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: М8O-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.

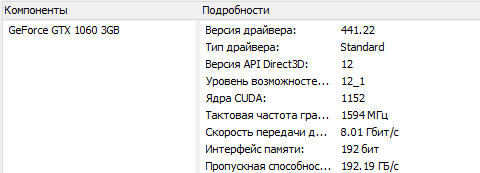
Должны быть реализованы:

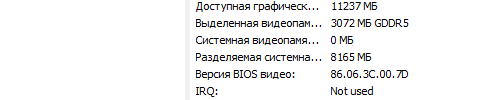
● Алгоритм гистограммы, с использованием атомарных операций.

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

**Программное и аппаратное обеспечение**

**Видеокарта**: NVIDIA GeForce GTX 1060 3Gb





**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.

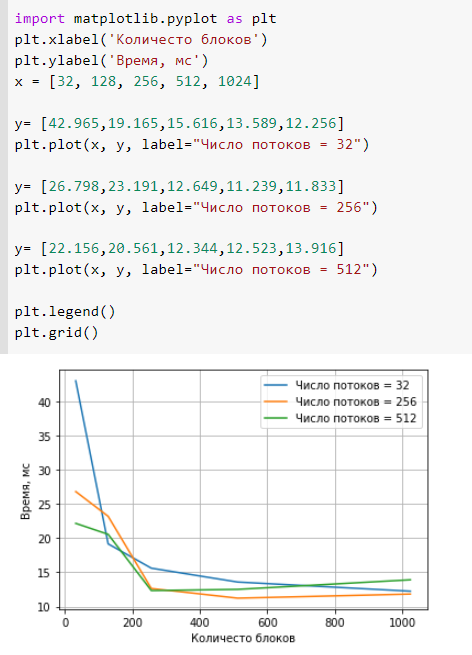
Далее были реализованы ядра для этапа сканирования и построения деревьев частичных сумм с ядром построения результирующего массива.

**Исследовательская часть и результаты**

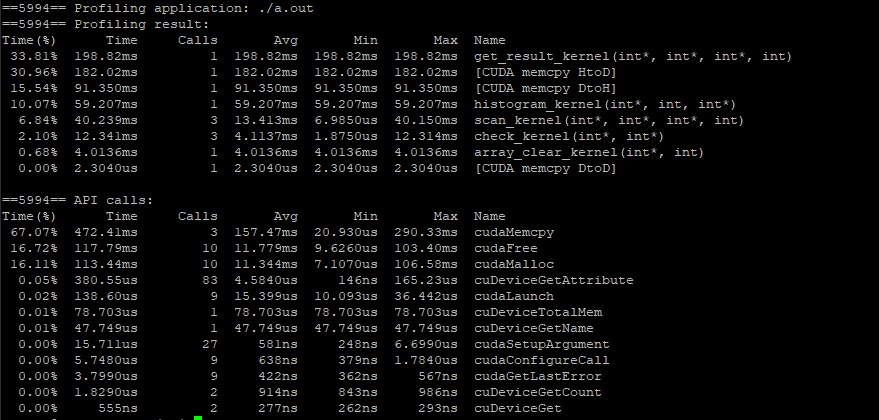
Таблица замеров времени работы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

