COMP 605: Introduction to Parallel Computing Homework 6: GPU/CUDA Programming: Calculating PI and Prime Numbers.

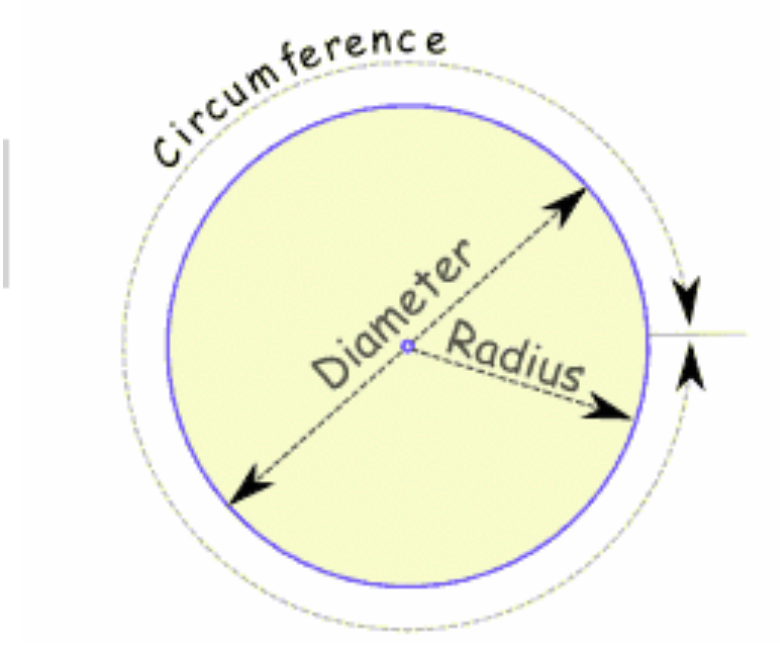
Mary Thomas

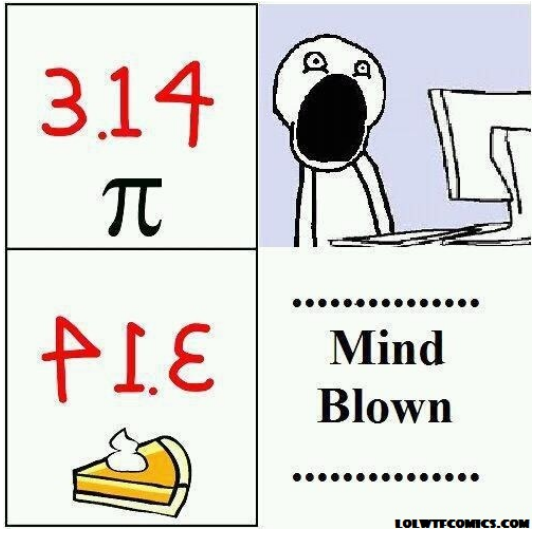
Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Due: 04/27/17
Posted: 04/25/17
Updated: May 9, 2017

Table of Contents

HW #6, P1: Using Numerical Integration to Estimate π



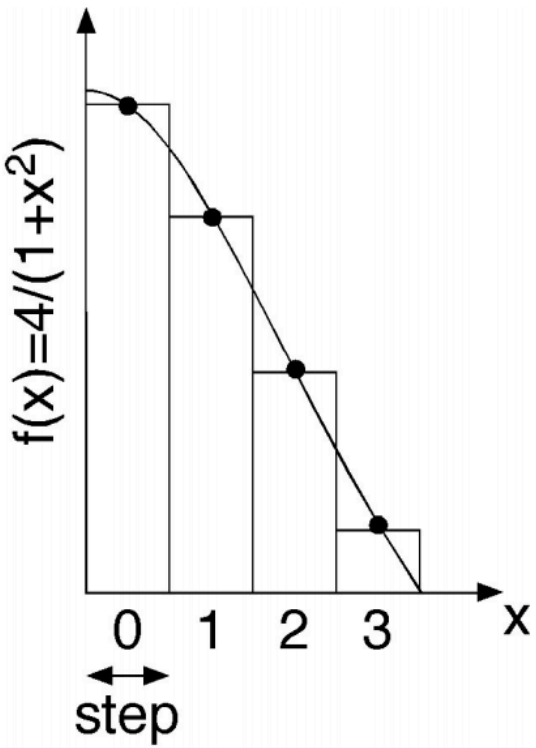


$$\pi = \frac{Circumference\ of\ a\ Circle}{Diameter\ of\ a\ Circle}$$

Image Source: `http://www.mathsisfun.com/numbers/pi.html`

HW #6, P1: Using Numerical Integration to Estimate π

- Integral representation for π
  $\int_0^1 dx \frac{4}{1+x^2} = pi$
- Discretize the problem:
  $\Delta = 1/N : step = 1/N_{areas}$
  $x_i = (i + 0.5)\Delta (i = 0, \ldots, N_{areas} - 1)$
  $\sum_{i=0}^{N-1} \frac{4}{1+x_i^2}\Delta \cong \pi$



π Formulae: http://en.wikipedia.org/wiki/Approximations_of_pi
Image: http://cacs.usc.edu/education/cs596/mpi-pi.pdf

HW #6, P1: Using Numerical Integration to Estimate π

```
#include <stdio.h>
#define NAREA 10000000
void main() {
    int i; double step,x,sum=0.0,pi;
    step = 1.0/NAREA;
    for (i=0; i<NAREA; i++) {
        x = (i+0.5)*step;
        sum += 4.0/(1.0+x*x);
     }
    pi = sum*step;
    printf(PI = %f\n,pi);
}
```

HW #6, P1: Instructions

- Write a CUDA program that uses *numerical integration* to estimate π.
- Find a reference value for π to the limits of a double precision number.
- Estimate π to the limits of a double precision number.
- Calculate the value for π as a function of the number or areas used and number of threads.
- Calculate the error of your estimate: $Err = \pi_{ref} - \pi_{measured}$
- Use double precision for all calculations and outputs.

