

Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра информационной безопасности

Мирпулатов Исломбек Пулат-угли

# **Численное интегрирование многомерных функций методом Монте-Карло**

Суперкомпьютерное моделирование и технологии

Группа 608, Вариант 1

Москва, 2022

# 1 Постановка задачи

В качестве модельной задачи предлагается задача вычисления многомерного интеграла методом Монте-Карло. Программная реализация должна быть выполнена на языке C или C++ с использованием библиотеки параллельного программирования MPI.

Требуется исследовать масштабируемость параллельной MPI-программы на следующих параллельных вычислительных системах ВМК МГУ:

1. IBM Blue Gene/P,
2. IBM Polus

## 1.1 Математическая постановка задачи

Функция  $f(x, y, z)$  - непрерывна в ограниченной замкнутой области  $G \subset \mathbb{R}^3$

Требуется вычислить определенный интеграл:

$$I = \iiint_G xy^2z^3 dx dy dz$$

где область  $G$  ограничена поверхностями  $z = xy, y = x, x = 1, z = 0$ .

## 1.2 Численный метод решения задачи

Метод Монте-Карло для численного интегрирования.

Пусть область  $G$  ограничена параллелепипедом:  $\Pi : \begin{cases} 0 \leq x \leq 1 \\ 0 \leq y \leq 1 \\ 0 \leq z \leq 1 \end{cases}$

Рассмотрим функцию:  $F(x, y, z) = \begin{cases} xy^2z^3, & (x, y, z) \in G \\ 0, & (x, y, z) \notin G \end{cases}$

Преобразуем искомый интеграл:

$$I = \iiint_G xy^2z^3 dx dy dz = \iiint_{\Pi} F(x, y, z) dx dy dz$$

Пусть  $p_1(x_1, y_1, z_1), p_2(x_2, y_2, z_2), \dots$  - случайные точки, равномерно распределённые в  $\Pi$ . 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  $\Pi$ .  $|\Pi| = 1$

Вариант 1 подразумевает независимую генерацию точек MPI-процессами.

## 2 Решение

### 2.1 Аналитическое решение

$$I = \iiint_G xy^2 z^3 dx dy dz$$

$$G = \{(x, y, z) : x \in [0, 1], y \in [0, x], z \in [0, xy]\}$$

$$\begin{aligned} I &= \int_0^1 dx \int_0^x dy \int_0^{xy} xy^2 z^3 dz = \\ &= \int_0^1 dx \int_0^x \frac{xy^2 z^4}{4} \Big|_0^{xy} dy = \int_0^1 dx \int_0^x \frac{x^5 y^6}{4} dy = \\ &= \int_0^1 \frac{x^5 y^7}{28} \Big|_0^x dx = \int_0^1 \frac{x^{12}}{28} dx = \frac{x^{13}}{364} \Big|_0^1 = \frac{1}{364} \end{aligned}$$

### 2.2 Описание программной реализации

Реализуем адаптивный по точности алгоритм Монте-Карло численного интегрирования.

Программа на вход принимает требуемую точность EPS. Задаем кол-во процессов M.

На каждом процессе генерируется  $\frac{N}{M}$  точек (x, y, z) из параллелепипеда (в моем случае куб  $[0, 1]^3$ ). Для каждой точки проверяем попала ли в заданную область :  $x \geq y$  и  $xy \geq z$ . Если да, то к локальной сумме добавляется значение  $xy^2 z^3$ , в противном случае 0. Далее все локальные суммы через MPI\_Allreduce объединяются в общую сумму. Эта сумма делится на количество точек N.

Проверяем, если разница между решением полученным аналитическим путем и нашим значением из предыдущего пункта не превышает требуемой точности, то останавливаемся. В обратном случае повторяем предыдущий пункт.

Оценка погрешности метода Монте-Карло:

$$EPS \approx O\left(\frac{1}{\sqrt{N}}\right)$$

Отсюда следует, что можно вычислить примерное N:

$$N \approx \frac{1}{EPS^2}$$

Но так как это число получается слишком большое, было решено выбрать:

$$N \approx \frac{1000}{EPS}$$

Этого количества оказалось достаточно.

