High-Performance Numerical Computation of Multidimensional Integral using Random Sampling

http://dx.doi.org/10.28932/jutisi.v11i3.12999

Riwayat Artikel
Received: 10 Agustus 2025 | *Final Revision*: 22 November 2025 | *Accepted*: 22 November 2025

Andreas Widjaja✉ #1, Tjatur Kandaga Gautama#2, Sendy Ferdian Sujadi#3, Bernadus Indra Wijaya#4

Faculty of Smart Technology and Engineering, Universitas Kristen Maranatha
Jalan Surya Sumantri No. 65, Bandung, Jawa Barat 40164

[1]andreas.widjaja@it.maranatha.edu

[2]tjatur.kandaga@it.maranatha.edu

[3]sendy.fs@it.maranatha.edu

[4]2072003@maranatha.ac.id

✉Corresponding author: andreas.widjaja@it.maranatha.edu

Abstract — ***This study examines the use of high-performance computing to carry out multidimensional integral calculation based on stochastic techniques, particularly in the context of random sampling integration, also known as Monte Carlo integration. Considering that traditional methods are facing extreme difficulty especially in high-dimension when encountered with "dimensionality curse", random sampling technique to estimate integral values is used. This technique is superior in many aspects, for example in terms of scalability and flexibility, even in complex and irregular domains. In particular, the work concentrates on the case of calculating the volume of a multidimensional sphere using random sampling integration technique which introduces a framework that employs the Graphics Processing Unit (GPU) to carry out these computations more effectively. Using dimensionalities from 2 to 24, the study compares both accuracy and computation time of the method. The results show that the random sampling integration method attains high accuracy in the computation of π which is used as a benchmark. The computational model is implemented in CUDA C/C++, taking advantage of GPU parallelism to process large sample sizes, in which the computations are performed efficiently. It is demonstrated in general that random sampling integration is a viable approach to high-dimensional problems when combined with rapid GPU parallelization.***

Keywords— ***Monte Carlo integration; Multidimensional integral*; *parallel computing; random sampling*; *stochastic*.**

Komputasi Numerik Berkinerja Tinggi dari Integral Multidimensi Menggunakan Sampel Acak

Abstrak — **Penelitian ini mengkaji penggunaan komputasi berkinerja tinggi untuk melakukan perhitungan integral multidimensi berdasarkan teknik stokastik, khususnya dalam konteks integrasi sampling acak, yang juga dikenal sebagai integrasi Monte Carlo. Mengingat metode tradisional menghadapi kesulitan ekstrem terutama pada dimensi tinggi ketika dihadapkan pada "*dimensionality curse*", teknik sampling acak digunakan untuk mengaproksimasi nilai integral. Teknik ini unggul dalam banyak aspek, misalnya dalam hal skalabilitas dan fleksibilitas, bahkan pada domain yang kompleks dan tidak teratur. Secara khusus, penelitian ini berfokus pada kasus perhitungan volume bola multidimensi menggunakan teknik integrasi sampling acak, yang memperkenalkan kerangka kerja yang memanfaatkan unit pemrosesan grafis (GPU) untuk melakukan perhitungan ini secara lebih efektif. Dengan dimensi dari 2 hingga 24, penelitian ini membandingkan akurasi dan waktu komputasi metode tersebut. Hasil menunjukkan bahwa metode integrasi sampling acak mencapai akurasi tinggi dalam perhitungan π yang digunakan sebagai**

acuan. Model komputasi diimplementasikan dalam CUDA C/C++, memanfaatkan paralelisme GPU untuk memproses ukuran sampel besar, di mana perhitungan dilakukan secara efisien. Secara umum, terbukti bahwa integrasi sampling acak merupakan pendekatan yang layak untuk masalah multidimensi ketika dikombinasikan dengan paralelisasi GPU yang cepat.

Kata kunci— **Integral multidimensi; integrasi Monte Carlo; komputasi paralel; sampling acak, stokastik.**

I. INTRODUCTION

Numerical integration is one of the most versatile and important techniques employed in scientific computation. It allows the user to compute definite integrals for functions that would otherwise be difficult to analyze. For example, when single or multidimensional integrals are involved, many traditional deterministic integration techniques like Gaussian quadrature or even trapezoidal and Simpson's rules [1] perform quite inefficiently. These traditional methods are numerically "hard" due to the exponentiation of the dimensionality cost of computation. This constrain is also known as the "curse of dimensionality" [2] [3] [4]. These considerations imply that in high dimensional problems, it is necessary to resort to more feasible numerical methods. Of these, random sampling algorithms, in particular, is also known as Monte Carlo integration (it is a generic name of algorithms where random numbers are used, coined from the Monte Carlo Casino in Monaco), have proven to be effective and flexible [5] [6].

The concept of multidimensional integral can be presented in the following form mathematically:

$$I = \int_{\Omega} f(\mathbf{x}) d\mathbf{x} \qquad (1)$$

is the integral in d-dimensional space, where $\mathbf{x} = (x_1, x_2, \dots, x_{d-1})$ is a vector in $(d-1)$ dimensions, $f(\mathbf{x})$ is the function which we wish to integrate and Ω is the $(d-1)$-dimensional domain to be integrated. For multidimensional problems ($d > 3$), the amount and density of points for classical grid techniques, such as multivariable Riemann sum [7], becomes tremendously larger. In particular, the evaluation of a d-dimensional space integral with N points per dimension requires N^{d-1} evaluations. This scaling problem makes such methods inappropriately efficient when the dimension, d, increases. Unlike traditional techniques such as the aforementioned Riemann sum, which is a deterministic approach, this random sampling technique employs a non-deterministic approach, that is, each computational result provides a different outcome [5].

Furthermore, integrals that appear in realistic problems usually have the additional challenge of featuring irregular domain or highly oscillatory functions which makes it particularly worse for deterministic methods. Approaches such as Random sampling methods or Monte Carlo methods based on probability theories easily overcome many of these complications as they offer improved scalability alongside reduced precision as toughness for problems.

Random sampling technique which is the core algorithm of Monte Carlo integration is based on the law of large numbers, where it approximates the value of an integral by taking the mean of random samples of the integrand. This method is useful when the average of the samples taken approaches the value of the integral as the number of samples approached infinity.

Assuming $f(\mathbf{x}) = f(x_1, x_2, \dots, x_{d-1})$ is defined on a $(d-1)$-dimensional domain Ω then the estimate for the Monte Carlo will be:

$$\hat{I} = \frac{|\Omega|}{N} \sum_{i=1}^{N} f(\mathbf{x}_i), \qquad (2)$$

where $|\Omega|$ is the "area" of the integration domain, N is the random sample size, and $\{\mathbf{x}_i\}_{i=1}^{N}$ are the uniformly distributed random points over Ω. The error of this estimate is reduced as $O(1/\sqrt{N})$ regardless of the dimensionality d which makes Monte Carlo integration for high-dimensional problems favorable [5] [6].

Random sampling techniques have proven to be very useful and applicable in many areas because of their flexibility and ease of use. For example, Achilles and Sharma et al. used Monte Carlo to approximate the value π using random sampling points counting in a simple circle [8] [9]. In engineering and physics, Monte Carlo methods are very popular when it comes to problems related to quantum mechanics, statistical mechanics, and radiative transfer. For example, there is often the need to perform integration, which is generally high dimension, in computing path integrals [10] within quantum field theory as studied by Metropolis et al. [11]. Applications in Finance including complex derivatives, portfolio optimization, and even risk management usually depend on Monte Carlo methods for computations. In stochastic processes and multivariate distributions, problems usually involve multidimensional integrals. This phenomenon occurs naturally as studied by Glasserman [12]. In the field of Bayesian statistics and machine learning practices, Bayesian computation techniques often require performing integration over high-dimensional posterior distributions. Markov Chain Monte Carlo (MCMC) sampling and Variational Bayes have been great contributors and tools in machine learning as well as data science, as shown by Andrieu et al. [13]. When it comes to climate modeling and environmental sciences, computing climate models, especially the

uncertainty quantification and even the sensitivity analysis, Monte Carlo integration is a well-established technique (see Saltelli et al. [14]). In statistics, Naimi et al. performed Monte Carlo integration in simulation designs, where they computed true estimand values of the designs, for a simple and complex one [15].

In this study, high-performance computation is performed, that is, utilizing a graphic processing unit (GPU), which has thousands of parallel computing cores, enabling a relatively large sample size of the computation is executed at once in parallel. This gives the advantage of computing the numerical integration in high-dimension to some extent. The purpose of this research is to find out and demonstrate the capability of the random sampling technique of the high-dimension numerical integration with the aid of parallel computing device, a GPU, will produce accurate results in a relatively short computation time. Hence, we formulate the research question as: "Can the random sampling technique of the high-dimensional numerical integration produces accurate results in a relatively short computation time using a GPU?"