HW #6, P1: Instructions (cont.)

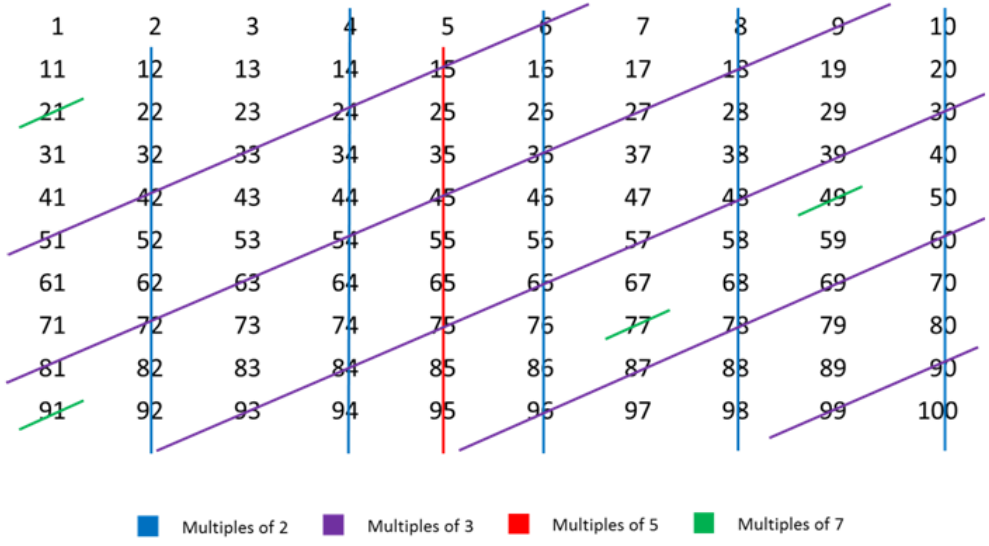
- Parse all key variables from the command line.
- Run the jobs using the batch queue
- ProbSize Scaling:
  - Choose N_{areas} to allow scaling from appx 10^3 to $> 10^7$ or greater.
- Thread scaling:
  - Vary the total number of threads on the GPU Device.
  - Vary the total number of threads by changing the number of threads-per-block and blocks-per-grid (e.g.):

```
int threadsperblock=atoi(argv[1]);  /* read num thds from command line */
blocksPerGrid = imin( 32, (N+threadsPerBlock-1) / threadsPerBlock );
add<<<blocksPerGrid,threadsPerBlock>>>( dev_a, dev_b, dev_c);
```

  - What is the max number of total threads you can use and why?
- Time the job runs to check that you getting the proper scaling.

HW #6, P2: Calculating Prime Numbers

- Develop a CUDA-based version of the Sieve of Eratosthenes approach to calculate all the prime numbers below some number N.
- Determine $N = [1, 2, 3, .., N_{max}]$ that can run on your device.
- Run jobs using the batch queue.



Img Src: http://mathworld.wolfram.com/SieveofEratosthenes.html

HW #6, P2: Instructions (cont.)

- Parse all key variables from the command line.
- Run the jobs using the batch queue
- ProbSize Scaling:
  - Choose N_{areas} to allow scaling from appx 10^3 to $> 10^7$ or greater.
- Thread scaling: vary the total number of threads ($Thds_{total}$) on the GPU Device:
  - Vary $Thds_{total}$ by changing the number of threads-per-block and blocks-per-grid:

```
int threadsperblock=atoi(argv[1]);  /* read num thds from command line */
blocksPerGrid = imin( 32, (N+threadsPerBlock-1) / threadsPerBlock );
add<<<blocksPerGrid,threadsPerBlock>>>( dev_a, dev_b, dev_c);
```

  - What is the max number of total threads you can use and why?
- Time the job runs to check that you getting the proper scaling.

What to Report/Turn in for both problems:

- Create the homework directory *USER*/*hw*/*hw*5 with correct access permissions.
- Short lab report with comments, figures and table labels.
- Explain your results for Thread and ProbSize scaling.
- Include relevant tables of your test data
- Evidence you ran your jobs using the batch queue (short/small job); examples of batch scripts
- Plot the runtime as a function of the number of threads and probsize.
- A copy of your code (single spaced, two sided, two column format is OK).
- Reference key sources of information *in your report and code* where applicable.

The CUDA Compiler support for doubles: nvcc

- you can install CUDA toolkit, compile code without a GPU device.
- To compile use: *nvcc*
- **NOTE: CUDA does not support doubles on the *device* by default:** You need to add the switch "-arch sm_13" (or a higher compute capability) to your nvcc command:

```
[mthomas/dblTst]
[mthomas/dblTst]nvcc -o dblTst dblTst.cu
nvcc warning : The 'compute_10' and 'sm_10' architectures are
deprecated, and may be removed in a future release.
ptxas /tmp/tmpxft_00006578_00000000-5_dblTst.ptx, line 76;
warning : Double is not supported. Demoting to float
[mthomas/dblTst]
```

```
[mthomas/dblTst]
[mthomas/dblTst] nvcc -arch=sm_13 -o dblTst dblTst.cu
[mthomas/dblTst]
```