях к разделяемой памяти. В результате чего удалось устранить конфликты банков памяти и ускорить вычисления.



|  |  |  |
| --- | --- | --- |
| Размер массива | Параметры ядра | Время |
|  | <32, 64> | 2038 |
|  | <32, 128> | 2352 |
|  | <32, 256> | 1959 |
|  | <32, 512> | 1926 |
|  | <64, 64> | 1986 |
|  | <64, 128> | 2025 |
|  | <64, 256> | 1985 |
|  | <64, 512> | 1968 |
|  | <128, 64> | 2092 |
|  | <128, 128> | 2085 |
|  | <128, 256> | 2036 |
|  | <128, 512> | 1959 |
|  | <256, 64> | 1965 |
|  | <256, 128> | 2102 |
|  | <256, 256> | 2018 |
|  | <256, 512> | 2044 |

**Выводы**

Алгоритм сортировки подсчетом отличается от остальных тем что не использует в своей основе сравнений. Алгоритмическая сложность алгоритма - ). Реализация подобного алгоритма для работы на CPU тривиальная задача, тем не менее для реализации на GPU необходимо уметь реализовать эффективный алгоритм вычисления префиксных сумм и учесть специфику архитектуры графического ускорителя. В данной лабораторной работе удалось выполнить эти задачи и провести исследование паттернов доступа к разделяемой памяти. В ходе выполнения данной работы были учтены рекомендации от разработчиков Nvidia CUDA — [Optimizing Parallel Reduction in CUDA](https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf).