МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 810 «Информационные технологии в моделировании и управлении»

Курсовая работа
по курсу «Основы Python, Java и Scala, платформы CUDA для анализа данных»

Сортировки на GPU. Reduce, scan, histogram.

Выполнил: А.С.Бобряков
Группа: М8О-103М-19
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

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. Все входные-выходные данные являются бинарными.

Вариант 2 - Сортировка подсчетом.

Требуется реализовать сортировку подсчетом для чисел типа int.

Должны быть реализованы:

● Алгоритм гистограммы, с использованием атомарных операций.

● Алгоритм сканирования для любого размера, с рекурсией и бесконфликтным использованием разделяемой памяти.

Программное и аппаратное обеспечение

Видеокарта: NVIDIA GeForce GTX 1060 3Gb

Компоненты	Подробности	
GeForce GTX 1060 3GB	Версия драйвера:	441.22
	Тип драйвера:	Standard
	Версия API Direct3D:	12
	Уровень возможносте...	12_1
	Ядра CUDA:	1152
	Тактовая частота гра...	1594 МГц
	Скорость передачи д...	8.01 Гбит/с
	Интерфейс памяти:	192 бит
	Пропускная способнос...	192.19 ГБ/с
	Доступная графическ...	11237 МБ
	Выделенная видеопам...	3072 МБ GDDR5
	Системная видеопамя...	0 МБ
	Разделяемая системна...	8165 МБ
	Версия BIOS видео:	86.06.3C.00.7D
	IRQ:	Not used

Процессор: Intel® Core™ i7-8700K CPU @ 3.70GHz

Другое: ОС Windows, IDE – Clion EAP,

Метод решения

Реализовано два этапа алгоритма: построение гистограммы и “скан”. По результатам гистограммы получаем счетчики, которые инкрементируются для массива входных данных по каждому числу. По результатам операции “скан” реализуем построение дерева частичных сумм и построение результирующего массива по дереву сумм.

Описание программы

На вход программы поступает бинарный файл с массивом, который необходимо отсортировать. Реализовано ядро гистограммы для построения гистограммы массива.

Далее были реализованы ядра для этапа сканирования и построения деревьев частичных сумм с ядром построения результирующего массива.

Исследовательская часть и результаты

Таблица замеров времени работы:

Число потоков / Число блоков	32	128	256	512	1024
32	42.965 мс	27.548 мс	26.798 мс	22.156 мс	23.156 мс
128	19.165 мс	18.616 мс	23.191 мс	20.561 мс	17.736 мс
256	15.616 мс	12.812 мс	12.649 мс	12.344 мс	14.684 мс
512	13.589 мс	11.513 мс	11.239 мс	12.523 мс	12.347 мс
1024	12.256 мс	11.462 мс	11.833 мс	13.916 мс	12.481 мс

Графики:

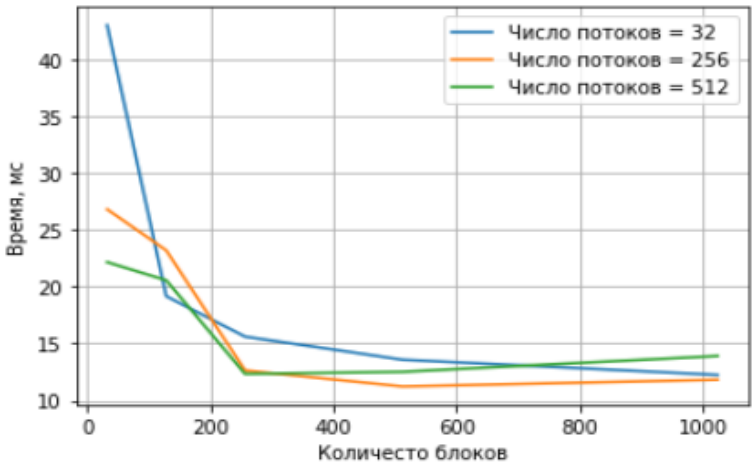
```
import matplotlib.pyplot as plt
plt.xlabel('Количесто блоков')
plt.ylabel('Время, мс')
x = [32, 128, 256, 512, 1024]

y= [42.965,19.165,15.616,13.589,12.256]
plt.plot(x, y, label="Число потоков = 32")

y= [26.798,23.191,12.649,11.239,11.833]
plt.plot(x, y, label="Число потоков = 256")

y= [22.156,20.561,12.344,12.523,13.916]
plt.plot(x, y, label="Число потоков = 512")

plt.legend()
plt.grid()
```



