

Московский государственный университет имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра математических методов прогнозирования

**Задание по курсу «Суперкомпьютерное
моделирование и технологии»**

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

Выполнил:

студент 617 группы

Г.В. Кормаков

Содержание

1	Постановка задачи	2
2	Аналитическое решение	3
3	Описание программной реализации	3
4	Результаты на системах Blue Gene/P и Polus	5
5	Заключение	8
6	Литература	9
7	Приложение 1. Код программы	10

1 Постановка задачи

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

Требуется вычислить определённый интеграл:

$$I = \iiint_G f(x, y, z) dx dy dz$$

В рамках задания требуется выполнить подсчёт данного интеграла для функции

$$f(x, y, z) = \sqrt{x^2 + y^2} \quad (1)$$

на области G , ограниченной поверхностями $x^2 + y^2 = z^2, z = 1$ ¹. **Дополнительно** **везде возьмём нижнюю границу равной $z = 0$** (в случае, если ограничения были вида $|z| = 1$ значение интеграла увеличивается в 2 раза и изменится вид параллелограмма.)

Для данного интеграла можно вычислить аналитическое выражение (приведено в разделе 2).

В рамках программной реализации предлагается осуществить вычисление данного интеграла с помощью метода Монте-Карло.

Для этого рассмотрим область G , ограниченную параллелепипедом:

$$\Pi : \begin{cases} a_1 \leq x \leq b_1 \\ a_2 \leq y \leq b_2 \\ a_3 \leq z \leq b_3 \end{cases} \quad (2)$$

Рассмотрим функцию:

$$F(x, y, z) = \begin{cases} f(x, y, z) = \sqrt{x^2 + y^2}, & (x, y, z) \in G \\ 0, & (x, y, z) \notin G \end{cases} \quad (3)$$

Преобразуем искомый интеграл:

$$I = \iiint_G f(x, y, z) dx dy dz = \iiint_{\Pi} F(x, y, z) dx dy dz$$

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

$$I \approx |\Pi| \cdot \frac{1}{n} \sum_{i=1}^n F(p_i) \quad (4)$$

¹Именно такое условие формулируется в задании, однако необходимо указать нижнюю границу поверхности

где $|\Pi|$ — объём параллелепипеда Π . $|\Pi| = (b_1 - a_1)(b_2 - a_2)(b_3 - a_3)$

Отметим, что необязательно ограничивать область определения функции прямоугольником, если есть знание об области G , поскольку достаточно уметь равномерно брать случайные точки в этой области.

2 Аналитическое решение

Распишем аналитическое решение интеграла с функцией 1.

$$\begin{aligned} I &= \iiint_G f(x, y, z) dx dy dz \equiv \iiint_G \sqrt{x^2 + y^2} dx dy dz = \{z \in [0, 1]\} = \\ &= \iiint_{\substack{x^2 + y^2 \leq z^2 \\ 0 \leq z \leq 1}} \sqrt{x^2 + y^2} dx dy dz = \int_0^1 \int_{-z}^z \int_{-\sqrt{z^2 - y^2}}^{\sqrt{z^2 - y^2}} \sqrt{x^2 + y^2} dx dy dz \end{aligned} \quad (5)$$

Осуществим в 5 переход к цилиндрическим координатам:

$$\begin{cases} x = \rho \cos(\phi) \\ y = \rho \sin(\phi) \\ z = h \end{cases}, \quad \rho \geq 0, \quad \phi \in [0, 2\pi], \quad h \in \mathbb{R} \quad (6)$$

Якобиан перехода 6 равен

$$J = \begin{vmatrix} \cos(\phi) & -\rho \sin(\phi) & 0 \\ \sin(\phi) & \rho \cos(\phi) & 0 \\ 0 & 0 & 1 \end{vmatrix} = \rho \cos^2(\phi) + \rho \sin^2(\phi) = \rho \quad (7)$$

