Cole Bardin
ECEC-413
3/7/25

Integration Estimation Report

Introduction

Evaluating the definite integral of a function can be a very compute heavy task. Increasing the resolution of the estimation can quickly increase the amount of work. This report will cover the benefits of computing integral estimations on the GPU compared to the CPU. The integral estimation algorithm used is the trapezoid method. The integral can be estimated by dividing the bounds of the integral into *n* subintervals and approximating the area over each subinterval by the area of a trapezoid. Since each trapezoid calculation is independent from each other, this is an ideal problem to be solved with parallelism.

Approach

To run this on the GPU with CUDA code, the range [a, b] can be divided into *n* subintervals. 1-dimensional thread blocks can be allocated and arranged in a 1-dimensional execution grid.

Each thread block will calculate the sum of its respective trapezoids into a shared memory array. Then, the threads perform a reduction on the array and atomically add it to the globally shared result variable. The kernel code can be seen below:

```
__global__ void trap_kernel(float a, float b, int n, float h, double *ret)
{
	__shared__ double int_per_thread[TB_SZ];
	size_t tid = blockIdx.x * blockDim.x + threadIdx.x;
	size_t stride = blockDim.x * gridDim.x;
	double sum = 0.0f;
	unsigned int i = tid;
	float x;

	// Seperate statement for first and last value
	// using ternary operaton in while loop below double execution time
	if(i == 0 || i == n - 1)
	{
	x = a + h * i;
	sum += fd(x) / 2.0f;
	i += stride;
	}
	while(i < n)
	{
	x = a + h * i;
	sum += fd(x);
	i += stride;
	}
```

```
        sum *= h;
        int_per_thread[threadIdx.x] = sum;
        __syncthreads();

        i = TB_SZ / 2;
        while(i > 0)
        {
        if(threadIdx.x < i)
        {
              int_per_thread[threadIdx.x] += int_per_thread[threadIdx.x + i];
        }
        __syncthreads();
        i /= 2;
        }

        if(threadIdx.x == 0) atomicAdd(ret, int_per_thread[0]);
}
```

On the current testing platform, the maximum 1-dimensional thread block size is 1024. However, this value can be adjusted to see how the thread block size affects execution time as well.

