Cole Bardin
ECEC-413
2/27/25

Image Blur Report

Introduction

A box blur is a linear filter in the spatial domain in which each pixel in the resulting image has a value equal to the average value of its neighboring pixels in the input image. While this algorithm is simple to implement and can run relatively quickly, large images processed on the CPU can leave performance on the table. Since every pixel calculation is independent of one another, they can be performed by threads in parallel. However, large images can have tens of millions of pixels. The GPU is a specialized piece of hardware for performing such tasks on large amounts of data.

The box blur algorithm can be executed faster when performed on the GPU when compared to the CPU.

Approach

To accomplish this algorithm on a GPU, the image pixels must be divided into a grid of thread blocks. Each thread block is a 32x32 array of threads. Each thread will get a single pixel allocated to it. The GPU is responsible for processing thread blocks and scheduling threads. The number of thread blocks is determined by the size of the image. Below is the pseudo code to setup and run the box blur algorithm with CUDA code.

```
image_size = image_width * image_width // square images
gpu_input_image = cudaMalloc(image_size)
gpu_output_image = cudaMalloc(image_size)
cudaMemcpy(gpu_input_image, input_image, image_size)

thread_block_size = 32
dim3 threads(thread_block_size, thread_block_size)
grid_size = (image_width + 32 - 1) / 32
dim3 grid(grid_size, grid_size)

blur_filter_kernel<<< grid, threads >>>(gpu_input_image, gpu_output_image,
image_width)
cudaDeviceSynchronize()

cudaMemcpy(output_image, gpu_output_image, image_size)
cudaFree(gpu_input_image)
cudaFree(gpu_output_image)
```

This code divides the image into a grid of 32x32 thread blocks. It allocates memory in the GPU for the input and output image. It copies the input image to GPU global memory. Then it

instructs the GPU to call the kernel for all of the thread blocks. Once all threads execute, it copies the output image from GPU memory to CPU memory.

The CUDA code for the blur filter kernel can be seen below:

```
__global__ void blur_filter_kernel (const float *in, float *out, int size)
{
	int threadX = threadIdx.x;
	int threadY = threadIdx.y;
	int blockX = blockIdx.x;
	int blockY = blockIdx.y;
	int col = blockDim.x * blockX + threadX;
	int row = blockDim.y * blockY + threadY;

	int i, j;
	int curr_row, curr_col;
	float blur_value;
	int num_neighbors;

	if((row < size) && (col < size)){
	/* Apply blur filter to current pixel */
	blur_value = 0.0;
	num_neighbors = 0;
	for (i = -BLUR_SIZE; i < (BLUR_SIZE + 1); i++) {
		for (j = -BLUR_SIZE; j < (BLUR_SIZE + 1); j++) {
			/* Accumulate values of neighbors while checking for
			* boundary conditions */
			curr_row = row + i;
			curr_col = col + j;
			if ((curr_row > -1) && (curr_row < size) &&\
				(curr_col > -1) && (curr_col < size)) {
			blur_value += in[curr_row * size + curr_col];
			num_neighbors += 1;
			}
		}
	}

	/* Write averaged blurred value out */
	out[row * size + col] = blur_value / num_neighbors;
	}
	return;
}
```

The kernel is the same as the CPU code except for how the row and column values are calculated. In the CPU version, the pixel value is just iterated over. But in the GPU version, CUDA intrinsics are used to map threads to unique pixels.

Results

To test the performance benefits of the CUDA implementation, the algorithm was run on CPU and GPU for image sizes 512x512, 1024x1024, 2048x2048, 4096x4096, 8192x8192, and 16,384x16,384. Execution times were recorded and compared.

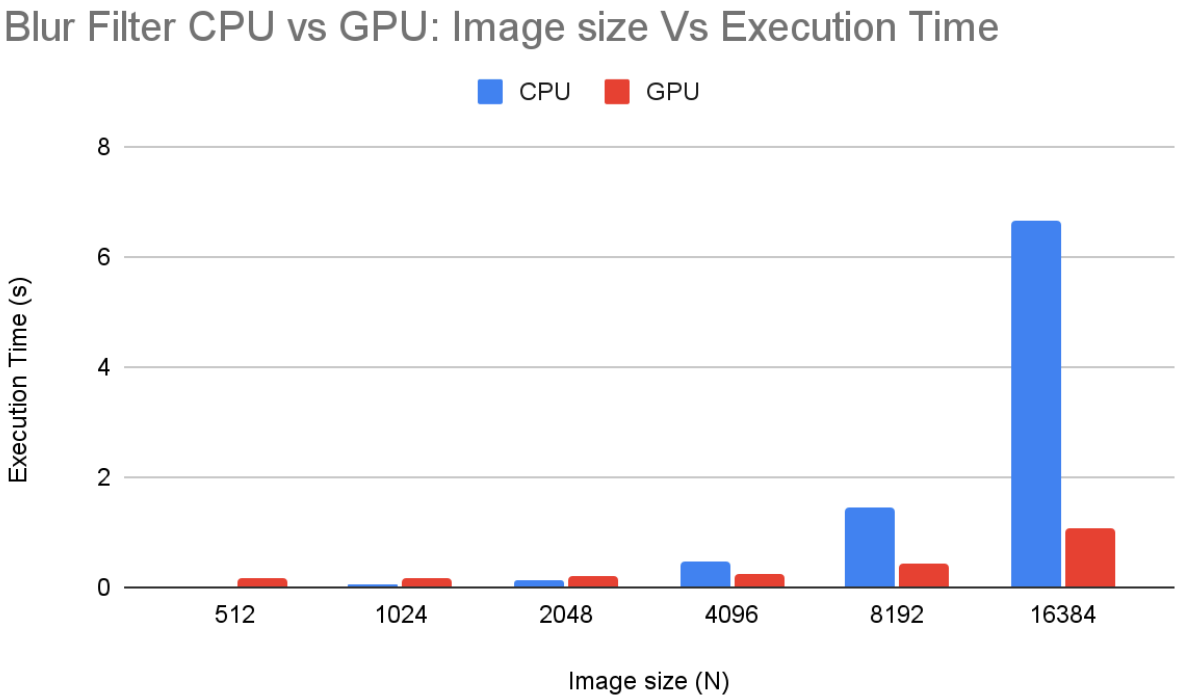

Figure 1. Execution times for GPU and CPU for varying image sizes

For smaller images, the CPU execution time was faster than the respective GPU implementation. However, as the image size increased, the CPU execution time drastically increased, surpassing the GPU times.

Conclusion

It can be seen that utilizing the GPU for image processing like the box blur filter can significantly improve the execution time of the algorithm. It is important to note the overhead of utilizing the GPU may increase execution time for small images. However, as the size of the image increases, so do the performance benefits. Therefore, it is critical to evaluate the problem to determine if the GPU overhead is minimal compared to the performance differences of the GPU and GPU. For most applications of similar algorithms, the GPU will provide large performance benefits.