В результате работы программы выводится:

- Посчитанное приближённое значение интеграла.
- Ошибка посчитанного значения.
- Количество сгенерированных случайных точек.
- Время работы программы в секундах.

## 2.3 Программная реализация

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <mpi.h>
4 #include <math.h>
5
6 double EXACT = 0.002747253;
7
8 int main(int argc, char **argv)
9 {
10     int psize, prank, ierr, rand_seed;
11     double result, eps;
12     double x, y, z;
13
14
15     eps = strtod(argv[1], NULL);
16     rand_seed = strtol(argv[2], NULL, 10);
17     ierr = MPI_Init(&argc, &argv);
18     ierr = MPI_Comm_size(MPI_COMM_WORLD, &psize);
19     ierr = MPI_Comm_rank(MPI_COMM_WORLD, &prank);
20
21     srand(prank * rand_seed);
22
23     double res_time, time_elapsed = MPI_Wtime();
24
25     double global_sum, local_sum = 0.0;
26     long n = 1000 / (eps);
27     long batch = n / psize;
28
29     long i, it = 0;
30     do
31     {
32         it++;
33         for (i = 0; i < batch; i++)
34         {
35             x = (double) rand() / RAND_MAX;
36             y = (double) rand() / RAND_MAX;
37             z = (double) rand() / RAND_MAX;
38             if ((x >= y) && (x * y >= z))
```