Запишем

$$\begin{aligned} (5) &= \iiint_{\substack{\rho^2 \leq h^2 \leq 1 \\ \phi \in [0, 2\pi]}} \rho J d\rho d\phi dh = \int_0^1 \int_0^{2\pi} \rho^2 d\phi \int_{\rho}^1 dh d\rho = \int_0^1 2\pi \rho^2 \cdot (1 - \rho) d\rho = \\ &= 2\pi \left(\frac{\rho^3}{3} - \frac{\rho^4}{4} \right) \Big|_0^1 = 2\pi \left(\frac{1}{3} - \frac{1}{4} \right) = \frac{\pi}{6} \end{aligned} \quad (8)$$

3 Описание программной реализации

Подсчёт интеграла методом Монте-Карло осуществляется по схеме «мастер-рабочий» (master-slave), предполагающей, что один главный процесс создаёт n случайных точек для приближённого вычисления интеграла по формуле 4. Также необ-

ходимо задать ограничения на область интегрирования в виде прямоугольника 2. Ограничения на область (по координатам x, y, z , соответственно) логичным образом дают параллелепипед с координатами $[-1, 1] \times [-1, 1] \times [0, 1]$.

Область для исходной функции задаётся ограничениями вида

$$[-\sqrt{z^2 - y^2}, \sqrt{z^2 - y^2}] \times [-z, z] \times [0, 1]$$

Опишем последовательность действий для требуемого вычисления. Поскольку на вход подаётся требуемая точность ε , то можно взять изначально некоторое число точек для генерации N . Например, можно сделать следующее предположение: из [1] известна оценка точности на текущем шаге: $\varepsilon = \sqrt{D_n(f)/n}$, где D_n – оценка дисперсии на этом шаге. Предположим, что $\varepsilon \geq \sqrt{1/N}$, тогда можем взять $N_0 \equiv \lfloor \frac{1}{\varepsilon^2} \rfloor$ в качестве начального числа точек (однако уменьшим до $N_0 \equiv \lfloor \frac{1}{\varepsilon} \rfloor$ для оптимизации по объёму выделенной начальной памяти).

Отметим, что в представленных в следующем разделе результатах выбор начального N следующим образом повлиял на оценку скорости некоторых замеров (например, на системе Polus в случае $\varepsilon = 5.0 \cdot 10^{-6}$).

Далее в программной реализации инициализируется основной цикл метода, процесс-мастер (далее «мастер») генерирует для каждого процесса-рабочего (далее «рабочего») случайные точки. Каждый «рабочий» считает сумму значений функций на этих случайных точках и отправляет результат обратно «мастеру». Также «рабочие» считают время собственного исполнения.

В результате, «мастер» получает обратно подсчитанные суммы на подмассивах и максимальное из времён исполнения «рабочих». «Мастер» осуществляет проверку на близость средней суммы к истинному значению (из раздела 2) и отправляет значение критерия всем «рабочим» для продолжения или остановки работы. Если критерий не достигнут, то генерируется ещё N новых точек.

В конце программы «мастер» выводит подсчитанное значение, абсолютную ошибку, количество потребовавшихся точек и затраченное время (как максимум из времени, потребовавшегося «рабочим»).

Полный текст программы приведён в разделе 7 (листинг 1).

4 Результаты на системах Blue Gene/P и Polus

Для данной задачи выполнены подсчёты ускорения программы на системах Blue Gene/P и Polus.

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

$$S_p = \frac{T_2}{T_p}$$

где T_2 — время работы на минимальном числе MPI-процессов (для схемы мастер-рабочие (master-slave) равно 2), T_p — время работы программы на p MPI-процессах.

Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$1.0 \cdot 10^{-4}$	2	2.367	1	$8.7 \cdot 10^{-5}$
	4	2.253	1.05	$8.5 \cdot 10^{-5}$
	16	2.217	1.07	$9.4 \cdot 10^{-5}$
	64	2.318	1.02	$8.3 \cdot 10^{-5}$
$2.0 \cdot 10^{-5}$	2	4.636	1	$1.8 \cdot 10^{-5}$
	4	4.402	1.05	$1.9 \cdot 10^{-5}$
	16	4.292	1.08	$1.6 \cdot 10^{-5}$
	64	4.313	1.07	$1.1 \cdot 10^{-5}$
$8 \cdot 10^{-6}$	2	4.663	1	$4.7 \cdot 10^{-6}$
	4	4.425	1.05	$5.7 \cdot 10^{-6}$
	16	5.220	0.89	$6.8 \cdot 10^{-6}$
	64	5.218	0.89	$7.2 \cdot 10^{-6}$

Таблица 1: Таблица с результатами расчётов для системы Blue Gene/P

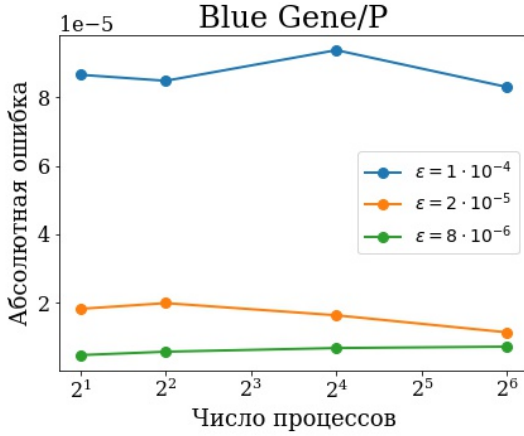
Точность ε	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	2	0.629	1	$1.8 \cdot 10^{-5}$
	4	0.584	1.08	$1.8 \cdot 10^{-5}$
	16	0.705	0.89	$1.7 \cdot 10^{-5}$
	64	0.649	0.97	$1.8 \cdot 10^{-5}$
$5.0 \cdot 10^{-6}$	2	0.753	1	$4.8 \cdot 10^{-6}$
	4	7.110	0.11	$2.4 \cdot 10^{-6}$
	16	7.599	0.10	$2.5 \cdot 10^{-6}$
	64	6.858	0.11	$4.3 \cdot 10^{-6}$
$1.5 \cdot 10^{-6}$	2	541.636	1	$1.45 \cdot 10^{-6}$
	4	522.505	1.03	$1.45 \cdot 10^{-6}$
	16	547.198	0.99	$1.45 \cdot 10^{-6}$
	64	477.506	1.13	$1.45 \cdot 10^{-6}$

Таблица 2: Таблица с результатами расчётов для системы Polus

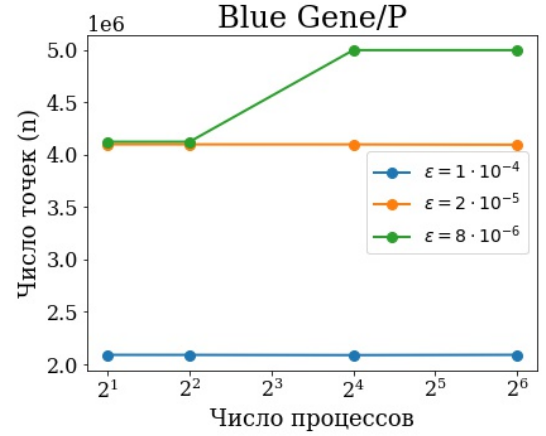
Более наглядно результаты продемонстрированы на графиках 1 и 2. Стоит отметить, что результаты по количеству взятых точек чаще всего константны и равны количеству точек, взятых по умолчанию изначально (как округление вниз $1/\varepsilon$). Отклонения связаны со случайностью выбора точек для каждого процесса. Прокомментируем результаты графиков.