Результат выполнения команды nvprof main.cu

```
==5994== Profiling application: ./a.out
==5994== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 33.81%  198.82ms         1  198.82ms  198.82ms  198.82ms  get_result_kernel(int*, int*, int*, int)
 30.96%  182.02ms         1  182.02ms  182.02ms  182.02ms  [CUDA memcpy HtoD]
 15.54%  91.350ms         1  91.350ms  91.350ms  91.350ms  [CUDA memcpy DtoH]
 10.07%  59.207ms         1  59.207ms  59.207ms  59.207ms  histogram_kernel(int*, int, int*)
  6.84%  40.239ms         3  13.413ms  6.9850us  40.150ms  scan_kernel(int*, int*, int*, int)
  2.10%  12.341ms         3  4.1137ms  1.8750us  12.314ms  check_kernel(int*, int*)
  0.68%  4.0136ms         1  4.0136ms  4.0136ms  4.0136ms  array_clear_kernel(int*, int)
  0.00%  2.3040us         1  2.3040us  2.3040us  2.3040us  [CUDA memcpy DtoD]

==5994== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 67.07%  472.41ms         3  157.47ms  20.930us  290.33ms  cudaMemcpy
 16.72%  117.79ms        10  11.779ms  9.6260us  103.40ms  cudaFree
 16.11%  113.44ms        10  11.344ms  7.1070us  106.58ms  cudaMalloc
  0.05%  380.55us        83  4.5840us     146ns  165.23us  cuDeviceGetAttribute
  0.02%  138.60us         9  15.399us  10.093us  36.442us  cudaLaunch
  0.01%  78.703us         1  78.703us  78.703us  78.703us  cuDeviceTotalMem
  0.01%  47.749us         1  47.749us  47.749us  47.749us  cuDeviceGetName
  0.00%  15.711us        27     581ns     248ns  6.6990us  cudaSetupArgument
  0.00%  5.7480us         9     638ns     379ns  1.7840us  cudaConfigureCall
  0.00%  3.7990us         9     422ns     362ns     567ns  cudaGetLastError
  0.00%  1.8290us         2     914ns     843ns     986ns  cuDeviceGetCount
  0.00%     555ns         2     277ns     262ns     293ns  cuDeviceGet
```

Результаты выполнения команды nvprof -e divergent_branch, global_store_transaction, l1_shared_bank_conflict, l1_local_load_hit ./a.out :

```
user17@server-i72:~/ab$ nvprof -e divergent_branch,global_store_transaction,l1_shared_bank_conflict,l1_local_load_hit ./a.out
==6057== NVPROF is profiling process 6057, command: ./a.out
==6057== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==6057== Replaying kernel "array_clear_kernel(int*, int)" (done)
==6057== Replaying kernel "histogram_kernel(int*, int, int*)" (done)
==6057== Replaying kernel "scan_kernel(int*, int*, int*, int)" (done)
==6057== Replaying kernel "scan_kernel(int*, int*, int*, int)" (done)
==6057== Replaying kernel "scan_kernel(int*, int*, int*, int)" (done)
==6057== Replaying kernel "check_kernel(int*, int*)" (done)
==6057== Replaying kernel "check_kernel(int*, int*)" (done)
==6057== Replaying kernel "check_kernel(int*, int*)" (done)
==6057== Replaying kernel "get_result_kernel(int*, int*, int*, int)" (done)
==6057== Warning: The following aggregate event values were extrapolated from limited profile data and may therefore be inaccurate. To see the non-aggregate event values, use
 "--aggregate-mode off".
l1_local_load_hit,l1_shared_bank_conflict,global_store_transaction
==6057== Profiling application: ./a.out
==6057== Profiling result:
==6057== Event result:
Invocations                                Event Name         Min         Max         Avg
Device "GeForce GT 545 (0)"
       Kernel: check_kernel(int*, int*)
         3                           divergent_branch           0           0           0
         3                      global_store_transaction          48     1055568      352560
         3                       l1_shared_bank_conflict           0           0           0
         3                             l1_local_load_hit           0           0           0
       Kernel: get_result_kernel(int*, int*, int*, int)
         1                           divergent_branch           0           0           0
         1                      global_store_transaction    44683608    44683608    44683608
         1                       l1_shared_bank_conflict           0           0           0
         1                             l1_local_load_hit           0           0           0
       Kernel: histogram_kernel(int*, int, int*)
         1                           divergent_branch           0           0           0
         1                      global_store_transaction           0           0           0
         1                       l1_shared_bank_conflict           0           0           0
         1                             l1_local_load_hit           0           0           0
       Kernel: scan_kernel(int*, int*, int*, int)
         3                           divergent_branch           0           0           0
         3                      global_store_transaction          51     1113993      372079
         3                       l1_shared_bank_conflict           0           0           0
         3                             l1_local_load_hit           0           0           0
       Kernel: array_clear_kernel(int*, int)
         1                           divergent_branch           1           1           1
         1                      global_store_transaction     1179648     1179648     1179648
         1                       l1_shared_bank_conflict           0           0           0
         1                             l1_local_load_hit           0           0           0
```

Результат выполнения команды nvprof –m m_efficiency ./a.out :

```
user17@server-i72:~/ab$ nvprof -m sm_efficiency ./a.out
==6074== NVPROF is profiling process 6074, command: ./a.out
==6074== Profiling application: ./a.out
==6074== Profiling result:
==6074== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GT 545 (0)"
       Kernel: check_kernel(int*, int*)
         3                            sm_efficiency                   Multiprocessor Activity      10.94%      99.51%      63.35%
       Kernel: get_result_kernel(int*, int*, int*, int)
         1                            sm_efficiency                   Multiprocessor Activity      99.27%      99.27%      99.27%
       Kernel: histogram_kernel(int*, int, int*)
         1                            sm_efficiency                   Multiprocessor Activity      99.71%      99.71%      99.71%
       Kernel: scan_kernel(int*, int*, int*, int)
         3                            sm_efficiency                   Multiprocessor Activity      22.37%      99.99%      72.41%
       Kernel: array_clear_kernel(int*, int)
         1                            sm_efficiency                   Multiprocessor Activity      90.32%      90.32%      90.32%
user17@server-i72:~/ab$
```

Заключение

В ходе курсовой работы был исследован метод сортировки подсчетом, реализован код сортировки на CUDA, а также изучен профилировщик для анализа эффективности программы и дальнейшей ее оптимизации.