Table of Contents

1. HW #4, P1: Using Numerical Integration to Estimate $\pi$
2. HW #4, P2: Calculating Prime Numbers
3. What to Report/Turn in
HW #4, P1: Using Numerical Integration to Estimate π

\[ \pi = \frac{\text{Circumference of a Circle}}{\text{Diameter of a Circle}} \]

Image Source: http://www.mathsisfun.com/numbers/pi.html
HW #4, P1: Using Numerical Integration to Estimate $\pi$

- Integral representation for $\pi$
  \[ \int_0^1 dx \frac{4}{1+x^2} = \pi \]

- Discretize the problem:
  \[
  \Delta = \frac{1}{N} : \text{step} = \frac{1}{N_{\text{areas}}}
  \]
  \[x_i = (i + 0.5) \Delta (i = 0, \ldots, N_{\text{areas}} - 1)\]
  \[\sum_{i=0}^{N-1} \frac{4}{1+x_i^2} \Delta \approx \pi\]

Image: [http://cacs.usc.edu/education/cs596/mpi-pi.pdf](http://cacs.usc.edu/education/cs596/mpi-pi.pdf)
HW #4, P1: Using Numerical Integration to Estimate $\pi$

```c
#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 #4, P1: Instructions

- Write an OpenMP program that uses numerical integration to estimate $\pi$.
- Use OpenMP directives for the parallelism.
- You may write your own code, use Pacheco example (e.g. `mpi_trap4.c`), or a program found online.
- See the *Trap* examples discussed in Pacheco 2011, Chs 3, 4, and 5.
- Find a reference value for $\pi$ to the limits of a double precision number.
- Estimate $\pi$ to the limits of a double precision number.
- Calculate the value for $\pi$ as a function of the number or areas used and number of threads.
- Calculate the error of your estimate: $Err = \pi_{\text{ref}} - \pi_{\text{measured}}$
- Use double precision for calculations and outputs.
HW #4, P1: Using Numerical Integration to Estimate $\pi$

**Instructions (cont.)**

- Run the jobs using the batch queue
- **Thread scaling:** Vary the number of threads $\# Thds$ used:
  - Where $\# Thds = [1, 2, \ldots, Thd_{max}]$.
  - What is the max number you can use? Why?
  - Use *binding* to control the number of threads per core
- **ProbSize Scaling:**
  - Choose $N_{areas}$, such that $N_{areas}$ is evenly divisible by $\#Thds$.
  - Choose a few values for $N_{areas}$ that allow scaling from $10^3$ to $> 10^7$ or $10^8$.
- Time the job runs, calculate run time statistics. Are the timings reproducible?
Develop an OpenMP version based on the Sieve of Eratosthenes approach to calculate all the prime numbers below some number $N$:

- Run jobs using the batch queue.
- Determine $N = [1, 2, 3, \ldots, N_{\text{max}}]$ for tuckoo.
- Vary the number of threads
- Use thread binding for better performance
- Time the job runs

Img Src: http://mathworld.wolfram.com/SieveofEratosthenes.html
HW #4, P1: Instructions (cont.)

- Use OpenMP directives for the parallelism.
- Run the jobs using the batch queue
- Thread scaling: Vary the number of threads #Thds used:
  - Where #Thds = [1, 2, . . . , Thd_{max}].
  - What is the max number you can use? Why?
  - Use binding to control the number of threads per core
- ProbSize Scaling:
  - Choose N, such that N is evenly divisible by #Thds.
  - Choose a few values for N that allows scaling from $10^3$ to $> 10^7$ or $10^8$.
- Time the job runs, calculate run time statistics. Are the timings reproducible?
What to Report/Turn in for both problems:

- 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)
- Plot the runtime as a function of the number of threads or 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 (Pacheco, lectures, Web, ).