For the sake of simplicity, yet still maintain the not-so-simple function, we choose the *d*-dimensional sphere as the function to be the integrand. The main reason of choosing such function is the obvious spherical symmetry, relatively easy and fast to compute, and including a square root, the relatively often basic function used in most practical computation. In other word, the function is a representative of computationally "cheap" function.

II. METHODOLOGY

This research methodology consists of three main steps, which are survey, computation, and results. The workflow of the methodology is shown in Figure 1 below.

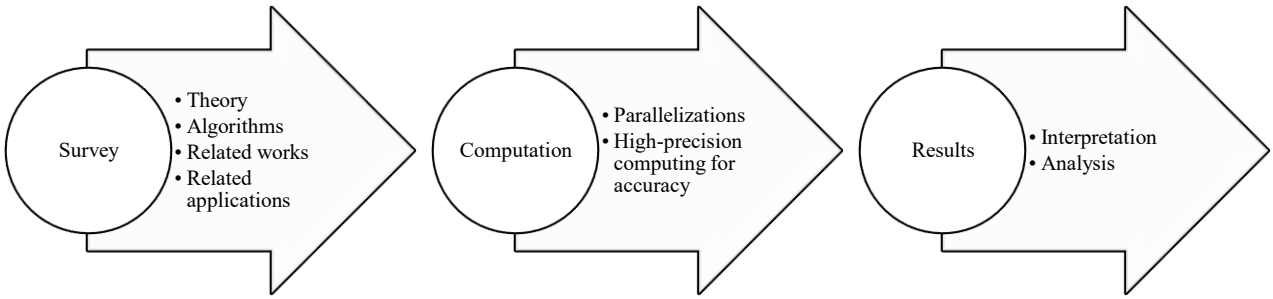

Figure 1. Research methodology workflow

A. Survey

The first step is survey of related computing theory of the technique and its algorithms. Many works related to the technique including its applications is also studied, to strengthen the understanding and foundations.

B. Computation

The second step is to develop the implementation of the computation model in the high-performance computing environment, that is, the parallel computation utilizing a relatively fast and powerful GPU. Since the research is aiming to finding out the capability of the technique, hence it is crucial to maintain the highest numerical precision available during internal computation, which, in this case will produce relatively high accuracy results.

C. Results

The next and final step is to acquire the computation results, including interpreting and analysis. Interpretation is done by observing the numerical output of the technique. Analysis is done by comparing the computational results with analytical values which are readily available. In addition, a performance comparison with another random sampling algorithm, a random point counting technique is also done.

D. Theoretical Foundation

The random sampling algorithm for multidimensional integration is briefly presented here, including its application to compute the volume of multidimensional sphere, as a relatively simple case implementation of the algorithm.

- *Random Sampling Integration Technique*

The average of a function, denoted as $\langle f(\mathbf{x}) \rangle$, by definition, is

$$\langle f(\mathbf{x}) \rangle = \frac{1}{|\Omega|} \int_{\Omega} f(\mathbf{x}) d\mathbf{x}\,, \qquad (3)$$

while computation of $\langle f(\mathbf{x}) \rangle$ can be approximated by

$$\langle f(\mathbf{x})\rangle \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i), \qquad (4)$$

where N is the number of random samples (sample size) taken and $\{\mathbf{x}_i\}_{i=1}^{N}$ are the uniformly distributed random points over Ω. Here the approximation is better when N is large enough. Equations (3) and (4) can be equated as

$$\frac{1}{|\Omega|}\int_{\Omega} f(\mathbf{x})d\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i), \qquad (5)$$

leading to

$$\int_{\Omega} f(\mathbf{x})d\mathbf{x} \approx \frac{|\Omega|}{N}\sum_{i=1}^{N} f(\mathbf{x}_i). \qquad (6)$$

It can be seen that equation (6) is indeed equation (2). Therefore, the basic idea of random sampling integration is to compute the average value of a function and multiply with the "area" of its domain $|\Omega|$.

The numerical integration in d-dimensional space with $(d-1)$-dimensional domain Ω as in equation (1) can be expressed in the simplest form as

$$I = \iiint_{\Omega} \cdots \int f(x_1, x_2, \dots, x_{d-1})\, dx_1 dx_2 dx_3 \cdots dx_{d-1} \qquad (7)$$

which numerical approximation can be computed using equation (6), to become

$$\iiint_{\Omega} \cdots \int f(x_1, x_2, \dots, x_{d-1})\, dx_1 dx_2 dx_3 \cdots dx_d \approx \frac{|\Omega|}{N}\sum_{i=1}^{N} f((x_1, x_2, \dots, x_{d-1})_i). \qquad (8)$$

where $\{(x_1, x_2, \dots, x_{d-1})_i\}_{i=1}^{N}$ are the uniformly distributed random points over Ω.

- *Volume of d-dimensional Sphere*

  One of simple cases of multiple integration is volume of a sphere with radius r in d dimensions, V_d, can be calculated as:

$$V_d = 2^d \iiint \int_{x_1^2+x_2^2+\cdots+x_{d-1}^2 \le r^2} \sqrt{r^2 - x_1^2 - x_2^2 - \cdots - x_{d-1}^2}\, dx_1 dx_2 dx_3 \cdots dx_{d-1}. \qquad (9)$$

The integral is the volume of the sphere in the first "orthant" if its d-dimensional space, that is, $0 \le x_i \le r, i = 1, 2, \dots, d$. A unit sphere in three dimensions is illustrated in Figure 2.

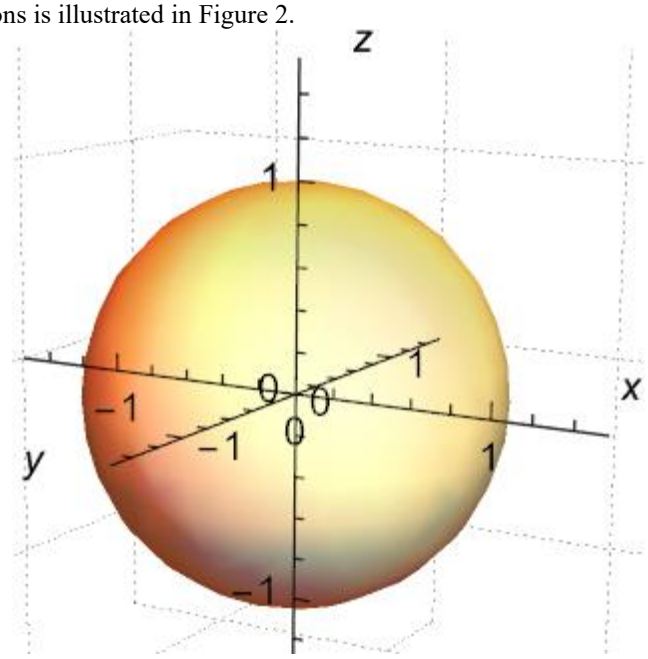

Figure 2. Illustration of a unit sphere in three dimensions.

Typically, an orthant [16] or hyperoctant [17] in *d*-dimensional Euclidean space can be viewed as the intersection of *d* half-spaces that are mutually orthogonal, analogous to a quadrant in the plane or an octant in three dimensions. The integral is $1/2^d$ part of the volume because of 2^d spherical symmetry. The domain of integration, Ω, is the first orthant of its $(d-1)$-dimensional "area" where $0 \le x_i \le r$, $i = 1, 2, \dots, d-1$. A traditional way of computing this integration numerically is by using multidimensional Riemann sum [7] [18] [19] [20], which, for a sphere, typically has a form of:

$$V_d \approx 2^d \sum_{x_1^2+x_2^2+\cdots+x_{d-1}^2 \le r^2} \Delta x_1 \Delta x_2 \cdots \Delta x_{d-1} \sqrt{r^2 - x_1^2 - x_2^2 - \cdots - x_{d-1}^2}\,. \qquad (10)$$

For illustration, an integration of over a unit sphere in three dimensions in the first octant is shown in Figure 3, where every bar illustrates the quantity $\Delta x \Delta y \sqrt{1-x^2-y^2}$.