```

39         {
40             local_sum = local_sum + x * y * y * z * z * z;
41         }
42     }
43
44     ierr = MPI_Allreduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM,
45                         MPI_COMM_WORLD);
46
47     result = global_sum / (batch * psize * it);
48
49 }
50 while (eps < fabs(EXACT - result));
51
52
53 time_elapsed = MPI_Wtime() - time_elapsed;
54
55 MPI_Reduce(&time_elapsed, &res_time, 1, MPI_DOUBLE, MPI_MAX, 0,
56           MPI_COMM_WORLD);
57
58 if (prank == 0)
59 {
60     double error = fabs(EXACT - result);
61     printf("result:\t%.10lf\n", result);
62     printf("error:\t%.10lf\n", error);
63     printf("iters:\t%ld\n", it * n);
64     printf("time:\t%.10lf\n", res_time);
65 }
66
67 ierr = MPI_Finalize();
68 return 0;
69 }

```

## 2.4 Исследование масштабируемости программы на системах Blue Gene/P и Polus

Таблица 1. Таблица с результатами расчётов для системы Polus для seed = prank \* 13

Точность EPS	Число MPI процессов	Время работы программы	Ошибка	Ускорение
$3.0 \cdot 10^{-5}$	1	3.761	0.0000017742	1.00
$3.0 \cdot 10^{-5}$	4	0.952	0.0000005586	3.95
$3.0 \cdot 10^{-5}$	16	0.435	0.0000038100	8.65
$3.0 \cdot 10^{-5}$	32	0.344	0.0000046010	10.93
$5.0 \cdot 10^{-6}$	1	22.482	0.0000012201	1.00
$5.0 \cdot 10^{-6}$	4	5.718	0.0000007715	3.93
