Numerical Integration

Prof. Naga Kandasamy
ECE Department, Drexel University

This assignment, worth ten points, is due March 6, 2025, by 11:59 pm. You can work on it in a group of up to two people. Please submit original code.

Given a function $f(x)$ and end points a and b, where $a < b$, we wish to estimate the area under this curve; that is, we wish to determine $\int_a^b f(x)\,\mathrm{d}x$.

The area between the graph of $f(x)$, the vertical lines $x = a$ and $x = b$, and the x-axis can be estimated as shown in Fig. 1 (b) by dividing the interval $[a, b]$ into n subintervals and approximating the area over each subinterval by the area of a trapezoid. Fig. 1(c) shows one such trapezoid where the base of the trapezoid is the subinterval, its vertical sides are the vertical lines through the endpoints of the subinterval, and the fourth side is the secant line joining the points where the vertical lines cross the graph. If the endpoints of the subinterval are x_i and x_{i+1}, then the length of the subinterval is $h = x_{i+1} - x_i$, and if the lengths of the two vertical segments are $f(x_i)$ and $f(x_{i+1})$, then the area of a single trapezoid is $\frac{h}{2}[f(x_i) + f(x_{i+1})]$. If each subinterval has the same

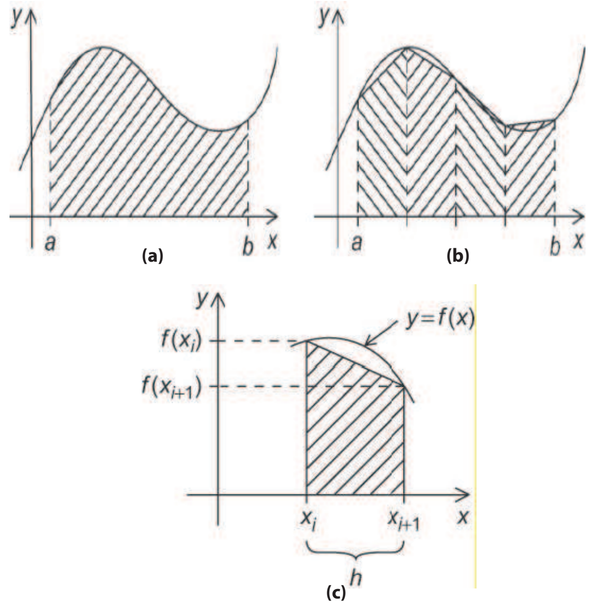

Figure 1: Illustration of the trapezoidal rule: (a) area to be estimated; (b) approximate area using trapezoids; and (c) area under one trapezoid.

length then $h = (b-a)/n$. Also, if we call the leftmost endpoint x_0 and the rightmost endpoint x_n, we have

$$x_0 = a, x_1 = a+h, x_2 = a+2h, \ldots, x_{n-1} = a+(n-1)h, x_n = b,$$

and our approximation of the total area under the curve will be

$$\int_a^b f(x)\,\mathrm{d}x = h[f(x_0)/2 + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n)/2].$$

Thus, the code for a serial algorithm looks like the following.

```
h = (b - a)/n;
integral = (f(a) + f(b))/2.0; /* f is function of interest. */

for (i = 1; i <= n - 1; i++){
        x = a + i * h;
        integral += f(x);
}

integral = h * integral;
```

The program given to you accepts three command-line arguments: the lower and upper limits of integration, a and b, respectively, and the number of trapezoids n to use to approximate the integral. The function $f(x)$ is defined within the file `trap_gold.cpp` as

$$f(x) = \sqrt{\frac{1+x^2}{1+x^4}}.$$

A CPU implementation generates a reference solution which will be compared with your GPU program's output. Edit the *compute_on_device()* function within the file *trap.cu* to complete the functionality of the trapezoidal rule on the GPU.

Implementation hint: Develop your kernel using the design pattern discussed in class for reducing an arbitrarily large array. The source code is available on BBLearn (*vector_reduction_large*).

Create an execution grid comprising of k 1D thread blocks and launch the kernel.

1. Each thread in the execution grid strides over the trapezoids and calculates a partial sum, which it stores in shared memory allocated for the thread block.

2. A barrier synchronization forces threads within the thread block to wait until shared memory is populated with the partial sums calculated from the previous step.

3. Each thread block reduces the values stored in shared memory to a single value using the tree-style reduction technique (use the one that exhibits minimal branch divergence).

4. A barrier synchronization forces threads within the thread block to wait until the reduced value is calculated. Then, designate a single thread within each thread block to accumulate this value into a shared variable that lives in GPU global memory. Use a critical section or atomic operation supported by CUDA to accomplish this (`docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions`).

Upload all of the source files needed to build your executable on `xunil-05` as a single zip file to the BBLearn site. Also, provide a brief report describing: (1) the design of your kernel(s) including the optimization techniques used (provide code or pseudocode to clarify the discussion); (2) the speedup achieved over the serial version; and (3) sensitivity of the kernel to thread-block size in terms of the execution time. Ignore the overhead due to CPU/GPU communication when reporting speedup. Use $a = 5$ and $b = 100$ as values for the upper and lower limits, respectively. Report the speedup achieved by the GPU code when using 10^4, 10^6, and 10^8 trapezoids to estimate the integral of the function.