МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Параллельная обработка данных»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма.

Выполнил: А. О. Дубинин
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

Цель работы:

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 4. Сортировка чет-нечет.

Требуется реализовать блочную сортировку чет-нечет для чисел типа int.

Должны быть реализованы:

- Алгоритм чет-нечет сортировки для предварительной сортировки блоков.
- Алгоритм битонического слияния, с использованием разделяемой памяти.

Ограничения: $n \leq 16 * 10^6$

Программное и аппаратное обеспечение

GeForce 940MX

Compute capability:	5.0
Dedicated video memory:	4096 MB
shared memory per block:	49152 bytes
constant memory:	65536 bytes
Total number of registers available per block:	65536
Maximum number of threads per multiprocessor:	2048
Maximum number of threads per block:	1024
(3) Multiprocessors, (128) CUDA Cores/MP:	384 CUDA Cores

Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz

Architecture:	x86_64
Byte Order:	Little Endian
CPU(s):	4
Thread(s) per core:	2
Core(s) per socket:	2
CPU MHz: CPU max MHz: CPU min MHz:	713.848 3100,0000 400,0000
L1d cache: L1i cache: L2 cache: L3 cache:	64 KiB 64 KiB 512 KiB 3 MiB

RAM	8GiB SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0,4 ns)

SSD(SPCC_M.2_SSD)	223,6G
HDD(ST1000LM035-1RK172)	931,5G

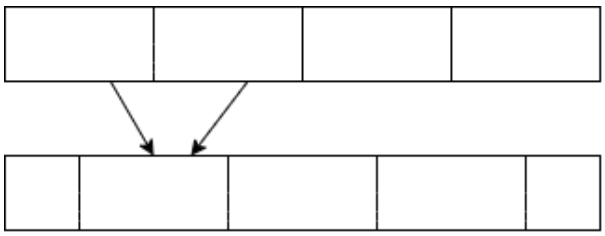
OS: Ubuntu 20.04 focal
IDE: jetbrains clion
compiler: nvcc

Метод решения

Реализация данной программы была итеративная, так как данная сортировка состоит из множества кусочков, которые можно писать отдельно.

Так для начала я дополнял данные, чтобы они были кратны 1024, так как сортировка у нас такая, чтобы каждый поток был ответственен за один элемент, а кол-во потоков в блоке может быть максимум 1024 на моей gpu. Дополнял я массив числом MAX_INT, а в конце сортировки я это кол-во чисел выкидывал из массива.

Реализовал классический вариант сортировки чет-нечет для предварительной сортировки с добавлением shared memory. Далее было реализовано битоническое слияние, которое идеально ложиться на наше общую блочную сортировку чет-нечет. Слияние происходило так, что сначала мы инвертировали данные из правого блока, так как для слияния битонического они должны убывать, эти данные мы инвертировали при перекопировании на shared memory. А далее просто запускали M1024, который отсортировывал наш блок поэтапно сравнивая наши элементы с использованием shared memory. Данное слияние мы выполняем нужное кол-во раз для каждого блока чередуя блоки меняя расположение блоков, как показано на диаграмме.



Описание программы

Рассмотрим код из чет-нечет предварительной сортировки блоков.
Обе части сортировки, предварительная сортировка и сливающая похожим образом сравнивают элементы, сначала каждый поток перекопирует один элемент (одно число)

из глобальной памяти в shared memory, а далее идет итерационный процесс, где на каждой итерации один поток сравнивает свой элемент с элементом другого потока, в то время как другой простаивает, а на следующей итерации они меняются. А в конце данные переносятся из shared memory в глобальную память и так для каждого блока. Лишь в конце при оптимизации я понял, что блоки могут сортироваться параллельно, так как shared memory позволяет нам поместить несколько блоков в свою память.