$5.0 \cdot 10^{-6}$	16	1.616	0.0000010205	13.91
$5.0 \cdot 10^{-6}$	32	0.875	0.0000025367	25.69
$1.5 \cdot 10^{-6}$	1	74.735	0.0000013346	1.00
$1.5 \cdot 10^{-6}$	4	19.122	0.0000009006	3.91
$1.5 \cdot 10^{-6}$	16	4.954	0.0000001053	15.08
$1.5 \cdot 10^{-6}$	32	2.777	0.0000002819	26.91

Таблица 2. Таблица с результатами расчётов для системы Polus для seed = prank \* 67

Точность EPS	Число MPI процессов	Время работы программы	Ошибка	Ускорение
$5.0 \cdot 10^{-6}$	1	22.592	0.0000012201	1.00
$5.0 \cdot 10^{-6}$	4	5.850	0.0000009998	3.86
$5.0 \cdot 10^{-6}$	16	1.682	0.0000015661	13.43

Таблица 3. Таблица с результатами расчётов для системы Polus для seed = prank \* 145

Точность EPS	Число MPI процессов	Время работы программы	Ошибка	Ускорение
$5.0 \cdot 10^{-6}$	1	22.538	0.0000012201	1.00
$5.0 \cdot 10^{-6}$	4	5.877	0.0000000150	3.83
$5.0 \cdot 10^{-6}$	16	1.714	0.0000005031	13.14

Таблица 4. Таблица с результатами расчётов для системы Polus для seed = prank \* 222

Точность EPS	Число MPI процессов	Время работы программы	Ошибка	Ускорение
$5.0 \cdot 10^{-6}$	1	23.281	0.0000012201	1.00
$5.0 \cdot 10^{-6}$	4	7.212	0.0000013670	3.22
$5.0 \cdot 10^{-6}$	16	1.615	0.0000011940	14.42

Таблица 5. Таблица с результатами расчётов для системы Polus для seed = prank \* 345

