%%writefile cudabasic.cu

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

double time\_h = 0;

double time\_d = 0;

int numOfRounds = 1;

void meanFilter\_h (unsigned char\* raw\_image, unsigned char\* filtered\_image, int img\_width, int img\_height, int window\_size)

{

int half\_window = (window\_size - 1) / 2;

for (int i=0; i < img\_height; i++)

{

for(int j=0; j < img\_width; j++)

{

int left\_limit, right\_limit, top\_limit, bottom\_limit;

if(j - half\_window >= 0){

left\_limit = j-half\_window;

}else{

left\_limit = 0;

}

if(j + half\_window <= img\_width-1){

right\_limit = j + half\_window;

}else{

right\_limit = img\_width-1;

}

if(i - half\_window >= 0){

top\_limit = i - half\_window;

}else{

top\_limit = 0;

}

if(i + half\_window <= img\_height-1){

bottom\_limit = i + half\_window;

}else{

bottom\_limit = img\_height-1;

}

double sum\_r = 0, sum\_g = 0, sum\_b = 0;

for(int k = top\_limit; k <= bottom\_limit; k++)

{

for(int m = left\_limit; m <= right\_limit; m++)

{

int index = (k \* img\_width + m) \* 3;

// Accumulate the values of each color channel separately

sum\_r += raw\_image[index];

sum\_g += raw\_image[index + 1];

sum\_b += raw\_image[index + 2];

}

}

int current\_window\_size = (bottom\_limit - top\_limit + 1) \* (right\_limit - left\_limit + 1);

// Calculate the mean value for each color channel

filtered\_image[(i \* img\_width + j) \* 3] = sum\_r / current\_window\_size;

filtered\_image[(i \* img\_width + j) \* 3 + 1] = sum\_g / current\_window\_size;

filtered\_image[(i \* img\_width + j) \* 3 + 2] = sum\_b / current\_window\_size;

}

}

}

\_\_global\_\_ void meanFilter\_d (unsigned char\* raw\_image, unsigned char\* filtered\_image, int img\_width, int img\_height, int window\_size)

{

int j = blockIdx.x \* blockDim.x + threadIdx.x;

int i = blockIdx.y \* blockDim.y + threadIdx.y;

int half\_window = (window\_size - 1) / 2;

if (i < img\_height && j < img\_width)

{

int left\_limit, right\_limit, top\_limit, bottom\_limit;

if(j - half\_window >= 0){

left\_limit = j-half\_window;

}else{

left\_limit = 0;

}

if(j + half\_window <= img\_width-1){

right\_limit = j + half\_window;

}else{

right\_limit = img\_width-1;

}

if(i - half\_window >= 0){

top\_limit = i - half\_window;

}else{

top\_limit = 0;

}

if(i + half\_window <= img\_height-1){

bottom\_limit = i + half\_window;

}else{

bottom\_limit = img\_height-1;

}

double sumR = 0, sumG = 0, sumB = 0;

int current\_window\_size = 0;

for(int k = top\_limit; k <= bottom\_limit; k++)

{

for(int m = left\_limit; m <= right\_limit; m++)

{

// Calculate the indices for Red, Green, and Blue components

int index = 3 \* (k \* img\_width + m);

sumR += raw\_image[index]; // Red component

sumG += raw\_image[index + 1]; // Green component

sumB += raw\_image[index + 2]; // Blue component

current\_window\_size++;

}

}

// Calculate the average values for Red, Green, and Blue components

unsigned char avgR = sumR / current\_window\_size;

unsigned char avgG = sumG / current\_window\_size;

unsigned char avgB = sumB / current\_window\_size;

// Write the average values to the filtered image

int output\_index = 3 \* (i \* img\_width + j);

filtered\_image[output\_index] = avgR; // Red component

filtered\_image[output\_index + 1] = avgG; // Green component

filtered\_image[output\_index + 2] = avgB; // Blue component

}

}

void saveBitmap(const char\* filename, int width, int height, unsigned char\* imageData,unsigned char \*info) {

FILE\* file = fopen(filename, "wb");

if (file == NULL) {

fprintf(stderr, "Error: Unable to open file %s for writing.\n", filename);

exit(1);

}

printf("File %s opened successfully for writing.\n", filename);

