Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчет по заданию «Численное интегрирование многомерных функций методом Монте-Карло»

в рамках курса «Суперкомпьютерно моделирование и технологии»

Вариант 4

Выполнила: Цянь Чэнсыцзинь, 614 группа

Москва
2022

Содержание

1 Математическая постановка задачи

Функция $f(x, y, z) = e^{x^2+y^2}z$ – непрерывна в ограниченной замкнутой области $G \subset \mathbb{R}^3$. Требуется вычислить определённый интеграл:

$$I = \iiint\limits_G e^{x^2+y^2}z \; dxdydz$$

где область $G = (x, y, z) : z \geq 0, x^2 + y^2 + z^2 \leq 1$.

2 Численный метод решения задачи

Используется метод Монте-Карло для численного интегрирования.

Пусть область G ограничена параллелепипедом: П: $\begin{cases} -1.0 \leq x \leq 1.0 \\ -1.0 \leq y \leq 1.0 \\ 0 \leq z \leq 1.0 \end{cases}$

Рассмотрим функцию: $F(x, y, z) = \begin{cases} e^{x^2+y^2}z, & (x, y, z) \in G \\ 0, & (x, y, z) \notin G \end{cases}$

Преобразуем искомый интеграл:

$$I = \iiint\limits_G e^{x^2+y^2}z \; dxdydz = \iiint\limits_{\text{П}} F(x, y, z)dxdydz$$

Пуст $p_1(x_1, y_1, z_1), p_2(x_2, y_2, z_2)$ – случайные точки, равномерно распределённые в П. Возьмём n таких случайных точек. В качестве приближённого значения интеграла предлагается использовать выражение:

$$I \approx |\text{П}| \cdot \frac{1}{n} \sum_{i=1}^{n} F(p_i)$$

где $|\text{П}|$ – объём параллелепипеда П. $|\text{П}| = 4$

3 Аналитическое решение задачи

Найдём точное значение интеграла аналитически:

$$I = \iiint\limits_G e^{x^2+y^2}z \; dxdydz$$

где область $G = (x, y, z) : z \geq 0, x^2 + y^2 + z^2 \leq 1$.

$$I = \iint\limits_{x^2+y^2\leq 1} e^{x^2+y^2}\Big(\int_0^{\sqrt{1-x^2-y^2}} z\ dz\Big)\ dxdy = \iint\limits_{x^2+y^2\leq 1} \Big(\frac{1}{2}z^2\Big|_0^{\sqrt{1-x^2-y^2}}\Big)\cdot e^{x^2+y^2}\ dxdy$$
$$= \frac{1}{2}\iint\limits_{x^2+y^2\leq 1} (1-x^2-y^2)\cdot e^{x^2+y^2}\ dxdy = \{\text{полярная система координат}\}$$
$$= \frac{1}{2}\int_0^{2\pi} d\theta \int_0^1 (1-r^2)e^{r^2}\cdot r dr = \pi\int_0^1 (1-r^2)e^{r^2}\cdot r dr = \{\text{замена } r^2 \text{ на } c\}$$
$$= \pi\int_0^1 (1-c)e^c\cdot\frac{1}{2}\ dc = \frac{1}{2}\pi\int_0^1 (1-c)\ d(e^c) = \{\text{по частям}\}$$
$$= \frac{1}{2}\pi\Big((1-c)e^c\Big|_0^1 - \int_0^1 -e^c\ dc\Big) = \frac{1}{2}\pi\Big(-1+e^c\big|_0^1\Big) = \frac{1}{2}\pi(-1+e-1) = \frac{1}{2}\pi(e-2)$$

4 Краткое описание программной реализации

Требуется реализовать параллельную MPI-программу, которая принимает на вход требуемую точность и генерирует случайные точки до тех пор, пока требуемая точность не будет достигнута.

