

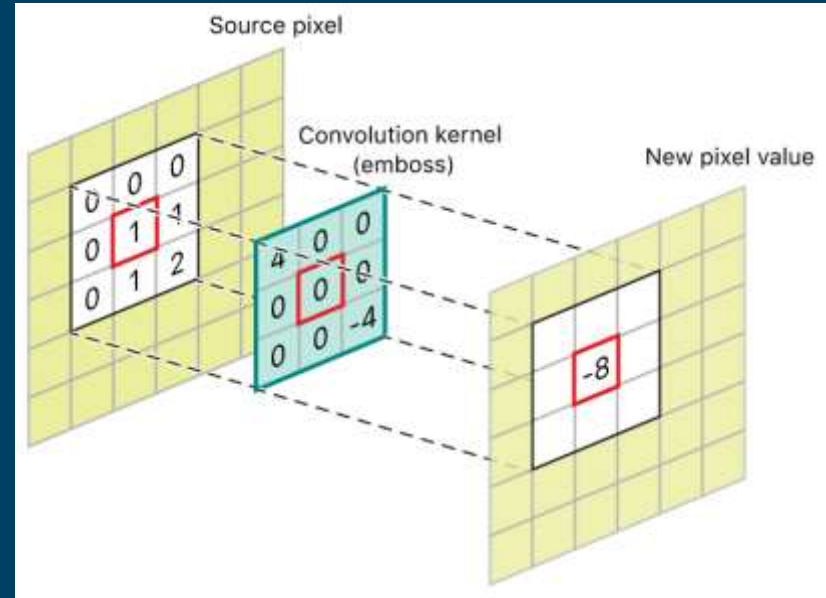
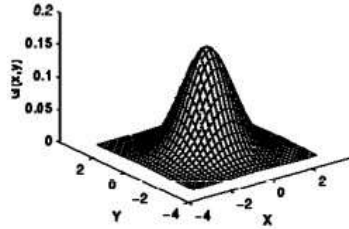
Image Processing Using CUDA

A Presentation by Leslie TIENTCHEU

Introduction

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

A graphical representation of the 2D Gaussian distribution with mean(0,0) and $\sigma = 1$ is shown to the right.



GaussianBlur

Base Sequential Code

```
int main()
{
    // Image path
    std::string imagePath = "content/sampleImage.jpg"; // Replace with your actual image path
    cv::Mat image = cv::imread(imagePath, cv::IMREAD_GRAYSCALE);
    if (image.empty())
    {
        std::cerr << "OpenCV version: " << CV_VERSION << std::endl;
        std::cerr << "Image load failed" << std::endl;
        return -1;
    }

    cv::Mat blurredImageCPU(image.size(), image.type());
    cv::Mat blurredImageGPU(image.size(), image.type());

    // [1] Gaussian Blur (for timing comparison)
    auto startCPU = std::chrono::high_resolution_clock::now();
    cv::GaussianBlur(image, blurredImageCPU, cv::Size(3, 3), 3.0);
    auto endCPU = std::chrono::high_resolution_clock::now();

    // Allocate device memory
    unsigned char *d_input, *d_output;
    cudaMalloc(&d_input, image.total());
    cudaMalloc(&d_output, image.total());

    // Copy input image to device
    cudaMemcpy(&d_input, image.data, image.total(), cudaMemcpyHostToDevice);

    // Device launch configuration
    dim3 blockDim(512, 512, 1); // 32 blocs
    dim3 gridSize((image.cols + blockDim.x - 1) / blockDim.x,
                  (image.rows + blockDim.y - 1) / blockDim.y); // 32 grid

    // GPU Gaussian Blur
    auto startGPU = std::chrono::high_resolution_clock::now();
    gaussianBlurGPU-->gridDim, blockDim-->d_input, d_output, image.rows, 1.0);
    cudaMemcpy(&d_output, image.data, image.total(), cudaMemcpyDeviceToHost);
    auto endGPU = std::chrono::high_resolution_clock::now();

    // Copy result back to host [2]
    cv::Mat blurredImageHost(image.size(), image.type());
    cudaMemcpy(blurredImageHost.data, d_output, image.total(), cudaMemcpyDeviceToHost);

    // Calculate execution times
    auto gpuDuration = std::chrono::duration<double>, std::milli-->endGPU - startGPU.count();
    auto cpuDuration = std::chrono::duration<double>, std::milli-->endCPU - startCPU.count();

    std::cout << "[1] Time: " << gpuDuration << " ms" << std::endl;
    std::cout << "CPU Time: " << cpuDuration << " ms" << std::endl;

    // Save only blurred image
    cv::imwrite("content/gpu_blurred_image.jpg", blurredImageHost);
    cv::imwrite("content/gpu_blurred_image.jpg", blurredImageGPU);

    // Display confirmation message: ScreenShot
    std::cout << "Blurred image saved as: 1-1" << std::endl;
}
```

