Гистограмма текста

GPU Teaching Kit – Accelerated Computing

# Цели

Цель данной лабораторной – реализовать эффективный алгоритм вычисления гистограммы массива символов ASCII. Имеется 128 ASCII символов и каждый соответствует своему собственному интервалу. Счетчики интервалов представлены 32-битными беззнаковыми целыми без насыщения. Используйте подход, выполняющий приватизацию в разделяемой памяти для каждого блока нитей, а после – автоматически модифицирующий глобальную гистограмму.

# Ход выполнения

Отредактируйте код, чтобы выполнить следующие действия:

* выделить память на устройстве
* скопировать память хоста на устройство
* инициализировать размерности блока нитей и ядра
* запустить ядро CUDA
* скопировать результат на хост
* освободить память устройства

Места, куда следует вставить код, помечены комментариями //@@.

# Инструкция по установке

Последнюю версию исходного кода лабораторной, наряду со скриптами сборки, можно найти в репозитории [Bitbucket](https://bitbucket.org/hwuligans/gputeachingkit-labs/src/master/Module7). Инструкции к Cmake и сборки лабораторной можно найти в файле [README](https://bitbucket.org/hwuligans/gputeachingkit-labs/src/master/README.md) в корневой директории репозитория.

Исполняемый файл, являющийся результатом компиляции лабораторной, можно запустить следующей командой:

./Histogram\_Template -e <expected.raw> \  
 -i <input.txt> -o <output.raw> -t integral\_vector

где <expected.ppm> – имя ожидаемого результата, <input.ppm> – входной набор данных и <output.ppm> – необязательный путь для результатов. Набор данных можно сгенерировать, используя генератор, собранный во время компиляции.

# Вопросы

1. Опишите все оптимизации, которые вы попробовали выполнить, вне зависимости от того, стали ли вы их использовать в дальнейшем или нет, и вне зависимости от того, как они повлияли на производительность.
2. С какими сложностями вы столкнулись во время выполнения оптимизации?
3. Какая оптимизация дала наибольший прирост производительности?
4. Сколько чтений из глобальной памяти выполнит выше ядро вычисления гистограммы? Объясните.
5. Сколько записей в глобальную память выполнит выше ядро вычисления гистограммы? Объясните.
6. Сколько атомарных операций выполнит выше ядро вычисления гистограммы? Объясните.
7. Большинство текстов содержит только буквы, цифры и пробелы. Эти 95 знаков в таблице ASCII имеют последовательные значения кода: 32 - 126. Что можно сказать о ситуации (порядок выполнения) с атомарным доступом, когда более 95 нитей одновременно пытаются инкрементировать одну гистограмму?

# Шаблон кода

Представленный код предлагается как отправная точка. Импорты, экспорты и проверка решения уже представлены в коде. Требуется вставить свой код в области, обозначенные //@@. Остальной код трогать не нужно. Руководство описывает функционал методов с префиксом wb\*.

#include <wb.h>  
  
#define NUM\_BINS 4096  
  
#define CUDA\_CHECK(ans) \  
 { gpuAssert((ans), \_\_FILE\_\_, \_\_LINE\_\_); }  
inline void gpuAssert(cudaError\_t code, const char \*file, int line,  
 bool abort = true) {  
 if (code != cudaSuccess) {  
 fprintf(stderr, "GPUassert: %s %s %d\n", cudaGetErrorString(code),  
 file, line);  
 if (abort)  
 exit(code);  
 }  
}  
  
int main(int argc, char \*argv[]) {  
 wbArg\_t args;  
 int inputLength;  
 unsigned int \*hostInput;  
 unsigned int \*hostBins;  
 unsigned int \*deviceInput;  
 unsigned int \*deviceBins;  
  
 args = wbArg\_read(argc, argv);  
  
 wbTime\_start(Generic, "Importing data and creating memory on host");  
 hostInput = (unsigned int \*)wbImport(wbArg\_getInputFile(args, 0),  
 &inputLength, "Integer");  
 hostBins = (unsigned int \*)malloc(NUM\_BINS \* sizeof(unsigned int));  
 wbTime\_stop(Generic, "Importing data and creating memory on host");  
  
 wbLog(TRACE, "The input length is ", inputLength);  
 wbLog(TRACE, "The number of bins is ", NUM\_BINS);  
  
 wbTime\_start(GPU, "Allocating GPU memory.");  
 //@@ Выделите память GPU  
 CUDA\_CHECK(cudaDeviceSynchronize());  
 wbTime\_stop(GPU, "Allocating GPU memory.");  
  
 wbTime\_start(GPU, "Copying input memory to the GPU.");  
 //@@ Скопируйте память с хоста на GPU  
 CUDA\_CHECK(cudaDeviceSynchronize());  
 wbTime\_stop(GPU, "Copying input memory to the GPU.");  
  
 // Запуск ядра  
 // ----------------------------------------------------------  
 wbLog(TRACE, "Launching kernel");  
 wbTime\_start(Compute, "Performing CUDA computation");  
 //@@ Выполните вычисления в ядре  
 wbTime\_stop(Compute, "Performing CUDA computation");  
  
 wbTime\_start(Copy, "Copying output memory to the CPU");  
 //@@ Скопируйте память обратно с GPU на хост  
 CUDA\_CHECK(cudaDeviceSynchronize());  
 wbTime\_stop(Copy, "Copying output memory to the CPU");  
  
 wbTime\_start(GPU, "Freeing GPU Memory");  
 //@@ Освободите память GPU  
 wbTime\_stop(GPU, "Freeing GPU Memory");  
  
 // Проверка корректности результатов  
 // -----------------------------------------------------  
 wbSolution(args, hostBins, NUM\_BINS);  
  
 free(hostBins);  
 free(hostInput);  
 return 0;  
}