Программа считывает в качестве аргумента командной строки требуемую точность ε и выводит четыре числа:

- Просчитанное приближённое значение интеграла.
- Ошибка посчитанного значения: модуль разности между приближённым, полученным методом Монте-Карло, и точным значениями интеграла.
- Количество сгенерированных случайных точек.
- Время работы программы в секундах. (Каждый MPI-процесс измеряет своё время выполнения, затем среди полученных значений берётся максимум)

В моем варианте требуется, что разные параллельные процессы генерируют разные случайные последовательности точек независимо друг от друга (т.е. необходимо инициализировать генератор псевдослучайных чисел, в случае использования стандартного генератора – функцией `srand()`), вычисляют свою часть суммы, и затем вычисляется общая сумма с помощью операции редукции. После чего вычисляется ошибка (разность между посчитанным значением и точным значением, вычисленным аналитически). В случае если ошибка выше требуемой точности, которую подали на вход программе, то генерируются дополнительные точки и расчёт

продолжается.

Algorithm 1 Вычисление интеграла методом Монте-Карло

инициализировать заданной точность ε, сумма значений функции F в точках, которые попадут в область G `sum` = 0.0, точное значение интеграла `I`, полученное значение интеграла `integral` = 0.0 и текущая разница `current_gap` = $fabs(I - integral)$, `n_times`=1.
while current_gap > eps:
    генерировать 1000 случайных чисел (x, y, z)
    if $(x, y, z) \in G$:
        sum += $f(x, y, z)$
    integral = (объем G · sum) / (1000 · n_times)
    if `current_gap` = |I − integral| $\leq \varepsilon$:
        break from while
вывод: приближенное значение интеграла `integral`.

В приложении показан код параллельной программы.

5 Исследование масштабируемости программы

Проведём запуски программы на системах Polus для различного числа MPI-процессов и различных значений входного параметра ε в соответствии с таблицей 5.1. И построим графики зависимости ускорения программы от числа используемых MPI-процессов для каждого значения ε. Под ускорением программы, запущенной на p MPI-процессах, понимается величина:

$$S_p = \frac{T_1}{T_p}$$

где T_1 – время работы программы на 1 MPI-процессе, T_p – время работы программы на p MPI-процессах.
С учетом вероятностной природы метода Монте-Карло, я решила запустить программу 5 раз с разными значениями shift$(0, 10, 100, 1000, 10000)$ для каждого числа MPI-процессов и каждой точности, и потом вычислить их среднее значение, чтобы получить разные значения `seed` и зависимость ускорение от числа MPI-процессов. Наконец-то получила следующие таблицы и графики.

Сравнивая ускорение работы программы с разным количеством процессов при одинаковой точности, видно, что при увеличении числа MPI-процессов, скорость работы программы увеличивается.

Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	1	0.0675	1	$1.95 \cdot 10^{-5}$
	4	0.0223	3.03	$1.95 \cdot 10^{-5}$
	16	0.0139	7.18	$0.36 \cdot 10^{-5}$
$5.0 \cdot 10^{-6}$	1	0.0669	1	$3.31 \cdot 10^{-6}$
	4	0.0178	3.76	$3.63 \cdot 10^{-6}$
	16	0.0102	6.56	$3.63 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	1	0.1947	1	$0.99 \cdot 10^{-6}$
	4	0.0443	3.04	$1.12 \cdot 10^{-6}$
	16	0.0285	4.40	$1.04 \cdot 10^{-6}$

