Calculation of π on GPU with Monte-Carlo method using CUDA

Alexey Ivakhnenko
ivakhnenko@parallel-compute.com

January 13, 2013

Contents

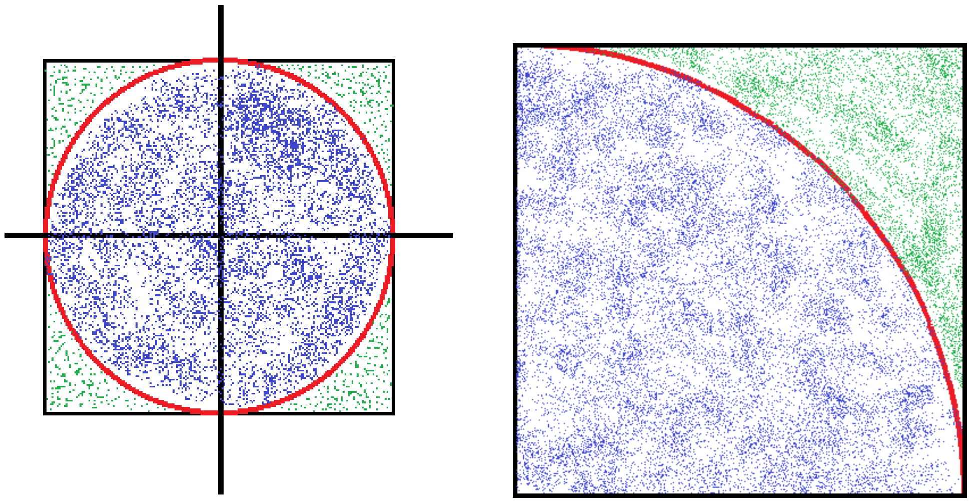

Figure 1. Random points distribution inside unit square (left) and in [0,0] - [1,1] area for π calculation using Mote-Carlo method (right)

1. Introduction

Calculation of π number is one of the problems frequently used in testing supercomputers and teaching parallel computing. Algorithms of calculating π number, suitable for parallel processing, could be subdivided into two classes: iterative and probabilistic. Among iterative methods, the most well-parallelized implementations are possible for those, that allow to compute separate digits independently, for instance, the Bailey–Borwein–Plouffe formula (in binary or hexadecimal representation):

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right).$$

Among probabilistic methods the most widely known is the Monte-Carlo method [1]. According to the Monte-Carlo method, the ratio between the number of points inside (0,0)-centred unit circle and the number of points inside the enclosing $(-1,-1)-(1,1)$ square with the same center approximates $\pi/4$ value (fig. 1).

2. Task definition

Given the number of points N, generate a random distribution in $(0,0)-(1,1)$ area and calculate the π number using CPU and GPU. The resulting π values should be printed out along with the execution times.

3. Proposed method

The following method could be used to implement Monte-Carlo algorithm:

1. Generate 2 sequences of values – X and Y coordinates
2. Calculate $V = X^2 + Y^2$ for each pair of values

3. If $V < 1$ return 1 to one of the vectors (X or Y), else return 0
4. Perform a reduction with sum operator
5. Multiply the result by $4/N$

4. Implementation requirements

4.1. Input data

- N – number of points;

4.2. Output data

- The time of GPU and CPU programs execution;
- π values calculated by GPU and CPU programs (results may be different).

4.3. Implementation

The program is required to work on Linux machine. It is also required to use *device API* of CURAND library for generating random numbers on GPU and to use CUDA kernel-functions for all GPU calculations.

5. The expected result

1. Getting familiar with mathematical constants calculation methods.
2. Gain basic experience in writing CUDA programs and using CURAND library.

References

[1] Monte-carlo method – http://www.mathresource.iitb.ac.in/linear%20algebra/FinalPi_Method/dots/index.html.