Contents

Problem 1:

In a particular game, n participants decide to arrange themselves in a circle in positions numbered from 0 to n-1. The game goes as follows: Starting from a given participant, a judge starts counting participants and removes the m-th person from the cirle. Then, the judge starts counting again going around the circle until only one participant, the winner, is left. Write a C++ program that takes n and m as integers and determines which position corresponds to the only player remaining in the game. HINT: Use a circular queue where after all the participants are added, the head is connected to the tail.

Problem 2:

You work for a company that creates a process where n jobs have to be sequenced. Let's number these jobs from 0 to n-1. There are dependencies between these jobs, that is some jobs must be completed before other jobs can be started. The parameters to this problem are: n: the number of jobs, then for every job: a tuple t(i,j) that specifies that job i depends on job j. In other words, job(j) must be completed before job(i) can be started. There may be more that one tuple or dependency relationship for any given job, that is a job may have more than one dependency. Your task is to write a progam to schedule the execution of these jobs. In case of a tie, choose any jobs in the tie arbitrarily.
HINT: For each job i build a queue of jobs that depend on it, that is, jobs that must be scheduled after job i. Then, for each job, count the number of dependencies that job has. Then, create a queue the *ready* queue that contains all jobs that have no dependencies. Select one job from this queue, this is the current or active job, and remove 1 dependency from every job that depends on this job. You can also delete this job. When the dependecy count for any given job reaches 0, add that job to the ready queue. Proceed until all jobs are executed. A tuple of the form (i,i) indicates that the job has no dependencies.

Problem 3:

Mathematical background: Length of a curve.

Let us compute the length of the curve described by the $sin(x) function. Due to the symmetry of this curve we just need to compute the length of a quarter arc. (The length of the entire cycle is 4 times this value). From calculus, the length of this arc is

$$l = \int_0^{\frac{\pi}{2}} \sqrt{1 + \left(\frac{d}{dx} sin(x)\right)^2} dx$$

Since $\left(\frac{d}{dx} sin(x)\right) = -cos(x)$ we have the following two equivalent expressions for the length of a quarter arc of the $sin()$ function:

$$l = \int_0^{\frac{\pi}{2}} \sqrt{1 + cos^2(x)} dx = \int_0^{\frac{\pi}{2}} \sqrt{2 - sin^2(x)} dx$$

Further algebraic manipulation leads to the following equivalent representation:

$$l = \sqrt{2} \int_0^{\frac{\pi}{2}} \sqrt{1 - \frac{1}{2} sin^2(x)} dx = \sqrt{2} \int_0^{\frac{\pi}{2}} \sqrt{1 - \left(\frac{\sqrt{2}}{2}\right)^2 sin^2(x)} dx = \sqrt{2} E\left(\frac{\sqrt{2}}{2}\right)$$

where $E\,()$ is the *Complete Elliptic Integral of the Second Kind*

Evaluation of the complete elliptic integral of the second kind.
Ref: `https://en.wikipedia.org/wiki/Elliptic_integral#Complete_elliptic_integral_of_the_second_kind`

$$E(k) = \frac{\pi}{2} \left(1 - \left(\frac{1}{2}\right)^2 \frac{k^2}{1} - \left(\frac{1 \times 3}{2 \times 4}\right)^2 \frac{k^4}{3} - \ldots - \left(\frac{(2n-1)!!}{(2n)!!}\right)^2 \frac{k^{2n}}{2n-1} - \ldots\right)$$

where $k = \frac{\sqrt{2}}{2}$ and !! is the double factorial (`https://en.wikipedia.org/wiki/Double_factorial`)
For n even

$$n!! = \prod_{i=0}^{\frac{n}{2}} 2i$$

For n odd

$$n!! = \prod_{i=1}^{\frac{(n+1)}{2}} (2i - 1)$$

Problem 3 - Part a:
Compute the length of a quarter arc of the $sin()$ function using the formula $\sqrt{2}E\left(\frac{\sqrt{2}}{2}\right)$. You have to determine how many terms to use in the $E(k)$ series.
Problem 3 - Part b:
Repeat the computation but this time accumulating the $E(k)$ series from right to left.

Problem 4:

In this problem we are going to numerically integrate the equation for the length of the arc of the $sin()$ function

$$l = \sqrt{2}\int_0^{\frac{\pi}{2}} \sqrt{1 - \frac{1}{2}sin^2(x)}dx = \sqrt{2}\int_0^{\frac{\pi}{2}} \sqrt{1 - \left(\frac{\sqrt{2}}{2}\right)^2 sin^2(x)}dx = \sqrt{2}E\left(\frac{\sqrt{2}}{2}\right)$$

You can use either formula in your implementation.

Numerical Integration, the trapezoidal rule Ref:`https://en.wikipedia.org/wiki/Trapezoidal_rule`
The basic idea is to approximate the integral by subdividing the integration interval in n subintervals of equal length, then compute the value of the function at the midpoint of each interval. Multiply the result of this computation by the length of the subinterval and the add all the values. In other words

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \frac{f(x_k) + f(x_{k+1})}{2}h$$

where $h = \frac{(b-a)}{(n-1)}$ and n is the number of integration intervals.

Observe that this integral can be computed by the following sum:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \ldots + 2f(x_{n-1} + f(x_n))\right)$$

which is equivalent to

$$\int_a^b f(x)dx \approx h\left(f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{n-1} + f(x_n)) - \frac{h}{2}\left(f(x_0) + f(x_n)\right)\right.$$

Part a:

- Implement the function double **func**(const double x) that computes either one of the square root terms in the integral formula for l above for a given input x_i.
- Implement a generic evaluation function double **evalfunc**(const double stepsize, const unsigned int step) that calls the function **func** defined above for a given point x_i where $x_i = step \times stepsize$
- Provide the function double **trinteg**(const unsigned int setpsize, const unsigned int nsteps) that implements the reduction $sum = \sum_{i=0}^{n-1}$**evalfunc**(setpsize, i)
- add the trapezoidal correction. Don't forget to multiply the final result by h (Note: so far, we have not multiplied sum by the stepsize h)
- compute $l = \sqrt{2} \times sum$
- compare your solution with your result from Problem 3.
- time the execution. The timing code will be provided

Part b:
Repeat the exercise but templetize the functions, like **template <class T> T func(const T)**. Now, run the code again for both double and float (like **trinteg<double>(..)** and **trinteg<float>(...)**.
Part c:
Implement **openmp** acceleration of reduction loop in **trinteg**. Hint: reduction (+:<var>). Try different values for the number of threads. Time it.
Part d: GPU implemnetation
Compute a gpu array *vals* of size *nsteps* **vals[step] = func(x)** where

```
x=step*stepsize
```

and

```
step=blockId.x*blockDim.x+threadIdx.x
```

The reduction phase on GPUs is not a trivial operation. I will provide the reduction routine. For those interested, see: `https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf`

As usual, check the correctnes of the result and compare the gpu time with the other execution times.