Results

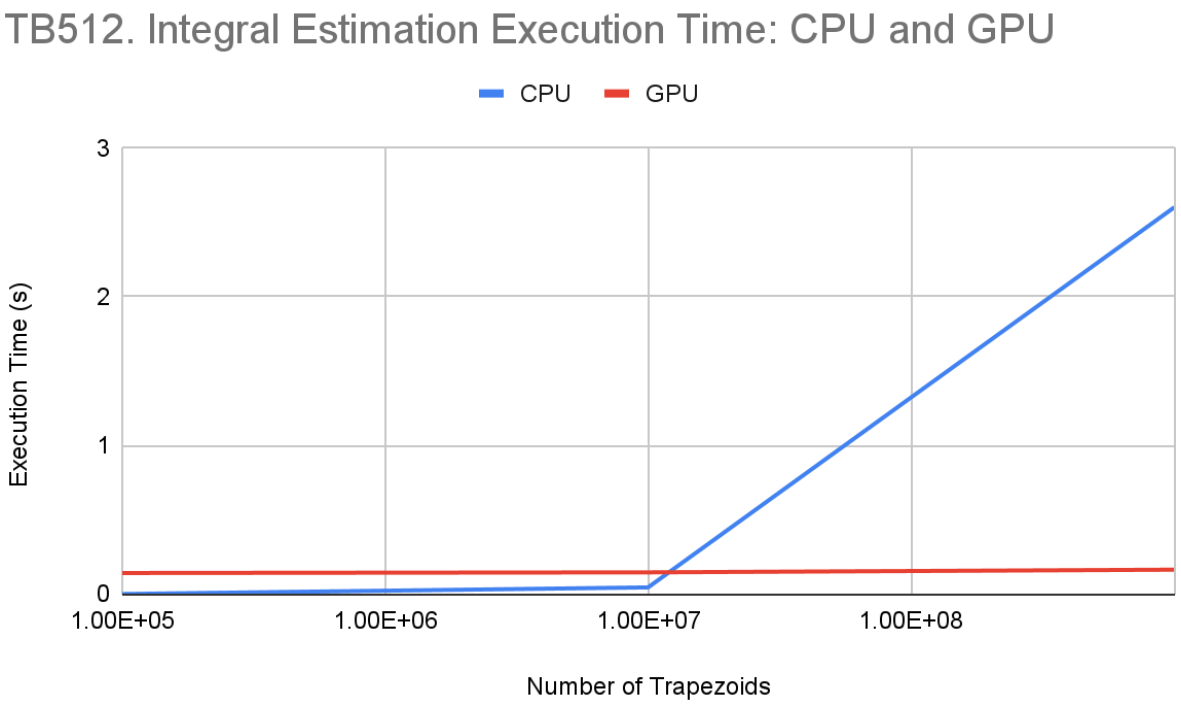

Figure 1. Execution times for CPU and GPU for various resolutions

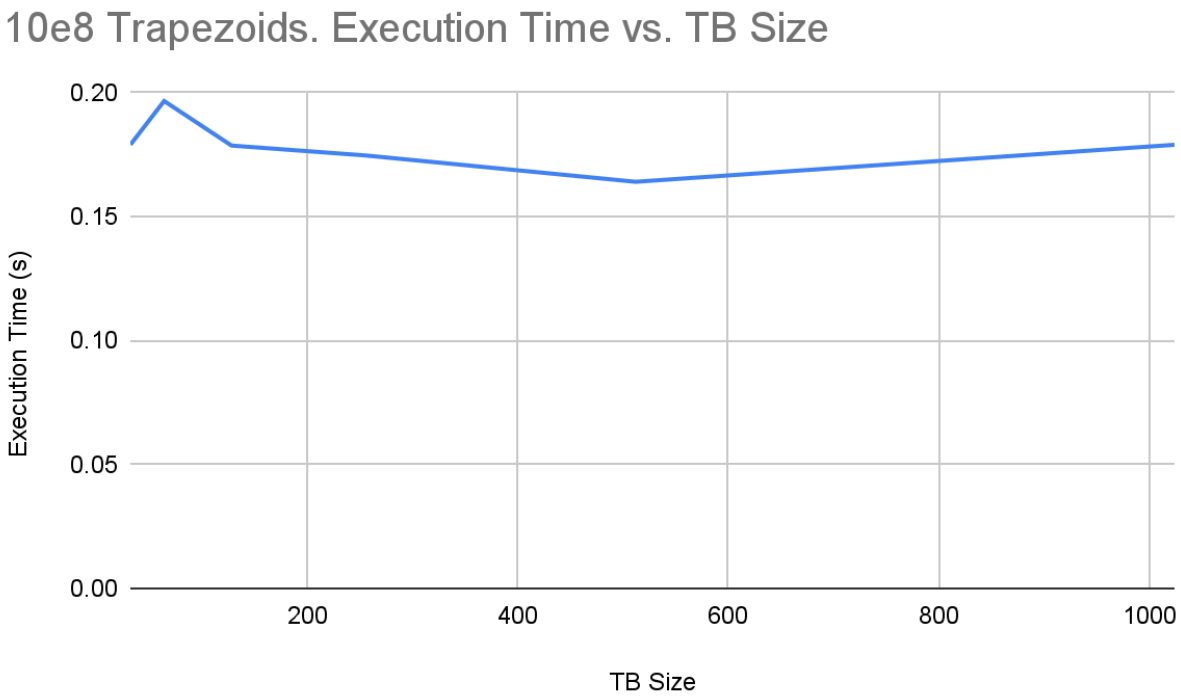

Figure 2. Execution times for GPU calculations as thread block size varies

Conclusion

It can be seen that there is major improvement available from using the GPU to perform integral estimations. As the number of trapezoids increases, i.e. the resolution of the estimate, the performance benefits are achieved. The project description says to ignore the overhead of GPU loading, however I think this is a vital part of the conclusion for this report. Using the GPU does not always provide performance benefits, but there is a point at which it does. So evaluating any problem, an informed decision has to be made about whether to implement GPU operations. And it is important to include the overhead when making that decision.

It can also be seen that the thread block size can have an impact on the performance of the algorithm. Factors like these are all application specific and should be evaluated for each individual case.