Разностная схема

GPU Teaching Kit – Accelerated Computing

# Цели

Цель данной лабораторной – расчет разностной схемы с 7-точечным шаблоном, с использованием разделяемой памяти.

# Ход работы

* Отредактировать код для реализации разностной схемы с 7-ми точечным шаблоном.
* Отредактировать код для запуска ядра, которое вы написали. Функция должна запускать двумерную сетку и блоки CUDA.
* Ответить на вопросы

# Алгоритм

Вам требуется рассчитать разностную схему с 7-точечным шаблоном. Обрабатывать граничные условия не обязательно. Результат должен попадать в диапазон от 0 до 255.

for i from 1 to height-1: # заметьте: диапазон открытый, без границ   
 for j from 1 to width-1: # это сделано для упрощения  
 for k from 1 to depth-1:# на границах результат равен 0  
 res = in(i, j, k + 1) + in(i, j, k - 1) + in(i, j + 1, k) +  
 in(i, j - 1, k) + in(i + 1, j, k) + in(i - 1, j, k) -  
 6 \* in(i, j, k)  
 out(i, j, k) = Clamp(res, 0, 255)  
 end  
 end  
end

где Clamp определено как

def Clamp(val, start, end):  
 return Max(Min(val, end), start)  
end

А in(i, j, k) и out(i, j, k) являются вспомогательными функциями, определенными как

#define value(arry, i, j, k) arry[(( i )\*width + (j)) \* depth + (k)]  
#define in(i, j, k) value(input\_array, i, j, k)  
#define out(i, j, k) value(output\_array, i, j, k)

# Вопросы

1. Сколько чтений из глобальной памяти выполнит программа?
2. Сколько чтений из разделяемой памяти выполнит ваша программа?
3. Как бы вы использовали готовый код, выполняющий свертку, для расчета данной разностной схемы?
4. Ваш код, рассчитывающий разностную схему, делает столько же, меньше или больше обращений к памяти, чем эквивалентный код расчета свертки 3x3?

# 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 wbCheck(stmt) \  
 do { \  
 cudaError\_t err = stmt; \  
 if (err != cudaSuccess) { \  
 wbLog(ERROR, "Failed to run stmt ", #stmt); \  
 wbLog(ERROR, "Got CUDA error ... ", cudaGetErrorString(err)); \  
 return -1; \  
 } \  
 } while (0)  
  
\_\_global\_\_ void stencil(float \*output, float \*input, int width, int height,  
 int depth) {  
 //@@ добавьте свой код сюда  
}  
  
static void launch\_stencil(float \*deviceOutputData, float \*deviceInputData,  
 int width, int height, int depth) {  
 //@@ добавьте свой код сюда  
}  
  
int main(int argc, char \*argv[]) {  
 wbArg\_t arg;  
 int width;  
 int height;  
 int depth;  
 char \*inputFile;  
 wbImage\_t input;  
 wbImage\_t output;  
 float \*hostInputData;  
 float \*hostOutputData;  
 float \*deviceInputData;  
 float \*deviceOutputData;  
  
 arg = wbArg\_read(argc, argv);  
  
 inputFile = wbArg\_getInputFile(arg, 0);  
  
 input = wbImport(inputFile);  
  
 width = wbImage\_getWidth(input);  
 height = wbImage\_getHeight(input);  
 depth = wbImage\_getChannels(input);  
  
 output = wbImage\_new(width, height, depth);  
  
 hostInputData = wbImage\_getData(input);  
 hostOutputData = wbImage\_getData(output);  
  
 wbTime\_start(GPU, "Doing GPU memory allocation");  
 cudaMalloc((void \*\*)&deviceInputData,  
 width \* height \* depth \* sizeof(float));  
 cudaMalloc((void \*\*)&deviceOutputData,  
 width \* height \* depth \* sizeof(float));  
 wbTime\_stop(GPU, "Doing GPU memory allocation");  
  
 wbTime\_start(Copy, "Copying data to the GPU");  
 cudaMemcpy(deviceInputData, hostInputData,  
 width \* height \* depth \* sizeof(float),  
 cudaMemcpyHostToDevice);  
 wbTime\_stop(Copy, "Copying data to the GPU");  
  
 wbTime\_start(Compute, "Doing the computation on the GPU");  
 launch\_stencil(deviceOutputData, deviceInputData, width, height, depth);  
 wbTime\_stop(Compute, "Doing the computation on the GPU");  
  
 wbTime\_start(Copy, "Copying data from the GPU");  
 cudaMemcpy(hostOutputData, deviceOutputData,  
 width \* height \* depth \* sizeof(float),  
 cudaMemcpyDeviceToHost);  
 wbTime\_stop(Copy, "Copying data from the GPU");  
  
 wbSolution(arg, output);  
  
 cudaFree(deviceInputData);  
 cudaFree(deviceOutputData);  
  
 wbImage\_delete(output);  
 wbImage\_delete(input);  
  
 return 0;  
}