МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №1**

**по курсу «Параллельная обработка данных»**

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

Выполнил: Гамов П.А.

Группа: 8О-407Б-18

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2022

**Условие**

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 ≤ 16 \* 10^6

**Программное и аппаратное обеспечение**

nvcc 7.0

Ubuntu 14.04 LTS

|  |  |
| --- | --- |
| Compute capability | 6.1 |
| Name | GeForce GTX 1050 |
| Total Global Memory | 2096103424 |
| Shared Mem per block | 49152 |
| Registers per block | 65534 |
| Max thread per block | (1024,1024,64) |
| Max block | (2147483647, 65535, 65535) |
| Total constant memory | 65536 |
| Multiprocessor’s count | 5 |

**Метод решения**

Для начала увеличим фиктивный размер массива, так, чтобы в нем помещалось кол-во элементов равное кол-ву потоков в блоке. В конце мы просто не будем выводить лишние значения. Потом для каждого блока, отсортируем их по возрастанию, так, чтобы в каждом блоке последовательность цифр возрастала. Далее применим битоническое слияние этих подмассивов. Таким образом мы получаем отсортированный массив.

**Описание программы**

К примеру, кол-во блоков равно 10, а потоков в блоке поставим равным 1024. Первой операцией мы делаем сортировку чет-нечет внутри каждого блока по 1024 элемента.

\_\_global\_\_ void oddEvenSortingStep(int \* A, int i, int n, int batch) {

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

int shift = blockDim.x \* gridDim.x;

for (int start = idx \* batch; start < n; start += shift \* batch)

for (int j = start + (i % 2); j + 1 < min(start + batch, n); j += 2)

if (A[j] > A[j + 1])

thrust::swap(A[j], A[j + 1]);

}

Далее уже вызываем ядро для сортировки блоков.

\_\_global\_\_ void bitonic\_merge(int \* arr, int size, bool is\_odd) {

int \* tmp = arr;

if (is\_odd)

swp(arr, tmp, size, (BLOCK\_SIZE / 2) + blockIdx.x \* BLOCK\_SIZE, size - BLOCK\_SIZE, gridDim.x \* BLOCK\_SIZE, threadIdx.x);

else

swp(arr, tmp, size, blockIdx.x \* BLOCK\_SIZE, size, gridDim.x \* BLOCK\_SIZE, threadIdx.x);

}

Которое вызывает другую функцию на девайсе, которая сортирует сами блоки между собой, таким образом через upd\_n / BLOCK\_SIZE итераций, все элементы всех подмассивов окажутся на своем месте в своем блоке.

\_\_device\_\_ void swp(int\* nums, int\* tmp, int size, int start, int stop, int step, int i) {

\_\_shared\_\_ int sh[BLOCK\_SIZE];

for (int shift = start; shift < stop; shift += step) {

tmp = nums + shift;

if (i >= BLOCK\_SIZE / 2)

sh[i] = tmp[BLOCK\_SIZE \* 3 / 2 - 1 - i];

else

sh[i] = tmp[i];

\_\_syncthreads();

for (int j = BLOCK\_SIZE / 2; j > 0; j /= 2) {

unsigned int XOR = i ^ j;

if (XOR > i) {

if ((i & BLOCK\_SIZE) != 0) {

if (sh[i] < sh[XOR])

thrust::swap(sh[i], sh[XOR]);

} else {

if (sh[i] > sh[XOR])

thrust::swap(sh[i], sh[XOR]);

}

}

\_\_syncthreads();

}

tmp[i] = sh[i];

}

}

**Результаты**

|  |  |  |
| --- | --- | --- |
|  | 10^5 | 10^6 |
| 10, 1024 | 1.3 sec | 10.9 sec |
| 256, 1024 | 1.3 sec | 9.1 sec |
| 1024, 1024 | 1.3 sec | 10.1 sec |

nvprof ./a.out < data.t > res.t

==7290== NVPROF is profiling process 7290, command: ./a.out

==7290== Profiling application: ./a.out

==7290== Profiling result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time(%) | Time | Calls | Avg | Min | Max | Name |
| 56.26% | 5.95006s | 1954 | 3.0451ms | 2.9646ms | 3.1484ms | bitonic\_merge(int\*, int, bool) |
| 43.72% | 4.62414s | 1024 | 4.5158ms | 4.3277ms | 4.7561ms | oddEvenSortingStep(int\*, int, int, int) |
| 0.01% | 694.66us | 1 | 694.66us | 694.66us | 694.66us | [CUDA memcpy DtoH] |
| 0.01% | 612.73us | 1 | 612.73us | 612.73us | 612.73us | [CUDA memcpy HtoD] |

==7290== API calls:

Time(%) Time Calls Avg Min Max Name

70.46% 7.49946s 2978 2.5183ms 4.0590us 4.7571ms cudaLaunch

28.89% 3.07515s 2 1.53758s 553.02us 3.07460s cudaMemcpy

0.63% 66.795ms 1 66.795ms 66.795ms 66.795ms cudaMalloc

0.01% 1.4034ms 9958 140ns 107ns 4.3230us cudaSetupArgument

0.01% 694.08us 2978 233ns 170ns 1.3260us cudaConfigureCall

0.00% 423.69us 83 5.1040us 217ns 182.98us cuDeviceGetAttribute

0.00% 155.62us 1 155.62us 155.62us 155.62us cudaFree

0.00% 119.77us 1 119.77us 119.77us 119.77us cuDeviceTotalMem

0.00% 46.562us 1 46.562us 46.562us 46.562us cuDeviceGetName

0.00% 2.0990us 2 1.0490us 545ns 1.5540us cuDeviceGetCount

0.00% 1.0040us 2 502ns 259ns 745ns cuDeviceGet

0.00% 825ns 2 412ns 320ns 505ns cudaGetLastError

**Выводы**

В этой работе я познакомился и научился реализовывать сортировку чет-нечет. Смог отладить и ускорить программу используя nvprof, который показывал проценты затрат времени на каждую функцию. Таким образом, данная утилита является основополагающей для отладки и исправлению ошибок, которые возникают во время написания программы.