Рис. 5.1: Результаты расчётов для системы Polus при `shift` $= 0$

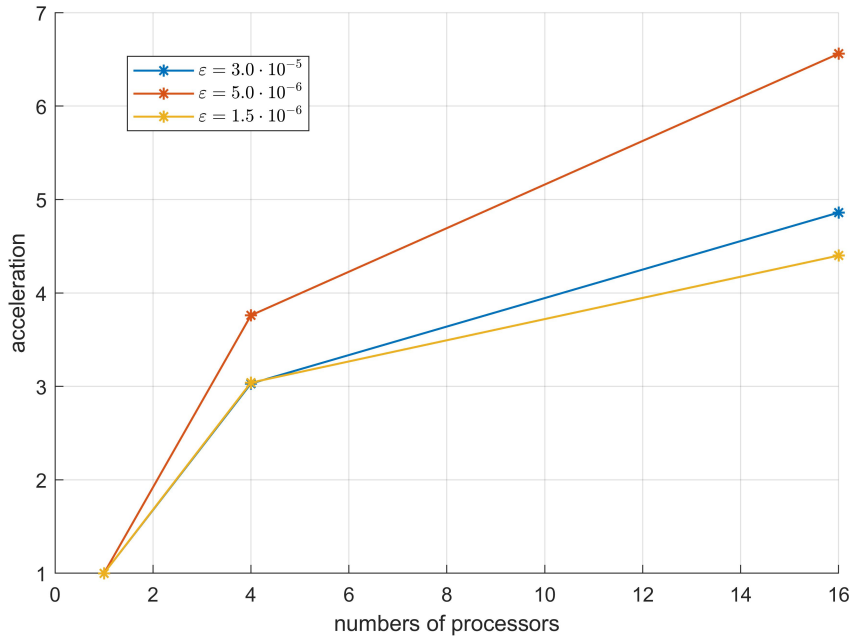

Рис. 5.2: Зависимость ускорения от числа MPI-процессов при `shift` $= 0$

Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	1	0.0717	1	$2.89 \cdot 10^{-5}$
	4	0.0253	2.83	$2.89 \cdot 10^{-5}$
	16	0.0142	5.04	$2.24 \cdot 10^{-5}$
$5.0 \cdot 10^{-6}$	1	0.0750	1	$1.47 \cdot 10^{-6}$
	4	0.0299	2.51	$4.20 \cdot 10^{-6}$
	16	0.0230	3.26	$4.47 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	1	0.0730	1	$1.47 \cdot 10^{-6}$
	4	0.0314	2.32	$0.91 \cdot 10^{-6}$
	16	0.0139	5.25	$0.95 \cdot 10^{-6}$

Рис. 5.3: Результаты расчётов для системы Polus при `shift` $= 10$

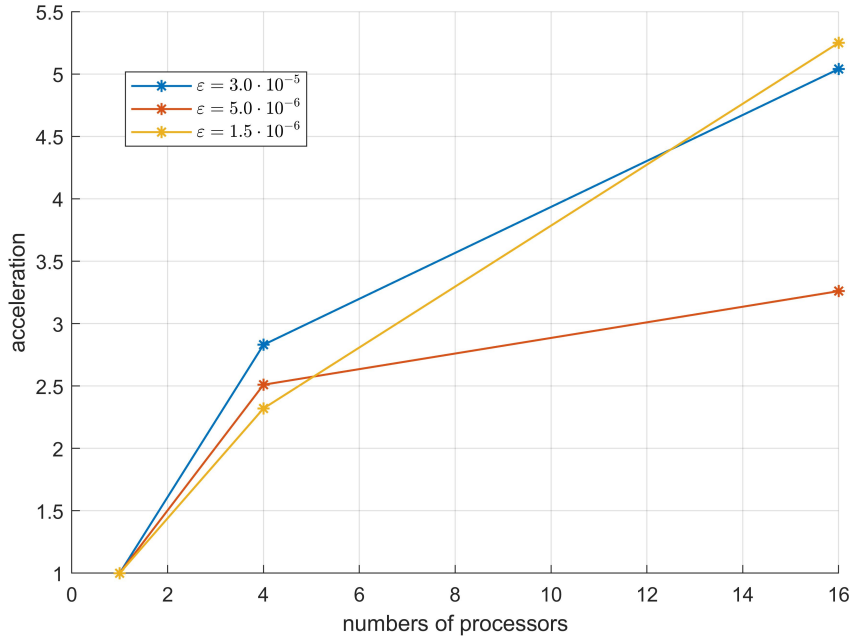

Рис. 5.4: Зависимость ускорения от числа MPI-процессов при `shift` = 10

Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	1	0.0203	1	$1.02 \cdot 10^{-5}$
	4	0.0178	1.14	$2.49 \cdot 10^{-5}$
	16	0.0119	1.71	$2.00 \cdot 10^{-5}$
$5.0 \cdot 10^{-6}$	1	0.1340	1	$1.56 \cdot 10^{-6}$
	4	0.0457	2.93	$1.56 \cdot 10^{-6}$
	16	0.0315	4.25	$015 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	1	0.1338	1	$0.21 \cdot 10^{-6}$
	4	0.0490	2.73	$0.21 \cdot 10^{-6}$
	16	0.0326	4.10	$0.15 \cdot 10^{-6}$

