Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Учебный курс

“Суперкомпьютерное моделирование и технологии”

Численное Интегрирование Многомерных Функций Методом Монте-Карло

ОТЧЕТ

о выполненном задании

студента 621 учебной группы факультета ВМК МГУ

Морквина Андрея Андреевича

Вариант 3

Москва, 2022

Оглавление

1. Постановка задачи

В качестве задания по курсу предлагается рассмотреть задачу вычисления многомерного интеграла методом Монте-Карло. В задании необходимо:

1. Выполнить программную реализацию на языке C/C++ с использованием библиотеки параллельного программирования MPI.
2. Исследовать масштабируемость созданной программы на вычислительных системах ВМК МГУ (IBM Polus).

Математически задача формулируется следующим образом. Дана функция $f(x, y, z)$ – непрерывная в ограниченной и замкнутой области $G \subset R^3$. Требуется вычислить определенный интеграл:

$$I = \iiint\limits_G f(x, y, z)\, dx\, dy\, dz = \iiint\limits_G \sqrt{x^2 + y^2}\, dx\, dy\, dz$$

, где G ограничена поверхностями $x^2 + y^2 = z^2, z = 1$.

2. Численный метод решения задачи

Пусть область G ограниченна параллелепипедом П: $\begin{cases} a_1 \le x \le b_1 \\ a_2 \le y \le b_2 \\ a_3 \le z \le b_3 \end{cases}$

Рассмотрим функцию $F(x, y, z) = \begin{cases} f(x, y, z), & (x, y, z) \in G \\ 0, & (x, y, z) \notin G \end{cases}$

Тогда:

$$I = \iiint\limits_G f(x, y, z)\, dx\, dy\, dz = \iiint\limits_{\Pi} F(x, y, z)\, dx\, dy\, dz$$

Пусть $p_1(x, y, z)$ ... $p_n(x, y, z)$ – случайные точки, равномерно распределенные в П. Тогда в качестве приближенного значения можно использовать выражение:

$$I \approx \frac{(b_1 - a_1)(b_2 - a_2)(b_3 - a_3)}{n} \sum_{i=1}^{n} F(p_i)$$

3. Аналитическое решение

Для вычисления интеграла воспользуемся цилиндрической системой координат (область G можно увидеть на рисунке 1):

$$\iiint\limits_G \sqrt{x^2+y^2}\,dx\,dy\,dz = \begin{vmatrix} x = r\cos\varphi \\ y = r\sin\varphi \\ z = z \end{vmatrix} = \iiint\limits_{\substack{0\le z\le 1 \\ 0\le\varphi\le 2\pi \\ r=z}} r^2\,dr\,d\varphi\,dz =$$

$$= \int\limits_0^1 dz \int\limits_0^{2\pi} d\varphi \int\limits_0^z r^2\,dr = \frac{2\pi}{3}\int\limits_0^1 z^3 dz = \frac{\pi}{6}$$

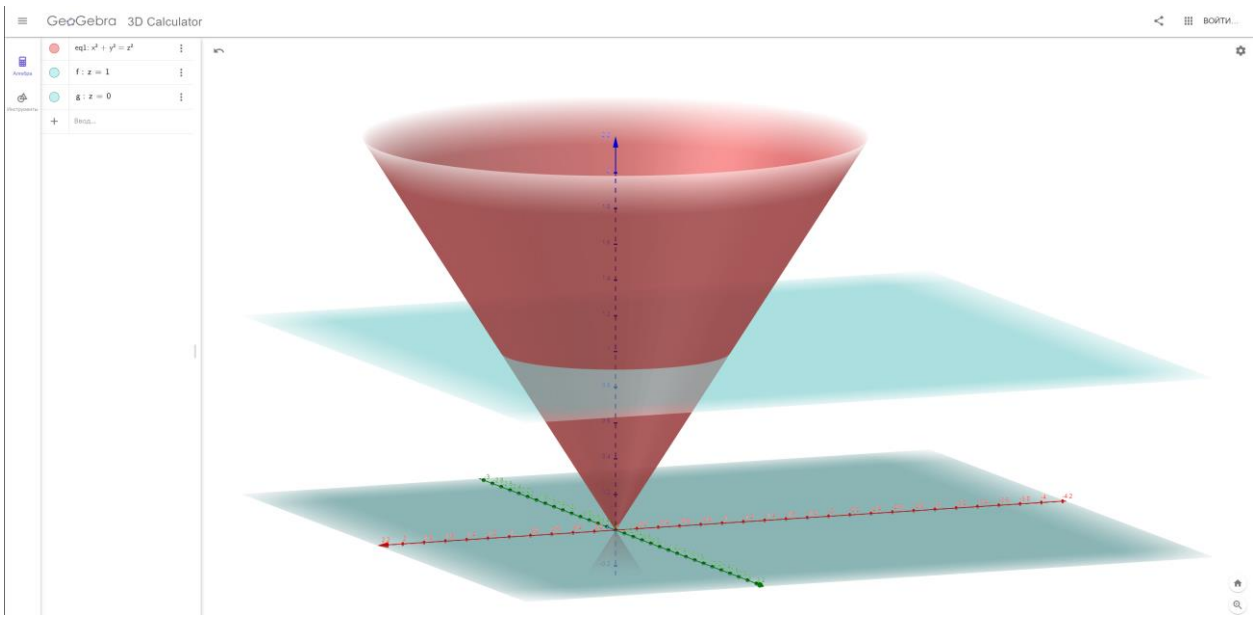

Рисунок 1. Область G.

