



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	3
2	Численный метод решения задачи	3
3	Аналитическое решение задачи	3
4	Краткое описание программной реализации	4
5	Исследование масштабируемости программы	5
6	Приложение: Код программы	7

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:

$$I = \iiint_G e^{x^2+y^2}z \, dxdydz$$

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: $\Pi: \begin{cases} -1.0 \leq x \leq 1.0 \\ -1.0 \leq y \leq 1.0 \\ 0 \leq z \leq 1.0 \end{cases}$

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:

$$I = \iiint_G e^{x^2+y^2}z \, dxdydz = \iiint_{\Pi} F(x, y, z) dxdydz$$

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

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

где $|\Pi|$ – объём параллелепипеда Π . $|\Pi| = 4$

3 Аналитическое решение задачи

Найдём точное значение интеграла аналитически:

$$I = \iiint_G e^{x^2+y^2}z \, dxdydz$$

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

$$\begin{aligned}
I &= \iint_{x^2+y^2 \leq 1} e^{x^2+y^2} \left(\int_0^{\sqrt{1-x^2-y^2}} z \, dz \right) dx dy = \iint_{x^2+y^2 \leq 1} \left(\frac{1}{2} z^2 \Big|_0^{\sqrt{1-x^2-y^2}} \right) \cdot e^{x^2+y^2} dx dy \\
&= \frac{1}{2} \iint_{x^2+y^2 \leq 1} (1 - x^2 - y^2) \cdot e^{x^2+y^2} dx dy = \{\text{полярная система координат}\} \\
&= \frac{1}{2} \int_0^{2\pi} d\theta \int_0^1 (1 - r^2) e^{r^2} \cdot r dr = \pi \int_0^1 (1 - r^2) e^{r^2} \cdot r dr = \{\text{замена } r^2 \text{ на } c\} \\
&= \pi \int_0^1 (1 - c) e^c \cdot \frac{1}{2} dc = \frac{1}{2} \pi \int_0^1 (1 - c) d(e^c) = \{\text{по частям}\} \\
&= \frac{1}{2} \pi \left((1 - c) e^c \Big|_0^1 - \int_0^1 -e^c dc \right) = \frac{1}{2} \pi \left(-1 + e^c \Big|_0^1 \right) = \frac{1}{2} \pi (-1 + e - 1) = \frac{1}{2} \pi (e - 2)
\end{aligned}$$

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.

Как видно из таблицы 5.1 и графики 5.2, сравнивая ускорение работы программы с разным количеством процессов при одинаковой точности, видно, что при увеличении числа MPI-процессов, скорость работы программы увеличивается. Это как наше ожидание.

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	4.86	$0.36 \cdot 10^{-5}$
	32	0.0094	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	$3.63 \cdot 10^{-6}$
	32	0.0094	7.12	$3.63 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	1	0.1947	1	$0.99 \cdot 10^{-6}$
	4	0.0639	3.04	$1.12 \cdot 10^{-6}$
	16	0.0443	4.40	$1.04 \cdot 10^{-6}$
	32	0.0285	6.83	$1.04 \cdot 10^{-6}$

Рис. 5.1: Таблица с результатами расчётов для системы Polus

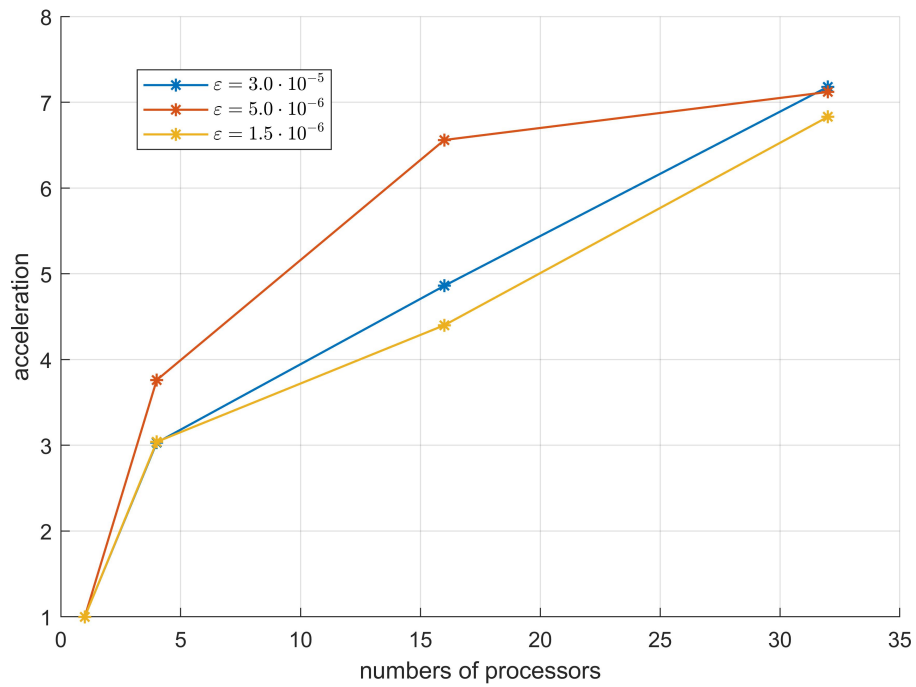


Рис. 5.2: Зависимость ускорения программы от числа используемых MPI-процессов

6 Приложение: Код программы

```
1 #include "mpi.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5 #include <math.h>
6
7 #define real_integral M_PI * (M_E - 2.0) / 2.0 // analytical value of the
   integral
8
9 double f(double x, double y, double z){
10     return z * exp(x * x + y * y);
11 }
12
13 double generate_point(double left_range, double right_range){
14     return ((double)rand() / RAND_MAX) * (right_range - left_range) +
   left_range; //generate point between [left_range, right_range]
15 }
16
17 int main(int argc, char *argv[]){
18
19     double eps = atof(argv[1]);
20     double start, end;
21     int my_id, num_procs;
22
23     MPI_Init(&argc, &argv);
24     MPI_Comm_size(MPI_COMM_WORLD, &num_procs); // Get the total number of
   processes
25     MPI_Comm_rank(MPI_COMM_WORLD, &my_id); // Get the rank of the current
   process
26     start = MPI_Wtime();
27
28     int i, seed = my_id;
29     int n_times = 0;
30     long long int n_points = 1000; // Each process adds n_points random
   points at a time
31     double sum_local = 0.0, sum = 0.0, integral = 0.0, current_gap = fabs(
   real_integral);
32     while (current_gap > eps){
33         n_times += 1;
34         srand(seed);
```

```

35     seed += num_procs;
36     for (i = 1; i <= n_points; ++i){
37         double x = generate_point(-1.0, 1.0);
38         double y = generate_point(-1.0, 1.0);
39         double z = generate_point(0.0, 1.0);
40
41         if (x * x + y * y + z * z <= 1){
42             sum_local += f(x, y, z);
43         }
44     }
45
46     MPI_Allreduce(&sum_local, &sum, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);
47
48     integral = 4.0 * sum / (n_points * num_procs * n_times);
49     current_gap = fabs(integral - real_integral);
50 }
51 end = MPI_Wtime();
52
53 if(my_id == 0){
54     printf("Finally, integral is approximated as %.8f \n", integral);
55     printf("Random points: %lld, the current gap: %.8f, runtime: %.4f.
\n\n", n_points * num_procs * n_times, current_gap, end - start);
56 }
57
58 MPI_Finalize();
59 return 0;
60 }

```