ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение высшего профессионального образования ордена Трудового Красного Знамени

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

МКиИТ

**ЛАБОРАТОРНАЯ РАБОТА №10**

По дисциплине «Распределенные и параллельные вычисления»

На тему **«Свертка»**

Выполнил студент 2 курса группы МБД2032:

Трояновский Семён Владиславович

Москва

2021

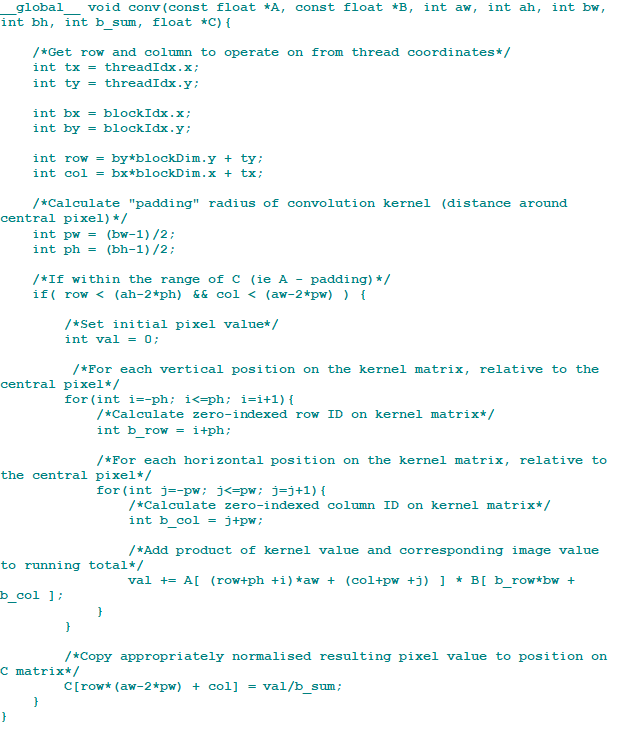
**Задача лабораторной работы:**

Цель данной лабораторной – реализовать блочный алгоритм свертки изображения, используя как разделяемую, так и константную память.

**Ход выполнения:**

1. **Алгоритм расчётов**

Для начала реализуем алгоритм расчётов:

****

1. **Полный код**

Полный код программы в моем случае выглядит так:

import numpy as np

import pycuda.driver as cuda

import pycuda.autoinit

from PIL import Image

from pycuda.compiler import SourceModule

from scipy import signal as sg

from scipy import misc

# DEVICE SETUP

BLOCK\_SIZE = 32 # Max 32. 32\*\*2 = 1024, max for GTX1060

kernel = '''

\_\_global\_\_ void conv(const float \*A, const float \*B, int aw, int ah, int bw, int bh, int b\_sum, float \*C){

/\*Get row and column to operate on from thread coordinates\*/

int tx = threadIdx.x;

int ty = threadIdx.y;

int bx = blockIdx.x;

int by = blockIdx.y;

int row = by\*blockDim.y + ty;

int col = bx\*blockDim.x + tx;

/\*Calculate "padding" radius of convolution kernel (distance around central pixel)\*/

int pw = (bw-1)/2;

int ph = (bh-1)/2;

/\*If within the range of C (ie A - padding)\*/

if( row < (ah-2\*ph) && col < (aw-2\*pw) ) {

/\*Set initial pixel value\*/

int val = 0;

/\*For each vertical position on the kernel matrix, relative to the central pixel\*/

for(int i=-ph; i<=ph; i=i+1){

/\*Calculate zero-indexed row ID on kernel matrix\*/

int b\_row = i+ph;

/\*For each horizontal position on the kernel matrix, relative to the central pixel\*/

for(int j=-pw; j<=pw; j=j+1){

/\*Calculate zero-indexed column ID on kernel matrix\*/

int b\_col = j+pw;

/\*Add product of kernel value and corresponding image value to running total\*/

val += A[ (row+ph +i)\*aw + (col+pw +j) ] \* B[ b\_row\*bw + b\_col ];

}

}

/\*Copy appropriately normalised resulting pixel value to position on C matrix\*/

C[row\*(aw-2\*pw) + col] = val/b\_sum;

}

}

'''

# Compile kernel

mod = SourceModule(kernel)

# Get functions

conv = mod.get\_function("conv")

def convolve(a, b):

global BLOCK\_SIZE

global conv

a, b = [np.array(i).astype(np.float32) for i in [a, b]]

# Matrix A

aw = np.int32(a.shape[1]) # Widthof in matrix

ah = np.int32(a.shape[0]) # Height of in matrix

# Matrix B (kernel)

bw = np.int32(b.shape[1]) # Widthof in matrix

if bw % 2 == 0:

print("Kernel width is not an odd number! Strange things will happen...")

bh = np.int32(b.shape[0]) # Height of in matrix

if bh % 2 == 0:

print("Kernel height is not an odd number! Strange things will happen...")

b\_sum = np.int32(np.absolute(b).sum())

# Matrix C, subtract 2\*padding, \*2 because it's taken off all sides

c = np.empty([ah - (bh - 1), aw - (bw - 1)])

c = c.astype(np.float32)

# Allocate memory on device

a\_gpu = cuda.mem\_alloc(a.nbytes)

b\_gpu = cuda.mem\_alloc(b.nbytes)

c\_gpu = cuda.mem\_alloc(c.nbytes)

# Copy matrix to memory

cuda.memcpy\_htod(a\_gpu, a)

cuda.memcpy\_htod(b\_gpu, b)