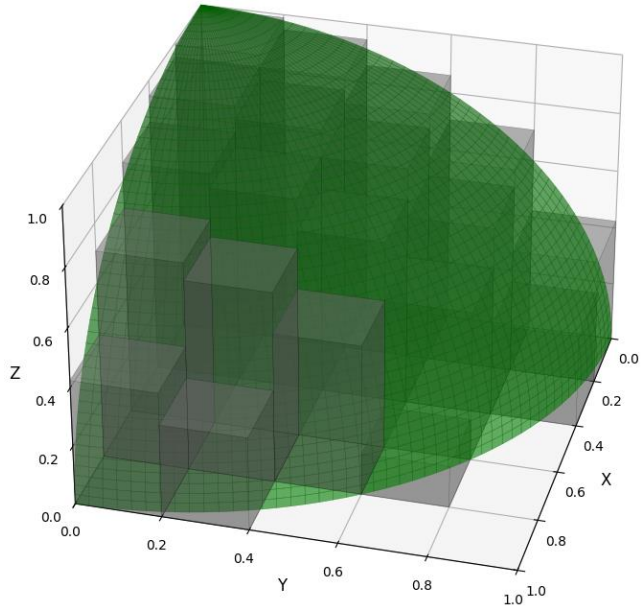

Figure 3. Illustration of Riemann sum integration over the first octant of a unit sphere in three dimensions

For comparison, using direct analytical integration technique in spherical coordinates [21] [22] [23] [24] [25], one can derive the closed formula of volume of *d*-dimensional sphere with radius *r*, which has the form as

$$V_d = \frac{\pi^{d/2} r^d}{\Gamma\left(\frac{d}{2}+1\right)}. \qquad (11)$$

For the sake of convenience, the exact and approximate values of the volume of unit spheres, V_d, and its orthant, $V_d/2^d$, with radius $r = 1$, according to equation (11) are presented in Table 1. Notice that Γ(…) in equation (11) is the gamma function. Surprisingly, as *d* increases, the value of V_d increases up to the maximum value when $d = 5$, and after that, V_d diminish rapidly, as shown in Figure 4.

TABLE 1
UNIT SPHERE VOLUMES, V_D, AND $V_D/2^D$, IN *D* DIMENSIONS, WITH THEIR MACHINE DOUBLE PRECISION APPROXIMATE VALUES

Dimension, *d*	$\boldsymbol{V_d}$	Approximate $\boldsymbol{V_d}$	$\boldsymbol{V_d/2^d}$	Approximate $\boldsymbol{V_d/2^d}$
2	$\boldsymbol{\pi}$	$\boldsymbol{3.14159265358979}$	$\boldsymbol{\frac{\pi}{4}}$	$\boldsymbol{0.785398163397448}$
3	$\frac{4\pi}{3}$	4.18879020478639	$\frac{\pi}{6}$	0.523598775598299
4	$\frac{\pi^2}{2}$	4.93480220054468	$\frac{\pi^2}{32}$	0.308425137534042

Dimension, d	V_d	Approximate V_d	$V_d/2^d$	Approximate $V_d/2^d$
5	$\frac{8\pi^2}{15}$	5.26378901391432	$\frac{\pi^2}{60}$	0.164493406684822
6	$\frac{\pi^3}{6}$	5.16771278004997	$\frac{\pi^3}{384}$	0.0807455121882808
7	$\frac{16\pi^3}{105}$	4.72476597033140	$\frac{\pi^3}{840}$	0.036912234143214
8	$\frac{\pi^4}{24}$	4.05871212641677	$\frac{\pi^4}{6144}$	0.0158543442438155
9	$\frac{32\pi^4}{945}$	3.29850890273871	$\frac{\pi^4}{15120}$	0.00644240020066154
10	$\frac{\pi^5}{120}$	2.55016403987735	$\frac{\pi^5}{122880}$	0.00249039457019272
11	$\frac{64\pi^5}{10395}$	1.88410387938990	$\frac{\pi^5}{332640}$	$9.19972597358350 \times 10^{-4}$
12	$\frac{\pi^6}{720}$	1.33526276885459	$\frac{\pi^6}{2949120}$	$3.25991886927390 \times 10^{-4}$
13	$\frac{128\pi^6}{135135}$	0.910628754783283	$\frac{\pi^6}{8648640}$	$1.11160736667881 \times 10^{-4}$
14	$\frac{\pi^7}{5040}$	0.599264529320792	$\frac{\pi^7}{82575360}$	$3.65762041821773 \times 10^{-5}$
15	$\frac{256\pi^7}{2027025}$	0.381443280823304	$\frac{\pi^7}{259459200}$	$1.16407251227815 \times 10^{-5}$
16	$\frac{\pi^8}{40320}$	0.235330630358893	$\frac{\pi^8}{2642411520}$	$3.59086044859151 \times 10^{-6}$
17	$\frac{512\pi^8}{34459425}$	0.140981106917139	$\frac{\pi^8}{8821612800}$	$1.07560048612319 \times 10^{-6}$
18	$\frac{\pi^9}{362880}$	0.0821458866111282	$\frac{\pi^9}{95126814720}$	$3.13361689037812 \times 10^{-7}$
19	$\frac{1024\pi^9}{654729075}$	0.0466216010300885	$\frac{\pi^9}{335221286400}$	$8.89236469842692 \times 10^{-8}$
20	$\frac{\pi^{10}}{3628800}$	0.025806891390014	$\frac{\pi^{10}}{3805072588800}$	$2.46113695049420 \times 10^{-8}$
21	$\frac{2048\pi^{10}}{13749310575}$	0.013949150409021	$\frac{\pi^{10}}{14079294028800}$	$6.65147324038553 \times 10^{-9}$
22	$\frac{\pi^{11}}{39916800}$	0.00737043094571435	$\frac{\pi^{11}}{167423193907200}$	$1.75724767344340 \times 10^{-9}$
23	$\frac{4096\pi^{11}}{316234143225}$	0.00381065638685212	$\frac{\pi^{11}}{647647525324800}$	$4.54265640598789 \times 10^{-10}$
24	$\frac{\pi^{12}}{479001600}$	0.00192957430940392	$\frac{\pi^{12}}{8036313307545600}$	$1.15011591279741 \times 10^{-10}$
25	$\frac{8192\pi^{12}}{7905853580625}$	$9.57722408823173 \times 10^{-4}$	$\frac{\pi^{12}}{32382376266240000}$	$2.85423519856683 \times 10^{-11}$

p-ISSN : 2443-2210
e-ISSN : 2443-2229

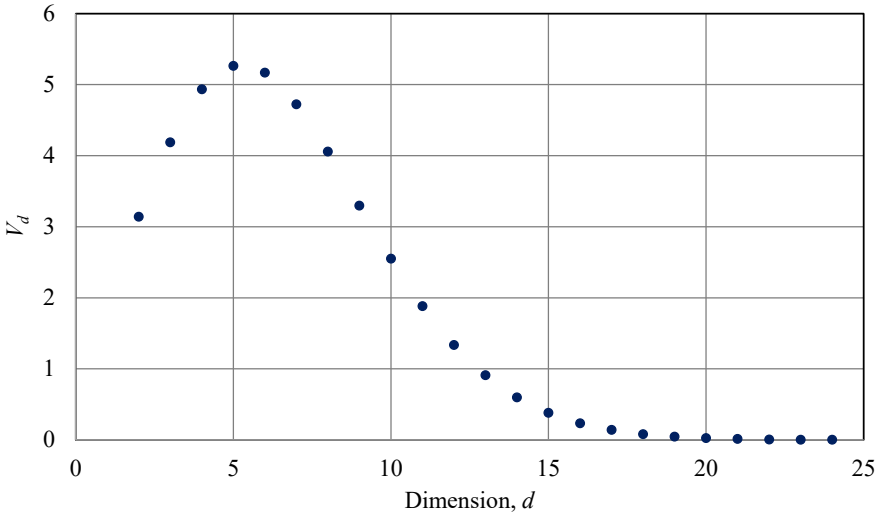

Figure 4. Volume of a unit sphere, V_d, in d dimensions

E. Computation Model

Our implementation of the computation model is simply computing the unit sphere volume of the part where all coordinates are positive, that is, the first "orthant" only, where $0 \leq x_i \leq 1, i = 1, 2, \ldots, d$, which is $V_o = V_d/2^d$. The exact and approximate values of $V_d/2^d$, in machine double precision, are also presented in Table 1. The values of $V_d/2^d$ diminish rapidly as d increases (see Figure 5).

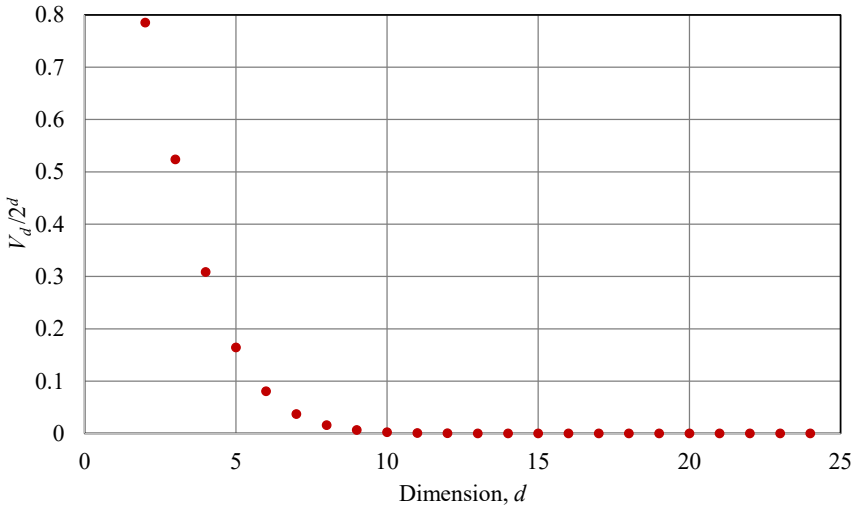

Figure 5. Volume of an orthant of a unit sphere, $V_d/2^d$, in d dimensions

As indicator of computation accuracy, we compute (or extract) the approximate value of π. The reason behind this is that π is known as a universal constant, which, despite it is an irrational number, its approximate value is well known very accurate to many (even billions and trillions) significant figures [26] [27] [28].