Snippet of base sequential

```
// CPU Gaussian Blur (for timing comparison)
auto startCPU = std::chrono::high_resolution_clock::now();
cv::GaussianBlur(image, blurredImageGPU, cv::Size(3, 3), 3.0);
auto endCPU = std::chrono::high_resolution_clock::now();
```

CUDA Kernel - gaussian Blur CUDA

Snippet of CUDA Kernel

```
int main() {
    // Image Path
    std::string imagePath = "/content/sampleImage.jpeg"; // Replace with your uploaded image path
    cv::Mat image = cv::imread(imagePath, cv::IMREAD_GRAYSCALE);
    if (image.empty()) {
        std::cerr << "OpenCV version: " << CV_VERSION << std::endl;
        std::cerr << "Image load failed!" << std::endl;
        return -1;
    }

    cv::Mat blurredImageGPU(image.size(), image.type());

    // Allocate device memory
    unsigned char *d_input, *d_output;
    cudaError_t err = cudaMalloc(&d_input, image.total());
    if (err != cudaSuccess) {
        std::cerr << "Failed to allocate device memory - " << cudaGetErrorString(err);
        return -1;
    }
    err = cudaMalloc(&d_output, image.total());
    if (err != cudaSuccess) {
        std::cerr << "Failed to allocate device memory - " << cudaGetErrorString(err);
        return -1;
    }

    // Copy input image to device
    err = cudaMemcpy(d_input, image.data, image.total(), cudaMemcpyHostToDevice);
    if (err != cudaSuccess) {
        std::cerr << "Failed to copy data from host to device - " << cudaGetErrorString(err);
        return -1;
    }

    // Kernel launch configuration
    dim3 blockSize(16, 16); // 16x16 block
    dim3 gridSize((image.cols + blockSize.x - 1) / blockSize.x,
                  (image.rows + blockSize.y - 1) / blockSize.y); // 30x30 grid

    // CUDA Gaussian Blur
    auto startGPU = std::chrono::high_resolution_clock::now();
    gaussianBlurGPU(gridSize, blockSize, d_input, d_output, image.cols, image.rows, 0.5f);
    cudaDeviceSynchronize();
    auto endGPU = std::chrono::high_resolution_clock::now();

    // Calculate execution time
    auto gpuDuration = std::chrono::duration_cast<std::chrono::seconds>(endGPU - startGPU).count();
    std::cout << "GPU Time: " << gpuDuration << " s" << std::endl;

    // Copy output image back to host
    cudaMemcpy(blurredImageGPU.data, d_output, image.total(), cudaMemcpyDeviceToHost);

    // Get the extension of the input image
    boost::filesystem::path p(imagePath);
    std::string extension = imagePath.substr(imagePath.find_last_of("."));
}
```

```
// CUDA kernel for Gaussian blur
__global__ void gaussianBlurCUDA(const unsigned char* input, unsigned char* output,
                                int width, int height, float sigma) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

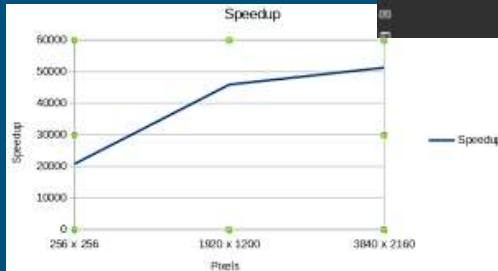
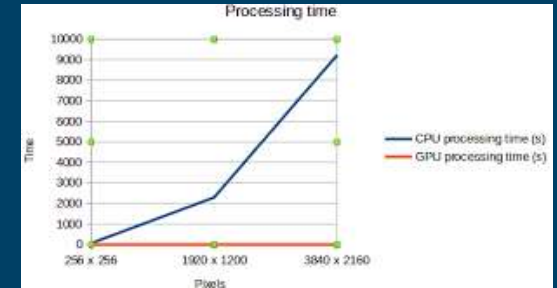
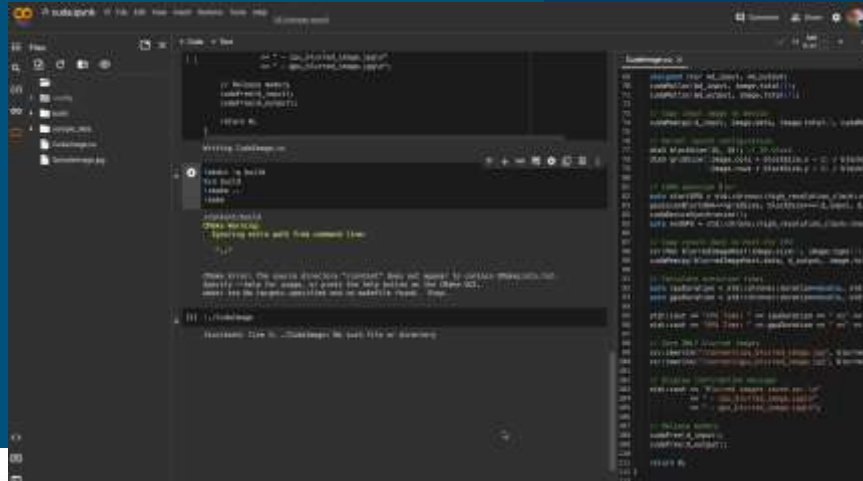
    if (x < width && y < height) {
        float sum = 0.0f;
        float totalWeight = 0.0f;

        // Sample 3x3 neighborhood for simplicity... you can increase the kernel size
        for (int dy = -1; dy <= 1; dy++) {
            for (int dx = -1; dx <= 1; dx++) {
                int pixelX = x + dx;
                int pixelY = y + dy;

                if (pixelX >= 0 && pixelX < width && pixelY >= 0 && pixelY < height) {
                    float weight = gaussianWeight(dx, dy, sigma);
                    sum += input[pixelY * width + pixelX] * weight;
                    totalWeight += weight;
                }
            }
        }

        // Normalize and cast to unsigned char before assigning to output
        output[y * width + x] = (unsigned char)((sum / totalWeight) + 0.5f); // Add 0.5 for rounding
    }
}
```