Рис. 5.5: Результаты расчётов для системы Polus при `shift` = 100

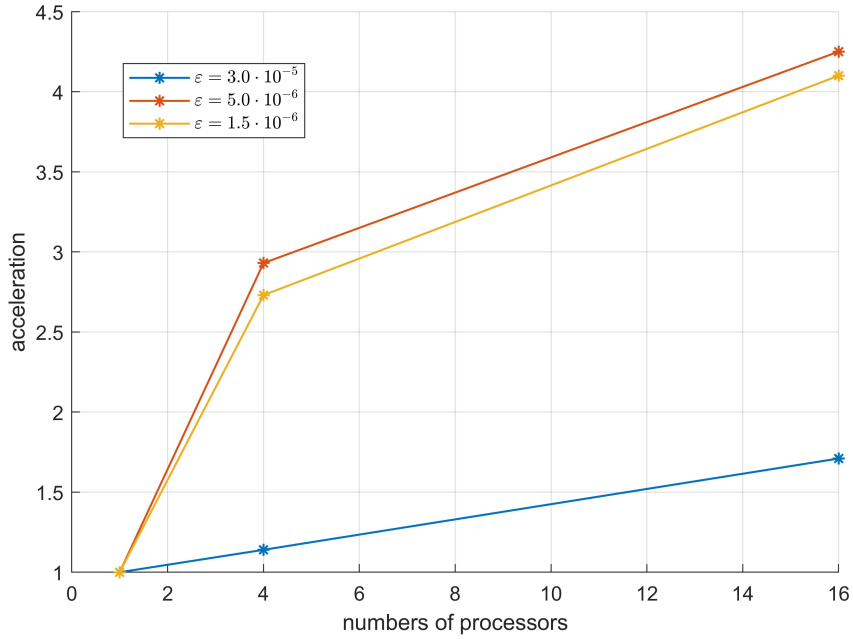

Рис. 5.6: Зависимость ускорения от числа MPI-процессов при `shift` = 100

Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	1	0.1000	1	$1.52 \cdot 10^{-5}$
	4	0.0380	2.63	$2.72 \cdot 10^{-5}$
	16	0.0107	9.34	$2.36 \cdot 10^{-5}$
$5.0 \cdot 10^{-6}$	1	0.1770	1	$1.62 \cdot 10^{-6}$
	4	0.0535	3.31	$1.48 \cdot 10^{-6}$
	16	0.0391	4.53	$1.48 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	1	0.2158	1	$0.34 \cdot 10^{-6}$
	4	0.0543	3.97	$1.48 \cdot 10^{-6}$
	16	0.0334	6.46	$1.48 \cdot 10^{-6}$