Computation of π from $V_o = V_d/2^d$ is done using equation (11), $\pi = \left(V_d\, \Gamma(1 + d/2)\right)^{2/d}$, and values from Table 1 for various value of d. Instead of using the matching decimal digits of π, we use the metric of the accuracy of computed value of π, which is calculated as

$$a_\pi = 1 - \frac{|\Delta\pi|}{\pi}, \qquad (12)$$

where $\Delta\pi = \hat{\pi} - \pi$ is the difference of computed $\hat{\pi}$ and the "actual" π, that is, a value of machine double precision π = 3.14159265358979, which is used here.

- *Volume Approximation by Random Sampling of the Unit Sphere Function*

Computation of the volume of the unit sphere's first orthant, $V_o = V_d/2^d$ is performed by using the integral in equation (9) where numerically approximated using random sampling in equation (8), as follow:

$$V_o = \iiint \int_{x_1^2+x_2^2+\cdots+x_{d-1}^2 \le 1} \sqrt{1-x_1^2-x_2^2-\cdots-x_{d-1}^2}\, dx_1 dx_2 dx_3 \cdots dx_{d-1} \approx \frac{|\Omega|}{N} \sum_{i=1}^{N} f((x_1, x_2, \dots, x_{d-1})_i)\,, \qquad (13)$$

where the function f in the right-hand side is averaged over the domain "area", $\Omega = (1-0)^{d-1} = 1$. The function f itself has the form of:

$$f(x_1, x_2, \dots, x_{d-1}) = \begin{cases} \sqrt{1-x_1^2-x_2^2-\cdots-x_{d-1}^2}\,, & x_1^2+x_2^2+\cdots+x_{d-1}^2 \le 1 \\ 0, & \text{elsewhere} \end{cases} \qquad (14)$$

A random sample of size N of function (14) is picked, and averaged, as in the equation (13), leading to the computed value of V_o. The situation of sampling of the function in three dimensions is illustrated in Figure 6.

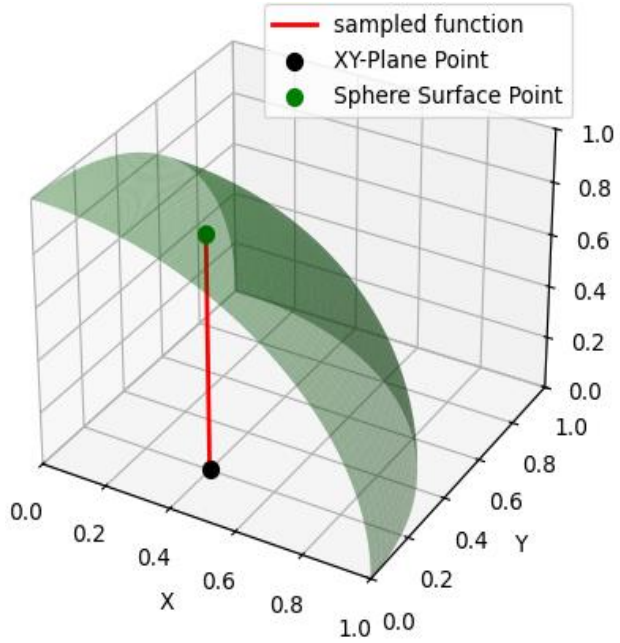

Figure 6. Illustration of random sampling of the first octant of a unit sphere function in three dimensions

Implementation of the random sampling of the function of a unit sphere in 7 dimensions is presented in Code 1, where the assignment of all coordinate variables was done directly without a loop, to prevent the introduction of index and array variables, in order to optimize the code to gain best execution performance possible.

```
01. // Computation loop
02. for (int i = 0; i < ITERATIONS; i++) {
03.     double x1 = curand_uniform(&rng); // Random x position in [0,1]
04.     double x2 = curand_uniform(&rng); // Random x position in [0,1]
05.     double x3 = curand_uniform(&rng); // Random x position in [0,1]
06.     double x4 = curand_uniform(&rng); // Random x position in [0,1]
07.     double x5 = curand_uniform(&rng); // Random x position in [0,1]
08.     double x6 = curand_uniform(&rng); // Random x position in [0,1]
09.     s2=x1*x1+x2*x2+x3*x3+x4*x4+x5*x5+x6*x6;
10.     if (s2 < 1.0) ssum[threadIdx.x] += sqrt(1.0 - s2);
11. }
```

Code 1. Random sampling of the function of a unit sphere's first orthant in 7 dimensions in CUDA C/C++

- *Volume Approximation by Random Point Sampling in the Unit Sphere*

For comparison, computation of the volume of the unit sphere's first orthant, $V_o = V_d/2^d$ is also performed by using random point sampling in the unit sphere. This method has been studied for simpler cases, for example, like in a two-dimensional circle [8] [9]. The algorithm woks by simply counting the number of random points inside the unit sphere's first orthant, divided by total number of random points generated in the *d*-dimensional unit cube of that particular orthant, that is:

$$V_o \approx \frac{n_S}{N}. \qquad (15)$$

For this computation, the greater the sample size *N*, the better, particularly when in higher dimension, as *d* increases. The random point sampling is illustrated in Figure 7. Implementation of the random point sampling inside the unit sphere's first orthant is presented in Code 2, where the assignment of all coordinate variables was done directly without a loop, to prevent the introduction of index and array variables, in order to optimize the code to gain best execution performance possible.

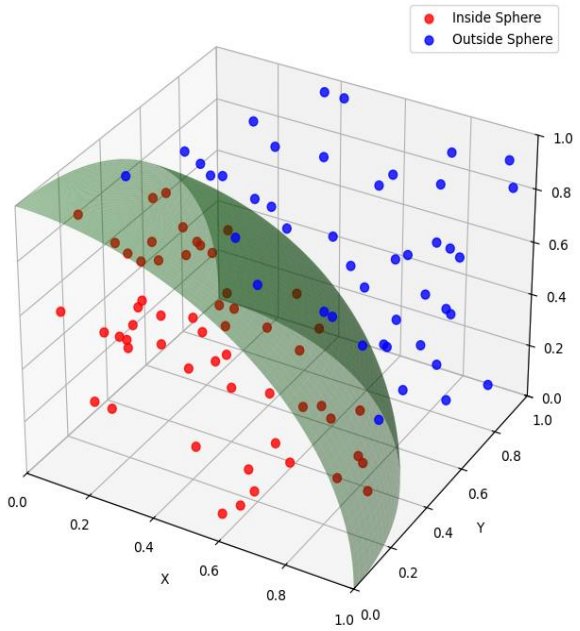

Figure 7. Illustration of random point sampling in a unit sphere's first orthant in three dimensions

```
// Computation loop
for (int i = 0; i < ITERATIONS; i++) {
    double x1 = curand_uniform(&rng); // Random x position in [0,1]
    double x2 = curand_uniform(&rng); // Random x position in [0,1]
    double x3 = curand_uniform(&rng); // Random x position in [0,1]
    double x4 = curand_uniform(&rng); // Random x position in [0,1]
    double x5 = curand_uniform(&rng); // Random x position in [0,1]
    double x6 = curand_uniform(&rng); // Random x position in [0,1]
    double x7 = curand_uniform(&rng); // Random x position in [0,1]
    if (x1*x1+x2*x2+x3*x3+x4*x4+x5*x5+x6*x6+x7*x7 < 1.0)
        counter[threadIdx.x]++;
}
```

Code 2. Random point sampling in a unit sphere's first orthant in 7 dimensions in CUDA C/C++

- *Hardware*

The computation was performed on a relatively fast hardware with CPU AMD Ryzen 9 5900X running at 4.8 GHz, 12 cores, 24 threads, equipped with 64 GB of DDR4 RAM and a GPGPU NVIDIA RTX3080Ti with 10240 computing cores, and 12 GB of GDDR5 VRAM. The computation code was developed in CUDA C/C++ and compiled in a Linux environment.

III. Results and Discussion

The computation is performed for 2, 3, 7, 15, and 24 dimensions. The reason behind this is that we want to perform the computation of a relatively high-dimensional calculations. Above 24 dimensions, starting $d = 25$ and more, the results were inaccurate due to data type limitation of the GPU capabilities, that is, the GPU's double precision data type is not enough to accurately calculate the result. Dimensions $d = 2$ and 3 were chosen simply because they are everyday geometry which can be checked in ease. While dimensions $d = 7$ and 15 were chosen because 7 and 15 are close to 8 and 16, respectively, which spans evenly between 1 and 24. Another reason dimension $d = 7$ and 15 were chosen because they are odd numbers, hence there are three odd dimensions (3, 7, 15) and two even dimensions (2, 24). In these computations, all of the accuracies, a_π, of the computed value of π were calculated according to equation (12).