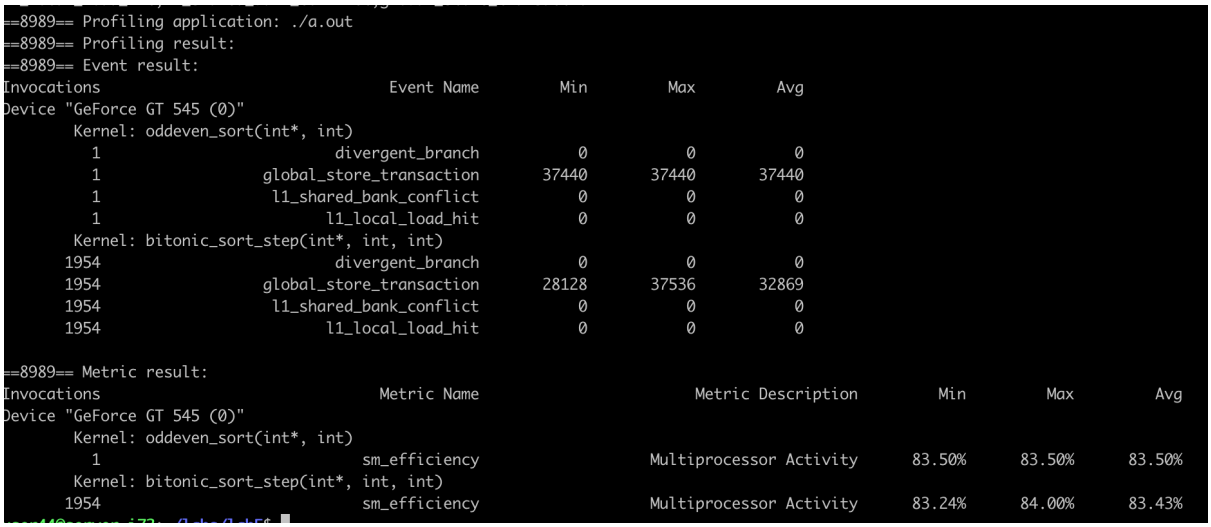
```
__global__ void oddeven_sort(int *dev_values, int sz) {
    __shared__ int shared[BUCKET_SIZE];

    int id = threadIdx.x;
    int block_id = blockIdx.x;
    int block_offset = gridDim.x;
    int odd, i, n = BUCKET_SIZE;
    int *values;

    for (int j = block_id * BUCKET_SIZE; j < sz; j = j + block_offset *BUCKET_SIZE)
{
        values = dev_values + j;
        shared[id] = values[id];
        __syncthreads();

        for (i = 0; i < n; i++) {
            odd = i % 2;
            if (odd == 0 && id % 2 == 0 && id + 1 < n) {
                if (shared[id] > shared[id + 1]) {
                    int tmp = shared[id];
                    shared[id] = shared[id + 1];
                    shared[id + 1] = tmp;
                }
            }
            if (odd == 1 && id % 2 == 1 && id + 1 < n) {
                if (shared[id] > shared[id + 1]) {
                    int tmp = shared[id];
                    shared[id] = shared[id + 1];
                    shared[id + 1] = tmp;
                }
            }
            __syncthreads();
        }
        values[id] = shared[id];
    }
}
```

Результаты

Сравним результаты на различном кол-ве блоков и с разным кол-вом входных данных. Посмотрев на результаты, мы можем сделать вывод, что отличия при сортировки в 5 блоков и в 10 блоков не значительны, когда на одном блоке программа работает устрашающее медленно, особенно на больших данных.

	INPUT	BLOCKS	THREADS	TIME
1	1mln	1	1024	12363.7
2		5		4966.13
3		10		4961.44
4	4mln	1	1024	192602
5		5		77222.3
6		10		77203.6

Исследование производительности программы с помощью утилиты nvprof на 1млн данных.



```
==8989== Profiling application: ./a.out
==8989== Profiling result:
==8989== Event result:
Invocations                                Event Name         Min         Max         Avg
Device "GeForce GT 545 (0)"
       Kernel: oddeven_sort(int*, int)
         1                           divergent_branch           0           0           0
         1                  global_store_transaction       37440       37440       37440
         1                   l1_shared_bank_conflict           0           0           0
         1                         l1_local_load_hit           0           0           0
       Kernel: bitonic_sort_step(int*, int, int)
      1954                           divergent_branch           0           0           0
      1954                  global_store_transaction       28128       37536       32869
      1954                   l1_shared_bank_conflict           0           0           0
      1954                         l1_local_load_hit           0           0           0

==8989== Metric result:
Invocations                                Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GT 545 (0)"
       Kernel: oddeven_sort(int*, int)
         1                               sm_efficiency                   Multiprocessor Activity      83.50%      83.50%      83.50%
       Kernel: bitonic_sort_step(int*, int, int)
      1954                               sm_efficiency                   Multiprocessor Activity      83.24%      84.00%      83.43%
```

Как мы видим, sm_efficiency очень даже высока. Так же хочется отметить, что нет дивергенции нитей, и конфликтов банков памяти.

Вывод

Данная ЛР мне была интересна с точки зрения новизны информации даже среди сортировок. Да, сортировки подходят только для параллельных вычислений. Так же было интересно поработать с shared memory, и написать код, где каждый поток ответственен за один элемент.