Точность EPS	Число MPI процессов	Время работы программы	Ошибка	Ускорение
$5.0 \cdot 10^{-6}$	1	22.9	0.0000012201	1.00
$5.0 \cdot 10^{-6}$	4	5.881	0.0000037048	3.89
$5.0 \cdot 10^{-6}$	16	1.706	0.0000011559	13.42

Таблица 6. Таблица с результатами расчётов для системы Polus для seed = prank \* 555

Точность EPS	Число MPI процессов	Время работы программы	Ошибка	Ускорение
$5.0 \cdot 10^{-6}$	1	22.86	0.0000012201	1.00
$5.0 \cdot 10^{-6}$	4	5.955	0.0000011063	3.83
$5.0 \cdot 10^{-6}$	16	1.689	0.0000005740	13.53

Таблица 7. Таблица с усредненными результатами расчётов для системы Polus

Точность EPS	Число MPI процессов	Среднее время работы	Среднее ускорение
$5.0 \cdot 10^{-6}$	1	22.775	1.00
$5.0 \cdot 10^{-6}$	4	6.082	3.76
$5.0 \cdot 10^{-6}$	16	1.670	13.64

График 1. Зависимость времени работы от количества процессов для seed = 13 \* rank

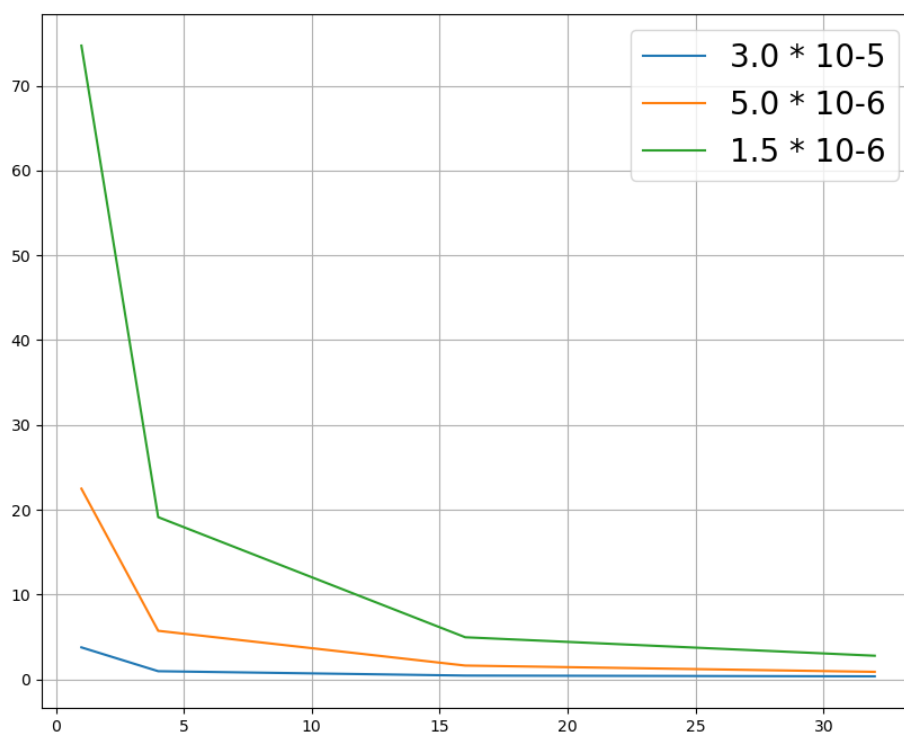


График 2. Зависимость времени работы от количества процессов для разных сидов

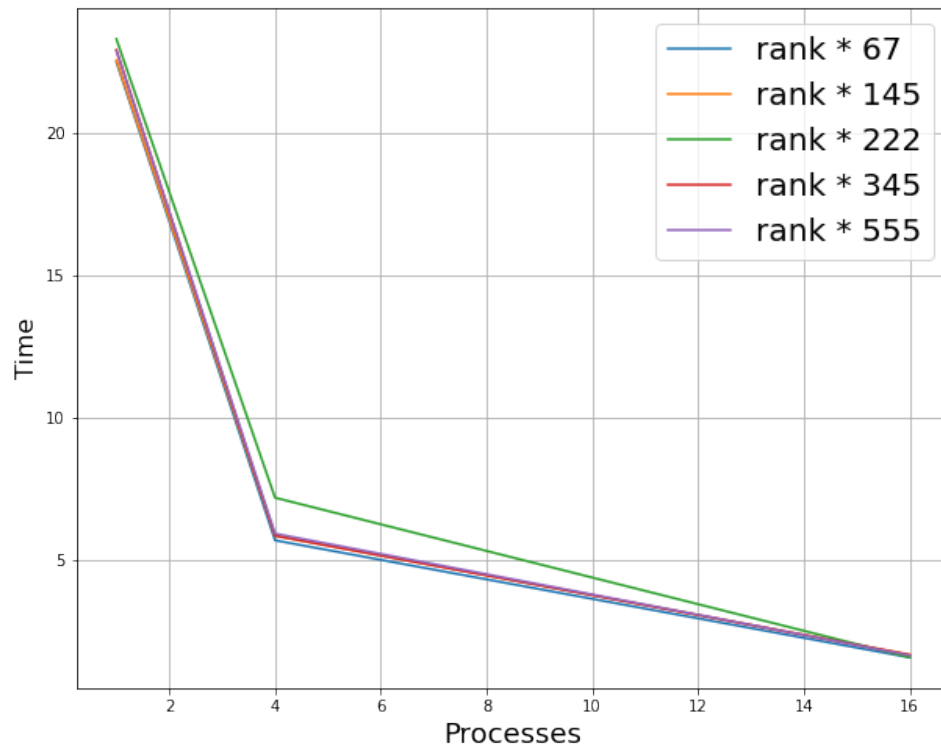


График 3. Зависимость ускорения от количества процессов для разных сидов

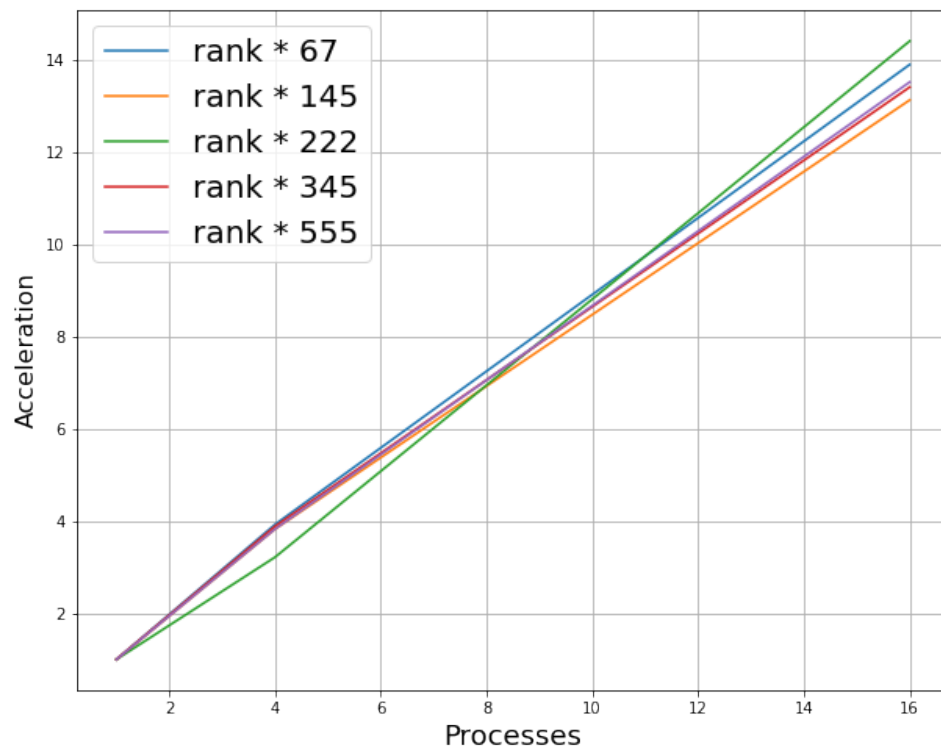




График 4. Зависимость усредненного времени работы от количества процессов

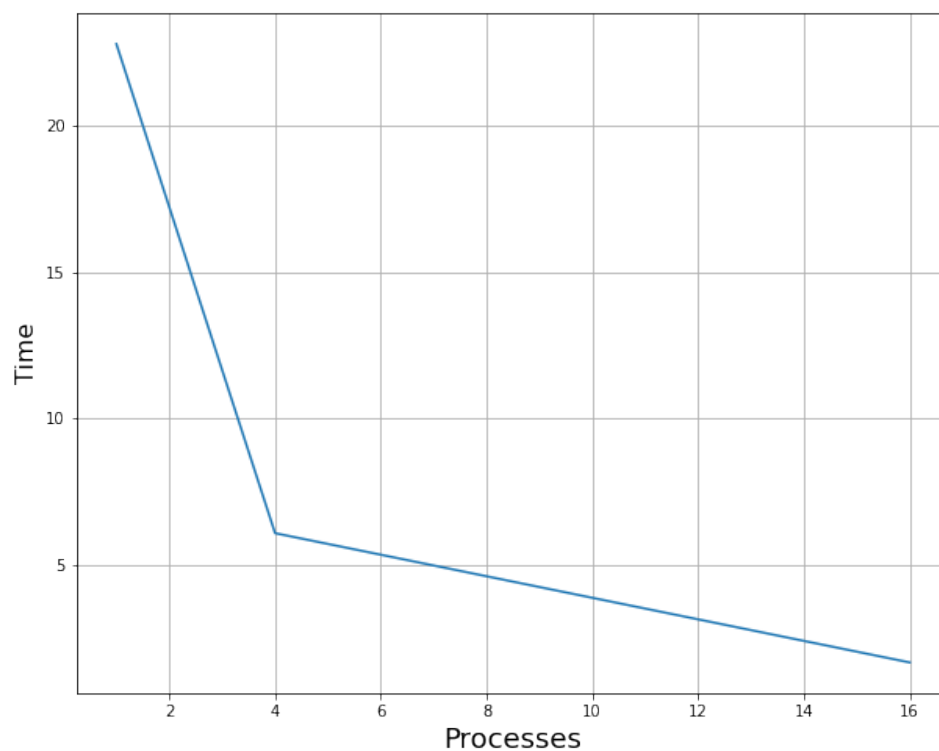


График 5. Зависимость усредненного ускорения от количества процессов

