

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. Тояков

Группа: М8О-407Б-18

Преподаватели: К. Г. Крашенинников,
А. Ю. Морозов

Москва, 2022

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.

Все входные-выходные данные являются бинарными и считываются из **stdin** и выводятся в **stdout**.

Входные данные. В первых четырех 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.

Пример входных-выходных данных. Десять чисел типа `int`, от 0 до 9.

Входной файл (`stdin`), hex:

```
0A000000 00000000 09000000 08000000 07000000 06000000 05000000
04000000 03000000 02000000 01000000
```

Выходной файл (`stdout`), hex:

```
00000000 01000000 02000000 03000000 04000000 05000000 06000000
07000000 08000000 09000000
```

Вариант на “два”. Сортировка подсчетом.

Вариант №2, с использованием алгоритма сканирования из библиотеки Thrust.

ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

МЕТОД РЕШЕНИЯ

Сортировка подсчётом содержит 2 фундаментальных алгоритма – гистограмму и сканирование. Последний, согласно варианту, можно было использовать из библиотеки thrust, поэтому я реализовал только гистограмму для больших чисел с использованием глобальной памяти. Вначале мы находим максимальный элемент в исходном массиве, затем создаём массив гистограммы размера $[0, \text{max}]$ и заполняем его нулями. После применяем алгоритм гистограммы и на полученном массиве используем включающее сканирование. Затем остаётся лишь записать данные в результирующий массив по формуле, используя атомарную операцию сложения.

ОПИСАНИЕ ПРОГРАММЫ

Макрос **CSC** отвечает за отслеживание ошибок в функциях cuda, поэтому все cuda-вызовы оборачиваются в него и при `cudaError_t != cudaSuccess` выводится сообщение об ошибке.

`__global__ void histogram()` – алгоритм гистограммы с использованием глобальной памяти.

`__global__ void out()` – записать результаты на GPU в результирующий массив.

`void counting_sort()` – сортировка подсчётом.

`int maximum()` – функция нахождения максимума в массиве.

`int main()` – отвечает за ввод, и перенос данных на GPU и вывод.

ТЕСТЫ ПРОИЗВОДИТЕЛЬНОСТИ

В обеих программах числа для массивов генерировались рандомно в промежутке [0, 1000].

Работа на GPU:

Тест: (размер массива)	Результат:			
	<<<64, 64>>>	<<<256, 256>>>	<<<512, 512>>>	<<<1024, 1024>>>
1000	time = 0.042272	time = 0.045408	time = 0.052128	time = 0.108704
10000	time = 0.048704	time = 0.058880	time = 0.055680	time = 0.119840
100000	time = 0.125824	time = 0.120960	time = 0.135936	time = 0.177216

Работа на CPU:

Тест:	Результат:
1000	time = 0.065
10000	time = 0.391
100000	time = 1.174

ПРОФИЛИРОВКА

По запросу `nvprof -query-metrics` нужные метрики не были найдены на моём устройстве. Соответственно профилировщик сообщил об этом при попытке посчитать требуемые метрики.

```
==8425== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	35.94%	133.82us	1	133.82us	133.82us	133.82us	[CUDA memcpy HtoD]
	31.80%	118.62us	1	118.62us	118.62us	118.62us	[CUDA memcpy DtoH]
	21.95%	81.727us	1	81.727us	81.727us	81.727us	out(int*, int*, int*, int)
	8.27%	30.785us	1	30.785us	30.785us	30.785us	histogram(int*, int*, int)
	1.07%	3.9679us	1	3.9679us	3.9679us	3.9679us	void cub::DeviceScanKernel<cub::AgentScanPolicy<int=128, int=12, int, cub::BlockLoadAlgorithm, cub::CacheLoadModifier, cub::BlockStoreAlgorithm, cub::BlockScanAlgorithm, cub::NewBoundScaling<int=128, int=12, int>, thrust::device_ptr<int>, thrust::device_ptr<int>, cub::ScanFileState<int, bool=1>, thrust::plus<void>, cub::NullType, int>(>int=12, int, cub::BlockLoadAlgorithm, int, cub::CacheLoadModifier, cub::BlockStoreAlgorithm, cub::BlockScanAlgorithm)
	0.62%	2.3050us	1	2.3050us	2.3050us	2.3050us	void cub::DeviceScanInitKernel<cub::ScanFileState<int, bool=1>(>int, int)
	0.30%	1.1200us	1	1.1200us	1.1200us	1.1200us	[CUDA memset]
API calls:	90.83%	101.93ms	4	25.476ms	2.1570us	101.90ms	cudaMalloc
	0.43%	446.90us	2	223.45us	128.02us	318.88us	cudaMemcpy
	0.24%	247.13us	4	61.792us	4.9110us	198.77us	cudaFree
	0.19%	197.33us	101	1.9530us	117ns	69.706us	cuDeviceGetAttribute
	0.11%	116.63us	1	116.63us	116.63us	116.63us	cudaStreamSynchronize
	0.09%	93.246us	1	93.246us	93.246us	93.246us	cuDeviceTotalMem
	0.04%	42.679us	1	42.679us	42.679us	42.679us	cuDeviceGetName
	0.03%	33.807us	4	8.4710us	3.0470us	17.853us	cudaLaunchKernel
	0.01%	7.1870us	1	7.1870us	7.1870us	7.1870us	cuDeviceGetPCIBusId
	0.00%	4.4200us	1	4.4200us	4.4200us	4.4200us	cudaMemset
	0.00%	3.0160us	30	100ns	80ns	161ns	cudaGetLastError
	0.00%	2.9050us	1	2.9050us	2.9050us	2.9050us	cudaFuncGetAttributes
	0.00%	1.8860us	1	1.8860us	1.8860us	1.8860us	cudaOccupancyMaxActiveBlocksPerMultiprocessorWithFlags
	0.00%	1.4529us	5	290ns	223ns	537ns	cuDeviceGet
	0.00%	1.1560us	1	1.1560us	1.1560us	1.1560us	cuDeviceGetAttribute
	0.00%	1.0050us	3	361ns	166ns	728ns	cuDeviceGetCount
	0.00%	687ns	2	343ns	114ns	573ns	cuDeviceGet
	0.00%	486ns	4	121ns	93ns	185ns	cudaPeekAtLastError
	0.00%	315ns	1	315ns	315ns	315ns	cudaGetDeviceCount
	0.00%	224ns	1	224ns	224ns	224ns	cuDeviceGetUuid

```

artoy@artoy:~/Desktop/PAI/graphics processor programming/lab5$ nuprof -e divergent_branch,global_store_transaction,li-local-load_hit,li-shared_bank_conflict -n sm efficiency ./a.out < test
----- Warning: Event "global_store_transaction" cannot be found on device 0.
----- Warning: Event "li-local-load_hit" cannot be found on device 0.
----- Warning: Event "li-shared_bank_conflict" cannot be found on device 0.
==8509== NUPROF is profiling process 8509, command: ./a.out
==8509== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
ERROR in main.cu:37. Message: unknown error
==8509== Warning: ERR_NUPROFPERM - The user does not have permission to profile on the target device. See the following link for instructions to enable permissions and get more information: https://developer.nvidia.com/ERR_NUPROFPERM
==8509== Profiling application: ./a.out
==8509== Profiling result:
No events/metrics were profiled.
==8509== Warning: Some profiling data are not recorded.
artoy@artoy:~/Desktop/PAI/graphics processor programming/lab5$

```

Несмотря на то, что на девайсе не было метрик, я думаю, что ошибок в моей программе быть не должно, т. к. на всех этапах алгоритма используются либо атомарные операции, либо функции из библиотеки thrust, а разделяемая память вообще не применялась.

ВЫВОДЫ

Сортировку подсчётом легко распараллелить, так как все этапы алгоритма можно выполнить на GPU, используя фундаментальные алгоритмы и средства технологии CUDA. По результатам тестов производительности видно, что программа на девайсе работает в разы быстрее программы на хосте. Однако, данная сортировка не самая универсальная. Требуется, чтобы входные данные были целочисленными, а также необходимо находить максимальный элемент, что занимает дополнительное время работы, но в благоприятных условиях программа обрабатывает за $O(n)$, что хорошо.