# Set grid size from A matrix

grid = (int(aw / BLOCK\_SIZE + (0 if aw % BLOCK\_SIZE is 0 else 1)),

int(ah / BLOCK\_SIZE + (0 if ah % BLOCK\_SIZE is 0 else 1)),

1)

# Call gpu function

conv(a\_gpu, b\_gpu, aw, ah, bw, bh, b\_sum, c\_gpu, block=(BLOCK\_SIZE, BLOCK\_SIZE, 1), grid=grid)

# Copy back the result

cuda.memcpy\_dtoh(c, c\_gpu)

# Free memory. May not be useful? Ask about this.

a\_gpu.free()

b\_gpu.free()

c\_gpu.free()

# Return the result

return c

k\_sv = [[-1., 0., 1.], [-2., 0., 2.], [-1., 0., 1.]]

a = Image.open('.\\resources\\blur\\to\_blur.jpg')

px = np.array(a)

px = px.astype(np.float32)

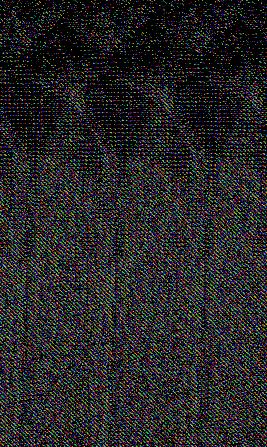
c = convolve(px, k\_sv)

pil\_im = Image.fromarray(c, mode="RGB")

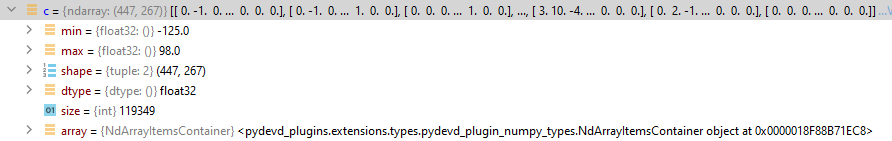
pil\_im.save('.\\resources\\convol\\convol.jpg')

1. **Результат**

Полученное изображение выглядит так:

****

И является такой матрицей:



1. **Ответы на вопросы**

(1) Назовите 3 применения свертки.

*Обработка изображений, обработка сигналов, разностные схемы.*

(2) Сколько операций с плавающей точкой будет выполнено вашим ядром свертки? Объясните.

*Для каждого выходного элемента требуется MASK\_WIDTH \* MASK\_WIDTH входных элементов, а всего выходных элементов width \* height \* channels.*

(3) Сколько чтений из глобальной памяти произведет ваше ядро? Объясните.

*Каждый блок нитей считает (TILE\_WIDTH+MASK\_WIDTH-1)^2 элементов. Всего width/TILE\_WIDTH\*height/TILE\_WIDTH блоков нитей,*

*что в сумме дает (TILE\_WIDTH + MASK\_WIDTH - 1)^2 \* height \* width/TILE\_WIDTH^2.*

*Это включает фантомные элементы, которые не будут считаны.*

*Нужно вычесть ((height + MASK\_WIDTH/2)(width + MASK\_WIDTH/2) - height \* width).*

(4) Сколько записей в глобальную память произведет ваше ядро? Объясните.

*Одна запись на один выходной элемент, или width \* height \* channels.*

(5) Какими будут максимальное, минимальное и среднее значения количества реальных операций, которые произведет одна нить?

Под реальными подразумеваются операции, которые напрямую влияют на финальный результат.

*Все потоки выполнят одну и ту же работу во время свертки, которая исчисляется KERNEL\_WIDTH \* KERNEL\_WIDTH операциями.*

(6) Какой была бы скорость выполнения операций с плавающей точкой для ядер CPU и GPU в данном приложении?

Как они зависят от размера входных данных?

*Это зависит от конкретной машины. Производительность реализации на CPU будет падать с ростом размера входных данных,*

*так как они станут слишком большими для кэша. Реализация на GPU будет иметь относительно постоянную скорость выполнения*

*операций с плавающей точкой.*

(7) Сколько дополнительного времени понадобится в качестве накладных расходов на вычисления на GPU? Учтите,

весь код выполняется на хосте, за исключением самого ядра. Как будут изменяться эти накладные расходы при увеличении

размера входных данных?

*Ответ зависит от конкретной машины. Количество данных, которые нужно переслать, и выполнение свертки растут*

*с одинаковым темпом, поэтому для больших входных данных эти показатели тоже будут расти с одинаковым темпом.*

(8) Как вы думаете, что случится, если вы увеличите размер маски (скажем, до 1024), в то время как размер блока будет 16x16?

На что вы потратите больше всего времени? Наложит ли это новые ограничения, которые нужно будет учесть в алгоритме

(подумайте о размере разделяемой/константной памяти)?

*Если маска свертки слишком большая, потребуется слишком много места для обеспечения параллелизма в блоке нитей.*

*Для больших масок большую часть времени будет производиться чтение фиктивных граничных элементов из глобальной памяти*

*в разделяемую, что нивелирует все преимущества алгоритма.*

(9) Обязательно ли вам иметь отдельный буфер для результата? Или, с другой стороны, почему нельзя записывать результат

свертки в то же самое место?

*Если вести запись результата в то же самое место, можно изменить значение, необходимое другой нити, что может*

*привести к некорректному результату.*

(10) Что такое тождественная маска?

*Все нули, кроме единицы в центре.*

1. **Вывод**

В данной работе я реализовал свертку изображения с использованием CUDA API.