// Calculate padding bytes

int padding = (4 - (width \* 3) % 4) % 4;

unsigned char fileHeader[138];

memcpy(fileHeader, info, 138);

// Write the file header

fwrite(fileHeader, sizeof(unsigned char), 138, file);

// Allocate memory for a single row including padding

unsigned char\* paddedRow = (unsigned char\*)malloc(sizeof(unsigned char) \* (width \* 3 + padding));

if (paddedRow == NULL) {

fprintf(stderr, "Error: Unable to allocate memory for padded row.\n");

fclose(file);

exit(1);

}

fwrite(imageData, sizeof(unsigned char), width \* height \* 3, file);

// Write padding bytes

//unsigned char paddingData[4] = {0, 0, 0, 0};

//for (int i = 0; i < padding; i++) {

// fwrite(paddingData, sizeof(unsigned char), 1, file);

// }

if (ferror(file)) {

fprintf(stderr, "Error writing to file.\n");

fclose(file);

free(paddedRow);

exit(1);

}

// Free allocated memory

// Close the file

fclose(file);

}

int main(int argc,char \*\*argv)

{

printf("Begin......\n");

//get bitmap to a char array

FILE\* file = fopen("/content/drive/MyDrive/img\_640.bmp", "rb");

unsigned char info[138];

fread(info, sizeof(unsigned char), 138, file);

int width, height;

memcpy(&width, info + 18, sizeof(int));

memcpy(&height, info + 22, sizeof(int));

int window\_size = 3;

int size = 3 \* width \* abs(height);

unsigned char \*inputImage = (unsigned char\*)malloc(size \* sizeof(unsigned char));

unsigned char\* result\_image\_data\_d;

unsigned char \*result\_image\_data\_h=(unsigned char\*)malloc(size \* sizeof(unsigned char));

unsigned char \*result\_image\_data\_h1=(unsigned char\*)malloc(size \* sizeof(unsigned char));

unsigned char\* image\_data\_d;

fread(inputImage, sizeof(unsigned char), size, file);

fclose(file);

int block\_size = 32;

int grid\_size = width/block\_size;

dim3 dimBlock(block\_size, block\_size, 1);

dim3 dimGrid(grid\_size, grid\_size, 1);

for(int x = 0; x < numOfRounds; x += 1)

{

cudaMalloc((void \*\*)&image\_data\_d,size\*sizeof(unsigned char));

cudaMalloc((void \*\*)&result\_image\_data\_d,size\*sizeof(unsigned char));

cudaMemcpy(image\_data\_d,inputImage,size\*sizeof(unsigned char),cudaMemcpyHostToDevice);

clock\_t start\_d=clock();

//execution of GPU code

meanFilter\_d <<< dimGrid, dimBlock >>> (image\_data\_d, result\_image\_data\_d, width, height, window\_size);

cudaDeviceSynchronize();

cudaError\_t error = cudaGetLastError();

if(error!=cudaSuccess)

{

fprintf(stderr,"ERROR: %s\n", cudaGetErrorString(error) );

exit(-1);

}

clock\_t end\_d = clock();

cudaMemcpy(result\_image\_data\_h, result\_image\_data\_d, size \* sizeof(unsigned char), cudaMemcpyDeviceToHost);

saveBitmap("image1.bmp", width, height, result\_image\_data\_h,info);

clock\_t start\_h = clock();

//executing CPU code

meanFilter\_h(inputImage, result\_image\_data\_h1, width, height, window\_size);

clock\_t end\_h = clock();

time\_h += (double)(end\_h-start\_h)/CLOCKS\_PER\_SEC;

time\_d += (double)(end\_d-start\_d)/CLOCKS\_PER\_SEC;

cudaFree(image\_data\_d);

cudaFree(result\_image\_data\_d);

}

printf("Average GPU execution time: %f\n",(time\_d/numOfRounds));

printf("Average CPU execution time: %f\n",(time\_h/numOfRounds));

printf("CPU/GPU time: %f\n",(time\_h/time\_d));

return 0;

}

**Output**

Begin......

File image1.bmp opened successfully for writing.

Average GPU execution time: 0.001159

Average CPU execution time: 0.041736

CPU/GPU time: 36.010354



Original Image



Blurred Image