4. Программная реализация

Вариант распараллеливания:

1. Параллельные процессы генерируют случайные точки независимо друг от друга.

Идея программной реализации:

Программа принимает на вход требуемую точность. Затем параллельные процессы генерируют случайные точки блоками по 10000 независимо друга от друга. После чего результаты обрабатываются на управляющем процессе и принимается решение о продолжении или остановке вычислений.

Листинг программы:

```cpp
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <random>

#include "mpi.h"

double fun() {
    double x = 2.0 * ((float) rand() / RAND_MAX) - 1.0;
    double y = 2.0 * ((float) rand() / RAND_MAX) - 1.0;
    double z = 1.0 * ((float) rand() / RAND_MAX);
    double value = 0;
    if (z >= sqrt(x*x + y*y))
        value = sqrt(x * x + y * y);
    return value;
}

int main(int argc, char **argv) {
    double eps = atof(argv[1]);
    const double solution = M_PI / 6;

    MPI_Init(&argc, &argv);
    const double startTime = MPI_Wtime();

    int wrank, wsize;
    MPI_Status status;
    MPI_Comm_rank(MPI_COMM_WORLD, &wrank);
    MPI_Comm_size(MPI_COMM_WORLD, &wsize);
    srand(wrank + wsize + 50);

    unsigned long long gNumPoints = 0;
    double gSum = 0.0;
    double gTime = 0.0;
    double gSolution = 0;

    int flag = 1;
    unsigned long long lNumPoints = 1;
    while (flag) {
        //std::cout << flag << " " << gSum << " " << gNumPoints << " " << gSolution << std::endl;
        double lSum = 0.0;
        lNumPoints = 10000;
        for (unsigned i = 0; i < lNumPoints; i++)
            lSum += fun();

        double tmp1 = 0;
        unsigned long long tmp2 = 0;
        MPI_Reduce(&lSum, &tmp1, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
        MPI_Reduce(&lNumPoints, &tmp2, 1, MPI_UNSIGNED_LONG, MPI_SUM, 0, MPI_COMM_WORLD);
        gSum += tmp1;
        gNumPoints += tmp2;
        if (wrank == 0) {
            gSolution = 4 * gSum / gNumPoints;
            if (fabs(gSolution - solution) < eps)
                flag = 0;
        }
        MPI_Bcast(&flag, 1, MPI_INT, 0, MPI_COMM_WORLD);
    }

    const double lTime = MPI_Wtime() - startTime;
    MPI_Reduce(&lTime, &gTime, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    if (wrank == 0) {
        std::cout << "Ans: " << gSolution << std::endl << "Error: " << fabs(gSolution - solution) << std::endl
                  << "N: " << gNumPoints << std::endl << "Time: " << gTime << std::endl;
    }
    MPI_Finalize();
    return 0;
}

```

5. Исследование масштабируемости программы

Необходимо провести запуски программы на системе Polus для различного числа MPI-процессов и различных значений входного параметра точности ε.

Требуется построить графики зависимости ускорения программы от числа используемых MPI-процессов для каждого значения ε.

Под ускорением программы, запущенной на p MPI-процессах, понимается величина:

$$S_p = T_1/T_p$$

где T_1 – время работы на 1 MPI-процессе, T_p – время работы программы на p MPI-процессах.

Результаты исследования на системе Polus:

ε	MPI-процессы	Время работы (с)	Ускорение	Ошибка
$3 * 10^{-5}$	1	0.04	1	$1.9 * 10^{-5}$
	4	0.04	1.1	$4 * 10^{-6}$
	16	0.02	2	$1.2 * 10^{-5}$
	60	0.04	1.3	$8 * 10^{-6}$
$5 * 10^{-6}$	1	6.3	1	$4 * 10^{-6}$
	4	0.03	210	$4 * 10^{-6}$
	16	0.12	53	$2.5 * 10^{-6}$
	60	0.23	27	$3.5 * 10^{-6}$
$1.5 * 10^{-6}$	1	6.4	1	$1.4 * 10^{-6}$
	4	0.04	160	$7 * 10^{-7}$
	16	0.72	9	$8 * 10^{-8}$
	60	0.27	24	$1.1 * 10^{-6}$