A. Random Sampling Integration

For these computations, random sample size, $N = 3.2768 \times 10^{11}$ is chosen. This number is from 32 warp size × 10240 computing core × 1000000 iterations. The computations were performed 128 times, and the results were averaged.
The computation results which are approximate values of π, and their accuracies, are presented in Figure 8a and Figure 8b, respectively. Execution time of every computation is shown in Figure 9a, where it increases in a linear trend as seen in Figure 9b. Error bars on those figures are margin errors calculated from those 128 computation batches.

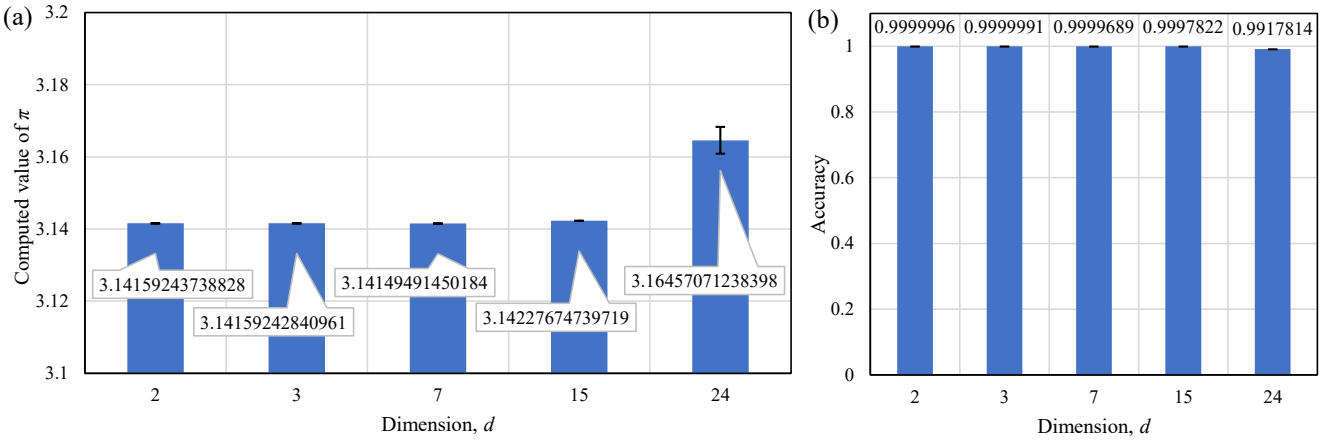

Figure 8. (a) Computed value of π, displayed in full machine double precision, averaged from batches. Please notice that the vertical axis is zoomed (3.1 to 3.2) to emphasize the small variations of the results. (b) Accuracy of the computed value of π, averaged from batches.

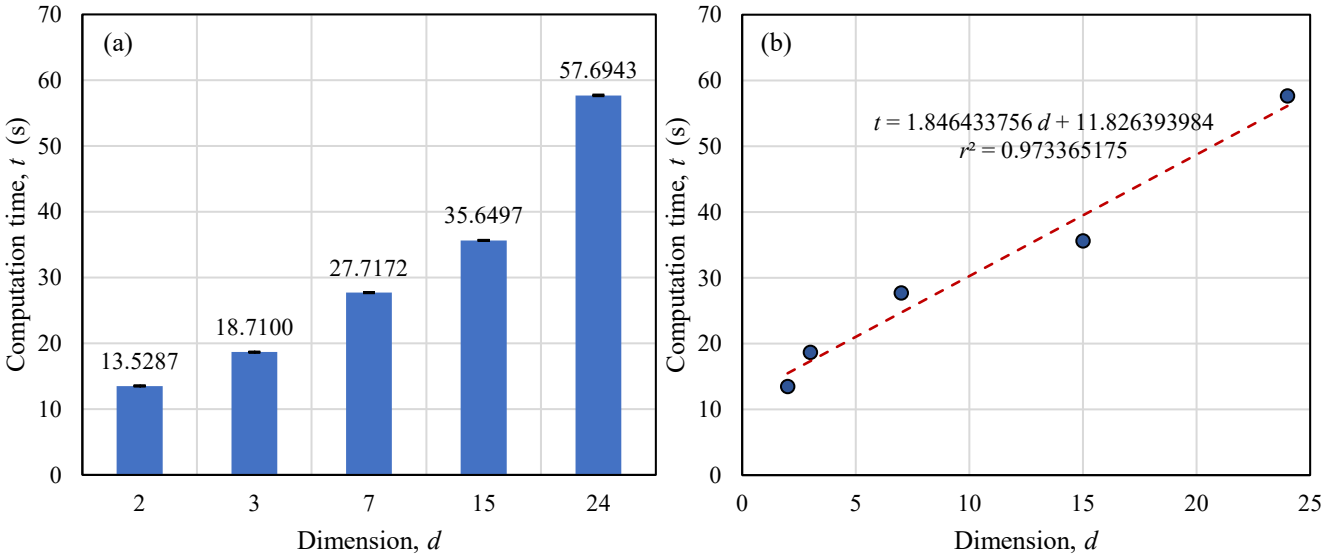

Figure 9. (a) Computation time in seconds, averaged from batches. (b) Linear relationship between computation time and number of dimensions.

p-ISSN : 2443-2210
e-ISSN : 2443-2229

B. Random Sampling Point Counting

For these computations, random sample size, $N = 3.2768 \times 10^{11}$ is chosen. This number is from 32 warp size × 10240 computing core × 1000000 iterations. The computations were performed 128 times, and the results were averaged.
The computation results which are approximate values of π, and their accuracies, are presented in Figure 10a and Figure 10b, respectively. Execution time of every computation is shown in Figure 11a, where it increases in a linear trend as seen in Figure 11b. Error bars on those figures are margin errors calculated from those 128 computation batches.

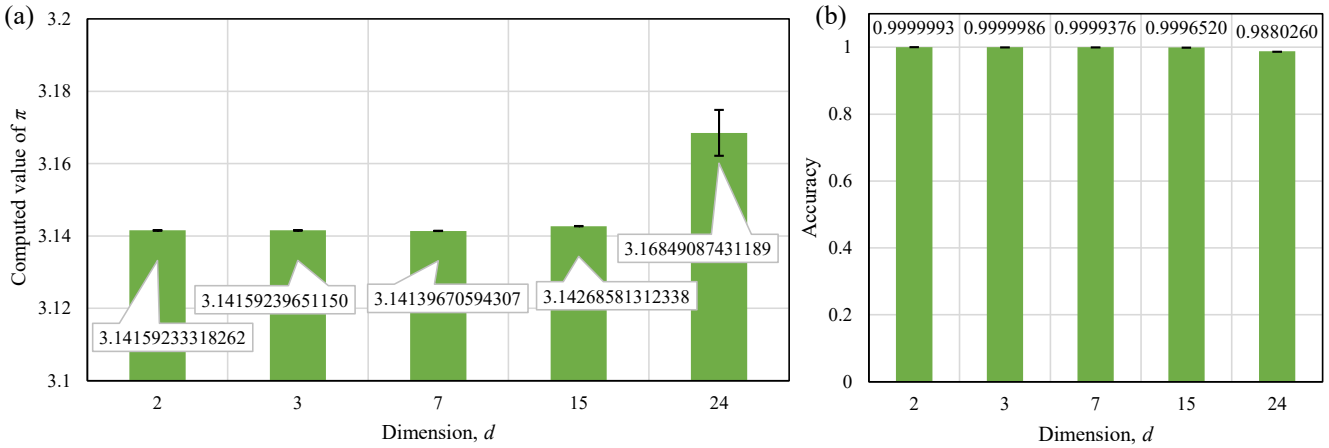

Figure 10. (a) Computed value of π, displayed in full machine double precision, averaged from batches. Please notice that the vertical axis is zoomed (3.1 to 3.2) to emphasize the small variations of the results. (b) Accuracy of the computed value of π, averaged from batches.