Рис. 5.7: Результаты расчётов для системы Polus при `shift` = 1000

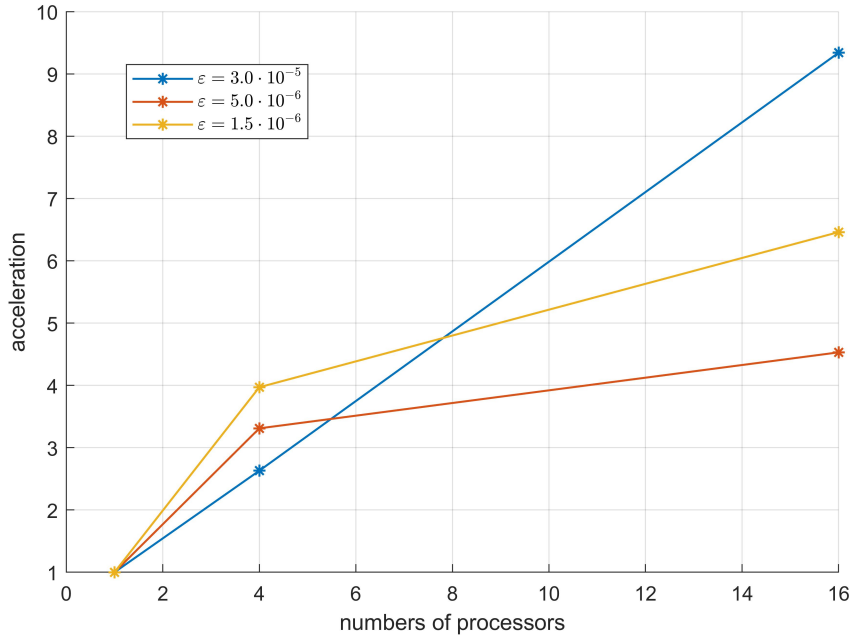

Рис. 5.8: Зависимость ускорения от числа MPI-процессов при `shift` = 1000

Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	1	0.0692	1	$2.63 \cdot 10^{-5}$
	4	0.0245	2.82	$2.63 \cdot 10^{-5}$
	16	0.0098	7.06	$1.92 \cdot 10^{-5}$
$5.0 \cdot 10^{-6}$	1	0.1669	1	$1.36 \cdot 10^{-6}$
	4	0.0762	2.19	$4.66 \cdot 10^{-6}$
	16	0.0293	5.69	$3.21 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	1	0.4073	1	$0.73 \cdot 10^{-6}$
	4	0.1298	3.13	$0.18 \cdot 10^{-6}$
	16	0.0830	4.90	$0.49 \cdot 10^{-6}$

Рис. 5.9: Результаты расчётов для системы Polus при `shift` = 10000

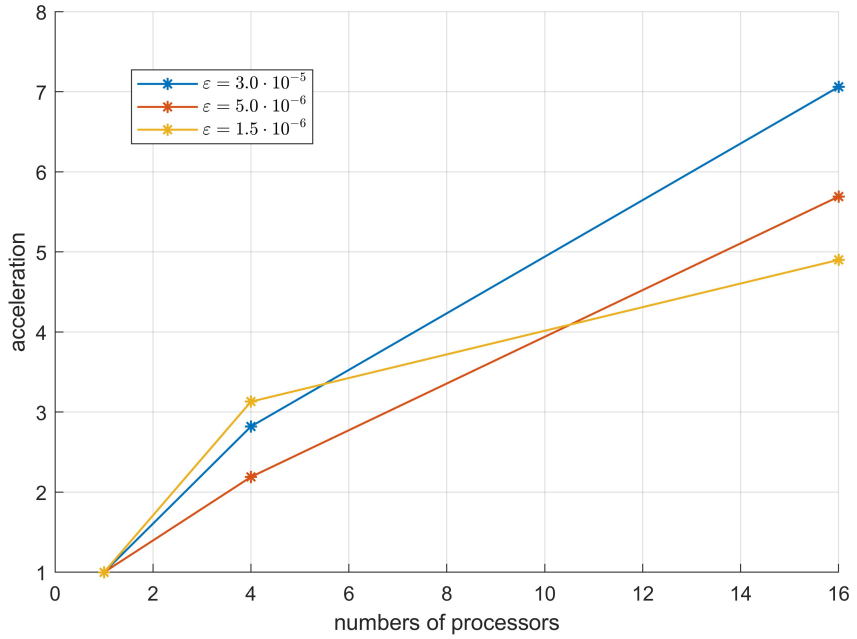

Рис. 5.10: Зависимость ускорения от числа MPI-процессов при `shift` = 10000

Точность ε	Число MPI-процессов	Ускорение (среднее)
$3.0 \cdot 10^{-5}$	1	1
	4	2.49
	16	6.07
$5.0 \cdot 10^{-6}$	1	1
	4	2.94
	16	4.86
$1.5 \cdot 10^{-6}$	1	1
	4	3.04
	16	5.02