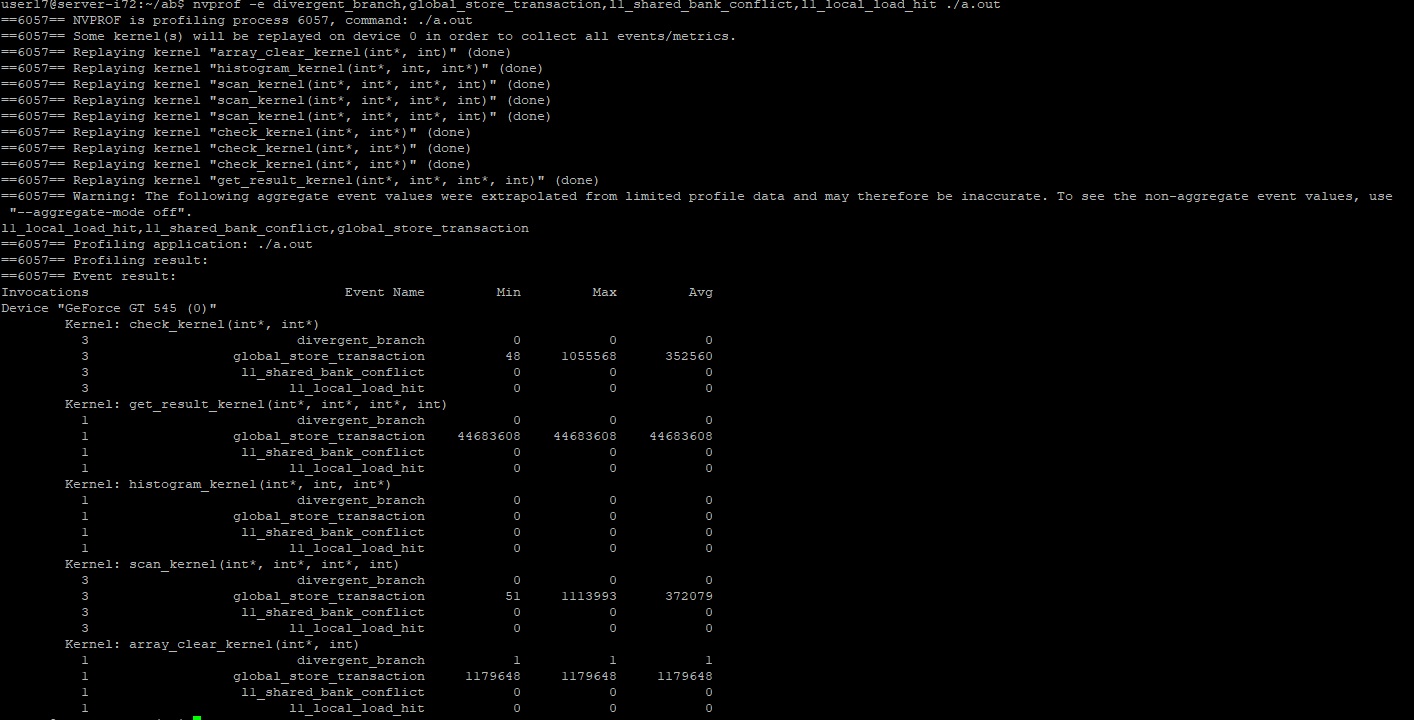
Графики:



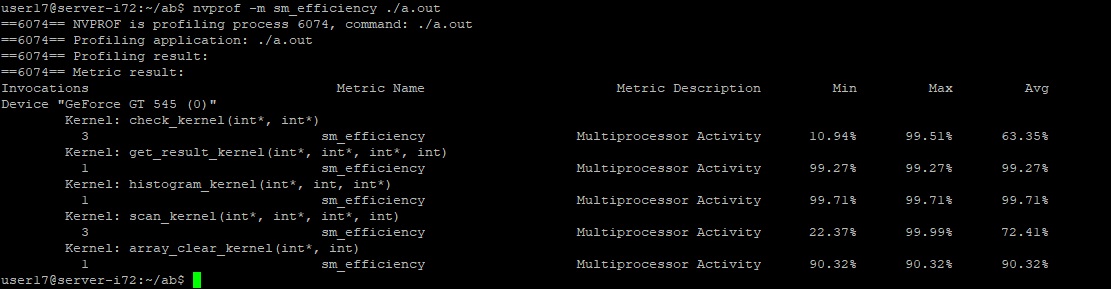
Результат выполнения команды nvprof main.cu



Результаты выполнения команды nvprof -e divergent\_branch, global\_store\_transaction, l1\_shared\_bank\_conflict, l1\_local\_load\_hit ./a.out :



Результат выполнения команды nvprof –m m\_efficiency ./a.out :



**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.