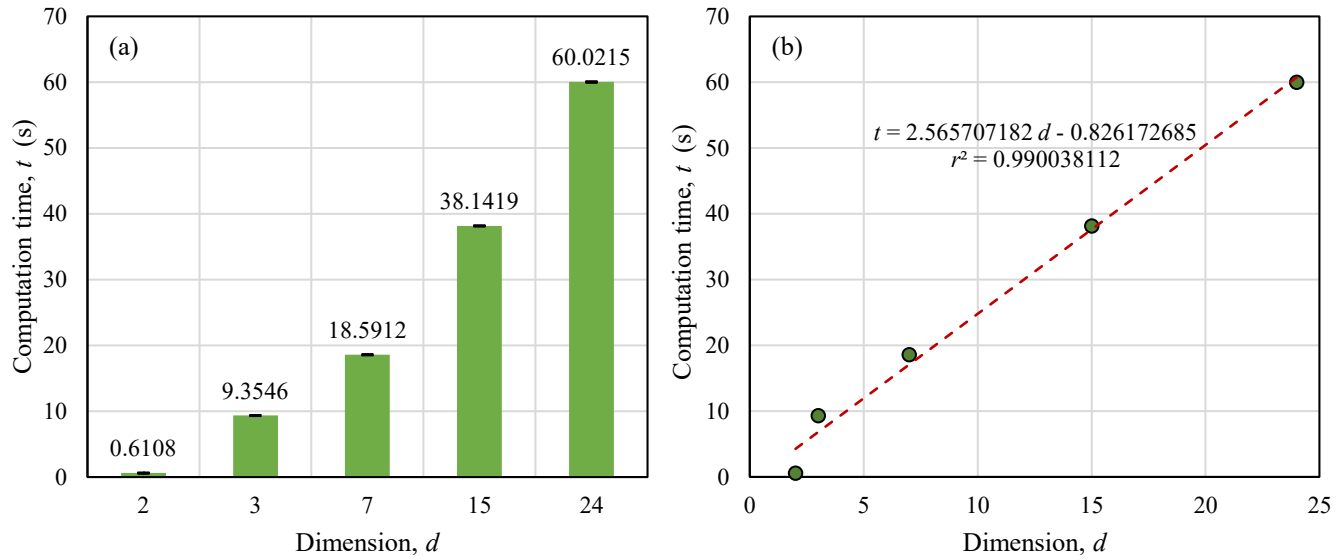

Figure 11. (a) Computation time in seconds, averaged from batches. (b) Linear relationship between computation time and number of dimensions.

C. Random Sampling Integration in 24-dimensional Space Using Various Sample Sizes.

Here the results of random sampling integration computation are presented using various sample sizes, from $N = 3.2768 \times 10^{9}$ up to $N = 3.2768 \times 10^{13}$.

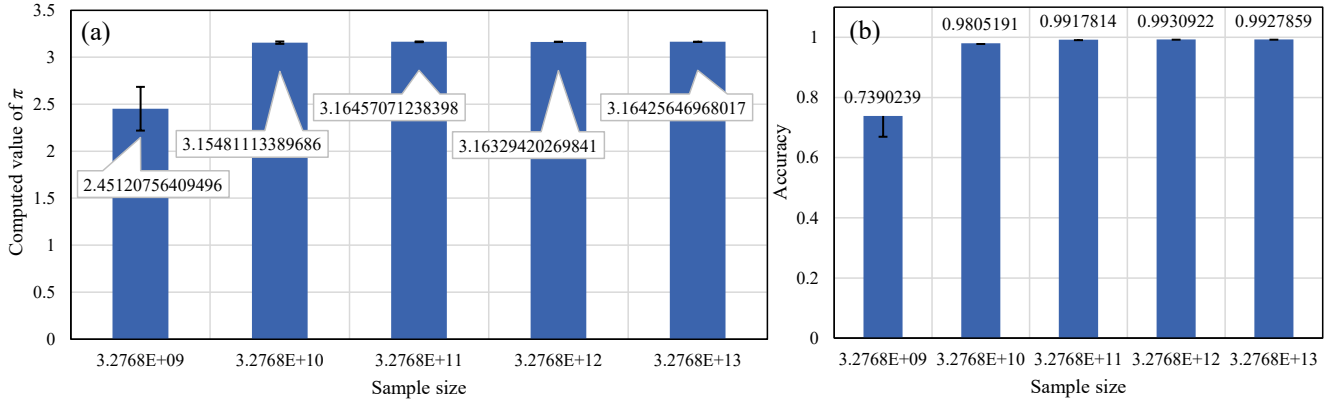

Figure 12 (a) Computed value of π, displayed in full machine double precision, averaged from batches. (b) Accuracy of the computed value of π, averaged from batches.

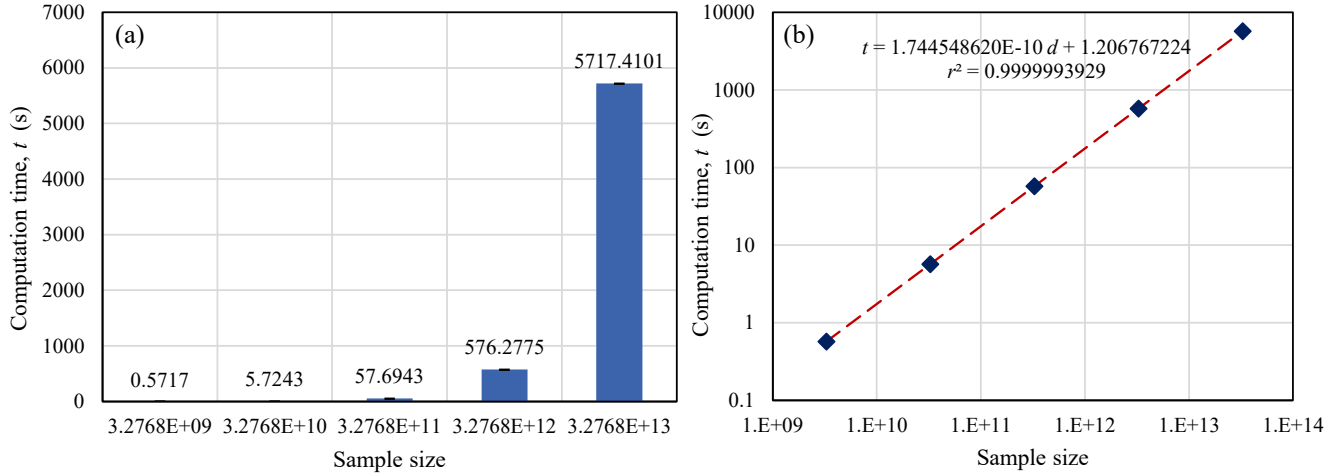

Figure 13 (a) Computation time in seconds, averaged from batches. (b) Linear relationship between computation time and sample size. Please notice that both axes use logarithmic scales.

D. Random Sampling Point Counting in 24-dimensional Space Using Various Sample Sizes.

Here the results of random sampling point counting computation are presented using various sample sizes, from $N = 3.2768 \times 10^9$ up to $N = 3.2768 \times 10^{13}$.

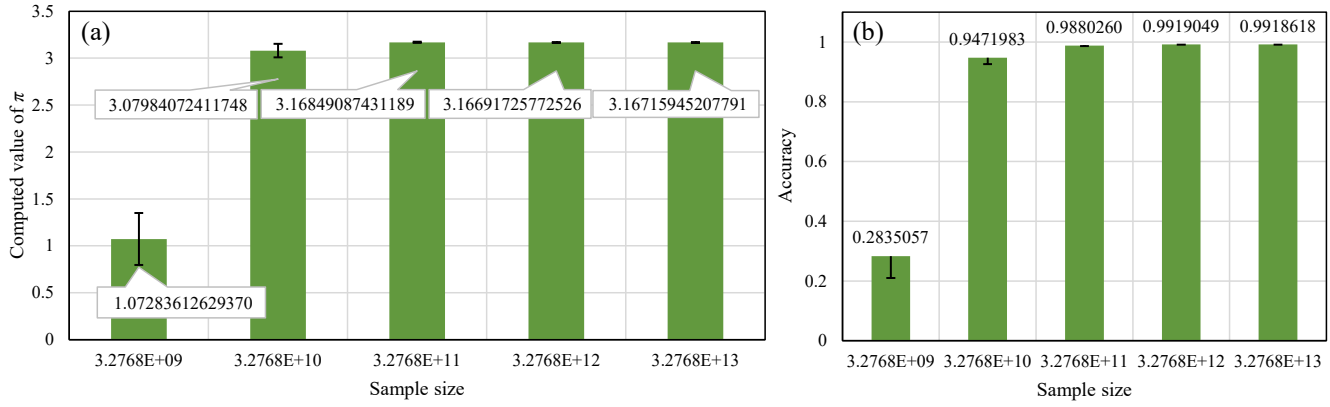

Figure 14 (a) Computed value of π, displayed in full machine double precision, averaged from batches. (b) Accuracy of the computed value of π, averaged from batches.