Рис. 5.11: Среднее значение разультатов расчётов

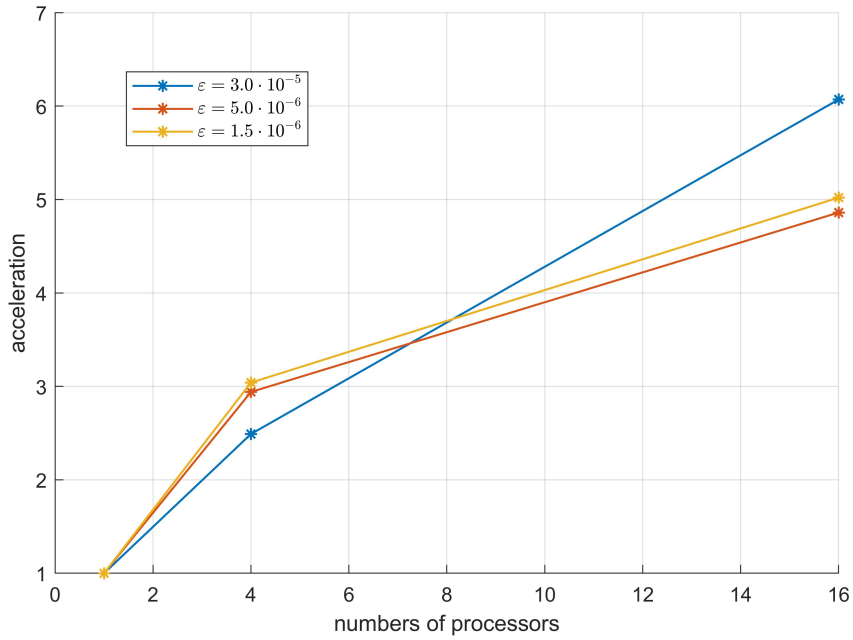

Рис. 5.12: Зависимость ускорения (среднее значение) от числа MPI-процессов

6 Приложение: Код программы

```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define real_integral M_PI * (M_E - 2.0) / 2.0 // analytical value of the integral

double f(double x, double y, double z){
    return z * exp(x * x + y * y);
}

double generate_point(double left_range, double right_range){
    return ((double)rand() / RAND_MAX) * (right_range - left_range) + left_range; //generate point between [left_range, right_range]
}

int main(int argc, char *argv[]){

    int n_points = atoi(argv[1]); // Each process adds n_points random points at a time
    double eps = atof(argv[2]);
    int shift = atoi(argv[3]);  // = 0; = 10; = 100; = 1000; = 10000;
    // mpirun -np 1(num_procs) task2_qc 100(n_points) 0.00003(eps) 0(shift)


    double start, end;
    int my_id, num_procs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);  // Get the total number of processes
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);  // Get the rank of the current process
    start = MPI_Wtime();

    double sum_local = 0.0, sum = 0.0, integral = 0.0, current_gap = fabs(real_integral);

```

```
    int i, seed = my_id + shift;
    int n_times = 0;
    while (current_gap > eps){
        n_times += 1;
        srand(seed);
        seed += num_procs;

        for (i = 1; i <= n_points; ++i){
            double x = generate_point(-1.0, 1.0);
            double y = generate_point(-1.0, 1.0);
            double z = generate_point(0.0, 1.0);

            if (x * x + y * y + z * z <= 1){
                sum_local += f(x, y, z);
            }
        }

        MPI_Allreduce(&sum_local, &sum, 1, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD);

        integral = 4.0 * sum / (n_points * num_procs * n_times);
        current_gap = fabs(integral - real_integral);
    }
    end = MPI_Wtime();

    if(my_id == 0){
        printf("Finally, integral is approximated as %.8f \n", integral);
        printf("Random points: %d, the current gap: %.8f, runtime: %.4f. \n
    \n", n_points * num_procs * n_times, current_gap, end - start);
    }

    MPI_Finalize();
    return 0;
}
```