Performance Comparison & Execution Times



Results - Images

Real-image



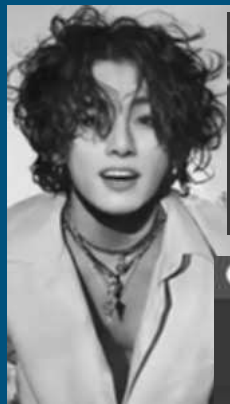
BlurredImage

```
!./CudaImage  
CPU Time: 25.2582 ms  
GPU Time: 1.7666 ms  
Blurred images saved as:  
- cpu_blurred_image.png  
- gpu_blurred_image.png
```

Real-image



Real-image



```
!./CudaImage  
CPU Time: 29.9277 ms  
Blurred image saved as:  
- cpu_blurred_image.jpeg
```

```
!./CudaImage  
GPU Time: 0.485882 ms  
Blurred image saved as:  
- gpu_blurred_image.jpeg
```

```
!./CudaImage  
CPU Time: 5.02373 ms  
Blurred image saved as:  
- cpu_blurred_image.jpg
```

```
!./CudaImage  
GPU Time: 1.49642 ms  
Blurred image saved as:  
- gpu_blurred_image.jpg
```



BlurredImage

BlurredImage

Enhancements for Better Performance

As we work on making our GPU-accelerated blur even better, let's focus on five main areas that can bring significant improvements:

Make Shared Memory Work Smarter:

- Improve overall speed by optimizing how we use shared memory. Smart caching of frequently accessed data in shared memory can speed things up by reducing delays in memory access.

Get the Kernel Just Right:

- Boost performance by finding the best setup for threads and blocks in the Gaussian blur CUDA kernel. A well-optimized kernel can really make the GPU work more efficiently.

Adapt to Image Details:

- Adjust the blur kernel size dynamically based on different parts of the image. This adaptive approach ensures we blur things just right, taking into account various details in the image.

Smooth Data Movement:

- Make the most of the GPU by using asynchronous data transfer between the CPU and GPU. This helps us do multiple things at once, making better use of the GPU's abilities.

Use Profiling Tools Wisely:

- Take advantage of CUDA profiling tools to understand where things can be faster. Profiling helps us find specific areas that need improvement, ensuring we focus on what really matters.

Thank You!!