p-ISSN : 2443-2210
e-ISSN : 2443-2229

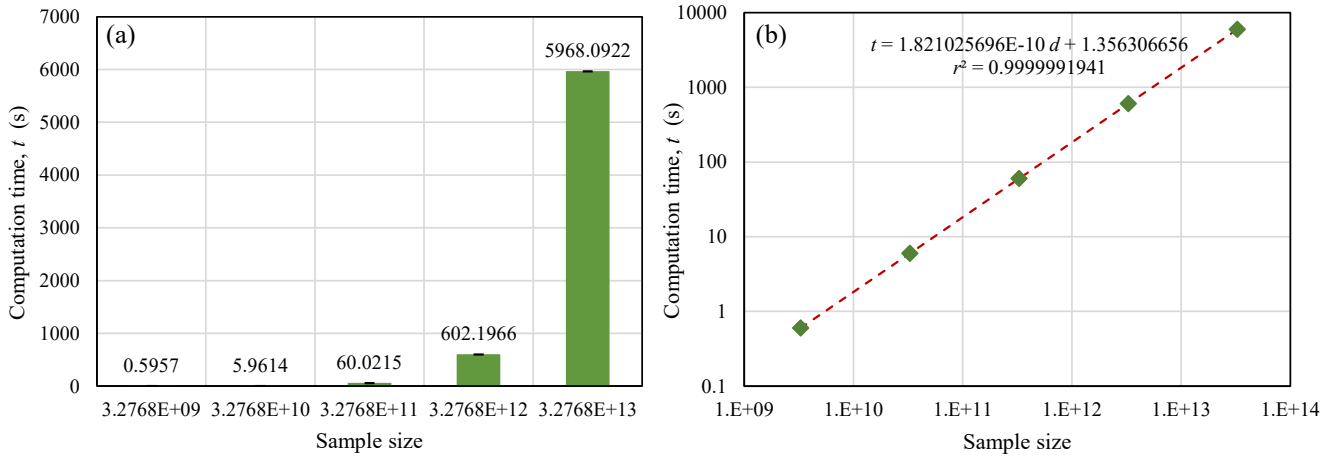

Figure 15 (a) Computation time in seconds, averaged from batches. (b) Linear relationship between computation time and sample size. Please notice that both axes use logarithmic scales.

E. Random Sampling Integration and Point Counting in 24-dimensional Space using CPU

For comparison, a single threaded CPU computation of π was also performed, in 24 dimensions, using sample size $N = 3.2768 \times 10^{10}$. The computations were performed in 128 batches, and the results were averaged. The results are displayed in Figure 16, where the random sampling integration method outperform the point counting method, that is, more accurate, slightly faster, and more stable (notice the much smaller error bars on random sampling compared to point counting method). Error bars on those figures are margin errors calculated from those 128 computation batches.

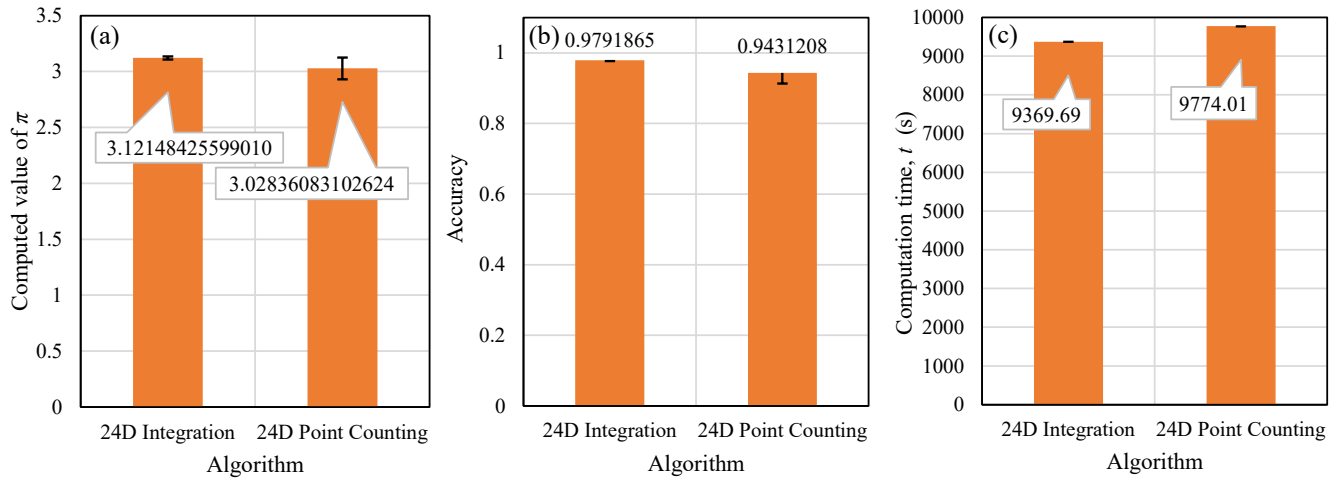

Figure 16. Computation of π using single threaded CPU: (a) Computed value of π, (b) accuracy, and (c) computation time.

F. General Discussion and Analysis

Here we examine what these results indicate for high-dimensional integration by random sampling in a GPU context. The objective was to determine whether this method functioned effectively when faced with the challenge of computing the volume of a unit sphere in *d*-dimensional space, extending up to 24 dimensions. The results were indeed quite revealing.

Initially, we would like to examine the accuracy of the results. Through the diverse dimensions explored, from the conventional two and three dimensions to the expansive seven, 15, or 24 dimensions, the random sampling method proved highly effective in estimating the volume within a sphere. These are spherical volumes, thereby serving as an effective standard. Previously, in the lower-dimensional realms of 2 or 3, we achieved a highly precise estimate of π. The calculated value corresponded with the analytically derived result to exceed double precision machine limits. Subsequently, circumstances began to deteriorate slightly. In 7 dimensions, the mean π from all our measurements approximated 3.14159 with little variations. However, up to 15 dimensions, the results grew progressively dispersed. The selected accuracy parameter, specifically the relative error formula, demonstrated a decline, from approximately 10^{-15} in lower dimensions to

roughly 10^{-10} or worse in 24 dimensions. As the dimensions expand, the volume of the orthant diminishes by several orders of magnitude. Essentially, this indicates that the computation requires far more samples to achieve equivalent precision, as elucidated by Monte Carlo theory.

The potential of the GPU is evident in this instance, achieving up to billions of points in certain runs, indicating a significant advancement. The durations observed were notably brief, frequently less than one second for low dimensions, and generally increased linearly with both dimension *d* and sample size *N*. Figure 9b and Figure 11b illustrate linear relationship between computation time and dimensions, with correlation r^2 close to 1, indicating that GPU parallelization effectively manages the additional coordinates, not to mention obvious linear relationship between computation time and sample size (Figure 13b and Figure 15b), also with correlation r^2 close to 1. The addition of each dimension results in an extra operations within the kernel; however, due to the parallel operation of the cores, the computational load is lessened compared to that on a CPU as the baseline (see Figure 16). We observed time intervals increase from milliseconds in two dimensions to several seconds in 24 dimensions for the same sample size.

Trade-offs were apparent in the comparison between random sampling integration and the point counting method. The concept of point counting is more straightforward than the sampling integration distribution; it exemplifies a Monte Carlo hit-or-miss method, as indicated by its name. In our experiments, point-counting yielded satisfactory results at lower dimensions; however, its precision was somewhat inferior to that of random sampling integration at higher dimensions. In a 24-dimensional space, for example, the point counting method yielded π estimates that exhibited less accuracy, particularly when sample size is relatively small, as we can compare Figure 12b versus Figure 14b. The rationale behind this is the following: Point counting is a binary process; it merely assesses whether the sum of squares is less than 1. This method is computationally efficient but slightly less accurate to represent the function's shape compared to the averaging of the square root in the random sampling approach.

Those samples integrate the actual height function $\sqrt{1-x_1^2-x_2^2-\cdots-x_{d-1}^2}$, it was much better fitted to the approximation. But on the whole, point counting was slightly slower in execution time. This is because although it didn't have to do a square root operation, which is computationally more expensive on a GPU (also on CPU), it needs to pick random samples on *d* dimensions, while random sampling integration picks samples on (d – 1) dimensions (its hyperplane domain). But it didn't make much difference, as we're talking fractions of a second per single sample calculation here, so for most practical purposes where accuracy is better achieved by random sampling integration than point counting.

In a Monte Carlo simulation, one would expect the error to go down like $1/\sqrt{N}$ as it fits our results quite well. For the 24 dimensions case with varying sample size *N* (see Figure 12 and Figure 13), as *N* increases, the computed π stabilized closer to the true value, and the accuracy improved. But as dimension *d* increases to 25, because of double-precision floating-point limits, one can't squeeze more digits out without higher precision arithmetic. That's why the computation hit a wall at 25 dimensions, that is, the volumes $V_o = V_d/2^d$ get so small, in the order of 10^{-11}, hence the sample size must be much greater than 10^{11}, and the $\pi = (32382376266240000\ V_o)^{1/12}$ computation (see Table 1) by the GPU's double precision just can't represent it accurately. It's a hardware limitation, not a flaw in the method.

G. Computation using GPU and CPU comparison

On a single-threaded computation using CPU (AMD Ryzen 9 5900X), the same code of both random sampling integration and point counting algorithms took much longer, even several order of magnitude longer for the same large sample size, as compared to parallel computation using GPU, as shown on Figure 13a, Figure 15a, and Figure 16c. Therefore, this method is most suited to large problems with massive parallelism. Again, as explained above, here the random sampling integration method outperformed the point counting method by a small amount (see Figure 16).

H. Random Sampling Integration Method Consideration versus Traditional Riemann Sum Integration

As we mentioned earlier in the introduction, for multidimensional integration, random sampling integration method is more suitable than traditional Riemann sum integration, as for *d*-dimensional space, random sampling integration with sample size N_S has complexity of $O((d-1)N_S)$, compare to $O((d-1)N_P^{d-1})$ for traditional Riemann sum integration with N_P sampling points on each dimension. Therefore, asymptotically for larger dimensions, the former method is still feasible whereas the latter grows exponentially with dimension, *d*, making it not feasible. Here is the limit of their complexity ratio:

$$\lim_{d\to\infty} \frac{(d-1)N_S}{(d-1)N_P^{d-1}} = \lim_{d\to\infty} \frac{N_S}{N_P^{d-1}} = 0. \qquad (16)$$

Equation (16) shows that the limit approaches zero as *d* approaches infinity, showing the superiority of the random sampling integration technique.

IV. CONCLUSION

We have shown that using random sampling in conjunction with high-performance computing using GPU is particularly attractive when dealing with those problematic multidimensional integrals which traditional methods find not feasible. By taking the example of volumes of spheres in different dimensions as our test case for this random sampling integration method, we found that it produces results of high accuracy at least up to dimension 24, with computation times within reason for parallel processing compared to longer computation time on the CPU as baseline (see Figure 16). This is the definitive answer to our research question. It might be interesting to look beyond 24 dimensions in the future or modify the sampling for even more complex functions, but for now one thing is clear: combining randomness with the power of high-performance hardware can fundamentally tackle multidimensional numerical integration problems.

On a larger scale random sampling integration on GPU's results suggest that they're powerful tools to solve high-dimensional problems which simple deterministic methods like Riemann sums couldn't ever reach. As the sphere example demonstrates, with the symmetry and computability like this, it only represents a certain type of integral in physics or statistics: those where there are radial dependencies and constraints. In quantum path integrals, for example, one is often integrating high-dimensional configuration spaces [10] [29] [30]. Similarly, in finance pricing options with many variables may benefit greatly from its precision at broad scales [12]. Of course, there are caveats. Random number generation by GPU isn't perfect: we used the pseudorandom number generator CURAND for our work, plus in very high dimensions correlations may crop up if proper seeding isn't done. We ameliorated those by using timer as random number seeds, and performed multiple batches and averaging them together. Also, there is the power drain, that is, GPUs consumes high electricity energy, so for these large jobs that's one factor. And though we did well up to 24 dimensions, actual integrals may have odd domains or functions which oscillate heavily. Techniques for variance reduction, such as importance sampling, could be added on top to solve this problem.

ACKNOWLEDGEMENT

This research is conducted under Skema B research project of Lembaga Penelitian dan Pengabdian kepada Masyarakat (LPPM), Universitas Kristen Maranatha. The funding and administrative supports from LPPM and the Faculty of Smart Technology and Engineering (FTRC), Universitas Kristen Maranatha are fully acknowledged.

REFERENCES

[1] J. Stewart, D. Clegg and S. Watson, Calculus, 9th Edition, Boston, Massachusetts: Cengage Learning, 2020.

[2] R. E. Bellman, Dynamic Programming, Courier Dover Corporation, 2003.

[3] C. M. Bishop, "Section 1.4: The Curse of Dimensionality," in *Pattern Recognition and Machine Learning*, Berlin, Springer, 2006.

[4] T. Hastie, R. Tibshirani and J. Friedman, "Section 2.5: The Curse of Dimensionality," in *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second edition*, Berlin, Springer, 2009.

[5] R. E. Caflisch, "Monte Carlo and quasi-Monte Carlo methods," *Acta Numerica,* vol. 7, pp. 1-49, 1998.

[6] M. Sugiyama, "Chapter 19 - Numerical Approximation of Predictive Distribution," in *Introduction to Statistical Machine Learning*, M. Sugiyama, Ed., Burlington, Massachusetts, Morgan Kaufmann, 2016, pp. 205-220.

[7] J. Stewart, D. Clegg and S. Watson, Multivariable Calculus, 9th Edition, Boston, Massachusetts: Cengage Learning, 2020.

[8] K. Achilles, "Monte Carlo Pi," in *BASIC und Pascal im Vergleich. Vieweg Programmbibliothek Mikrocomputer*, vol. 3, Springer Fachmedien Wiesbaden, 1983.

[9] R. Sharma, P. Singhal and M. K. Agrawal, "Application of Monte-Carlo Simulations in Estimation of Pi," in *IOP Conference Series Materials Science and Engineering*, 2021.

[10] R. P. Feynman and A. R. Hibbs, Quantum Mechanics and Path Integrals, Emended Edition, Mineola, New York: Dover Publications, 2010, pp. 29-31.

[11] N. Metropoli, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics,* vol. 21, pp. 1087-1092, 1953.

[12] P. Glasserman, Monte Carlo Methods in Financial Engineering, Berlin: Springer Science & Business Media, 2004.

[13] C. Andrieu, N. de Freitas, A. Doucet and M. Jordan, "An introduction to MCMC for machine learning," *Machine Learning,* vol. 50, no. 1, pp. 5-43, 2003.

[14] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana and S. Tarantola, Global Sensitivity Analysis The Primer, Chichester: Wiley, 2008.

[15] A. I. Naimi, D. Benkeser and J. E. Rudolph, "Monte Carlo Integration in Simple and Complex Simulation Designs," *arXiv:2406.15285v2,* 2024.

[16] S. Roman, Advanced Linear Algebra, 2nd Edition, New York: Springer, 2005.

[17] E. W. Weisstein, "Hyperoctant," Wolfram, [Online]. Available: https://mathworld.wolfram.com/Hyperoctant.html. [Accessed 1 Jan 2025].

[18] D. S. Balsara and M. Dumbser, "Multidimensional Riemann problem with self-similar internal structure. Part II – Application to hyperbolic conservation laws on unstructured meshes," *Journal of Computational Physics,* vol. 287, pp. 269-292, 15 April 2015.

[19] R. Martínez-Planell and M. Trigueros, "Students' understanding of Riemann sums for integrals of functions of two variables," *The Journal of Mathematical Behavior,* vol. 59, Sep 2020.

[20] K. A. Schneider, J. M. Gallardo, D. S. Balsara, B. Nkonga and C. Parés, "Multidimensional approximate Riemann solvers for hyperbolic nonconservative systems. Applications to shallow water systems," *Journal of Computational Physics,* vol. 444, p. 110547, 1 November 2021.

[21] D. J. Smith and M. K. Vamanamurthy, "How Small Is a Unit Ball?," *Mathematics Magazine,* vol. 62, no. 2, p. 101–107, 1989.

[22] B. Hayes, "An Adventure in the Nth Dimension," Sigma Xi, The Scientific Research Honor Society, 2025. [Online]. Available: https://www.americanscientist.org/article/an-adventure-in-the-nth-dimension. [Accessed 1 May 2025].

[23] NIST, "NIST Digital Library of Mathematical Functions," National Institute of Standards and Technology, 2001. [Online]. Available: https://dlmf.nist.gov/5.19#E4. [Accessed 1 Jan 2025].

[24] H. R. Parks, "The volume of the unit n-ball," *Mathematics Magazine,* vol. 86, no. 4, p. 270–274, 2013.

[25] J. Gipple, "The volume of n-balls," *Rose-Hulman Undergraduate Mathematics Journal,* vol. 15, no. 1, p. 14 (online article), 2014.

[26] D. H. Bailey, S. M. Plouffe, P. B. Borwein and J. M. Borwein, "The quest for PI," *The Mathematical Intelligencer,* vol. 19, p. 50–56, 01 December 1997.

[27] R. P. Agarwal, H. Agarwal and S. K. Sen, "Birth, growth and computation of pi to ten trillion digits," *Advances in Difference Equations,* vol. 100 (2013), 2013.

[28] A. J. Yee, "Pi Record Smashed at 202 Trillion Digits," NumberWorld.org, 28 June 2024. [Online]. Available: https://www.numberworld.org/y-cruncher/news/2024.html#2024_6_28. [Accessed 1 Jan 2025].

[29] S. A. Chin, "High-order path-integral Monte Carlo methods for solving quantum dot problems," *Physical Review E,* vol. 91, no. 3, p. 031301, 2015.

[30] H. A. Camargo, P. Caputa and P. Nandy , "Q-curvature and path integral complexity," *Journal of High Energy Physics,* vol. 2022, no. 81, 2022.