На Blue Gene/P

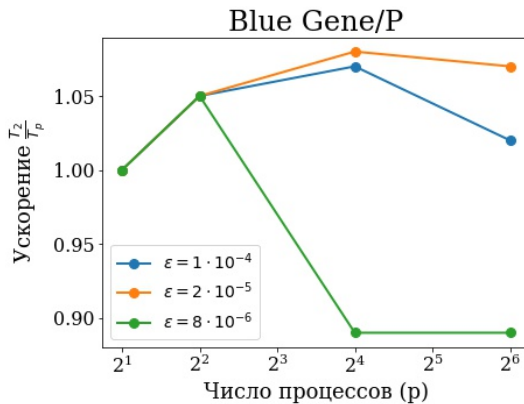
- Не видна зависимость точности (по абсолютной ошибке) от числа точек. Для оценки данной зависимости недостаточно приведённых данных.
- Для случая $8 \cdot 10^{-6}$ из-за выбора большего числа точек время выполнения возросло, что отражает график ускорения 1c.
- Также на графике 1c видно, что при увеличении числа процессов ускорение начинает снижаться. «Мастер» принимает посчитанные данные от большего числа «рабочих» и максимальное время работы «рабочих» увеличивается.



(a)



(b)

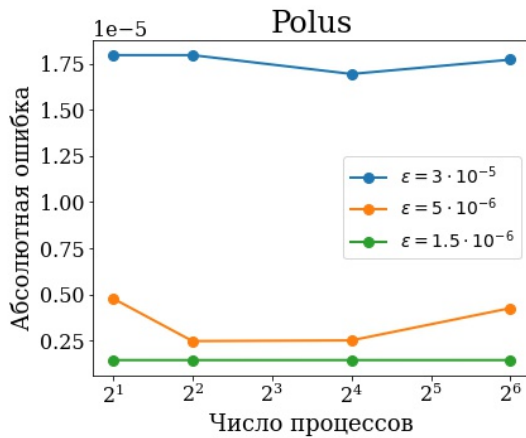


(c)

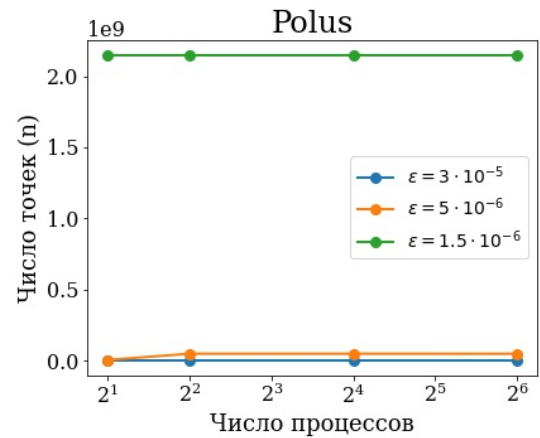
Рис. 1: Сравнение получаемых характеристик запусков на Blue Gene/P (абсолютная ошибка – a, число требуемых точек для достижения точности – b, ускорение относительно запуска на 2-х процессах – c)

На Polus

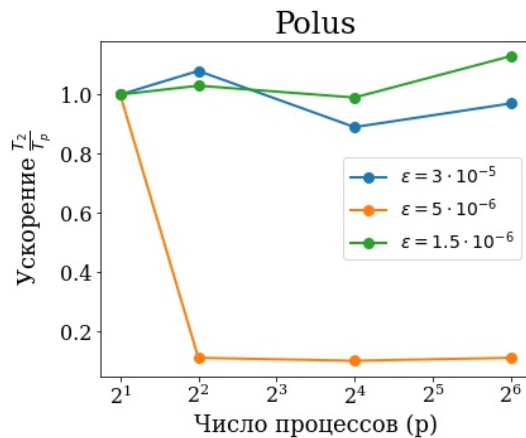
- Не видна зависимость точности (по абсолютной ошибке) от числа точек. Для оценки данной зависимости также недостаточно приведённых данных.
- График ускорения показывает странное поведение для $\varepsilon = 5 \cdot 10^{-6}$. Однако данное поведение воспроизводилось на нескольких запусках. Объяснить этот эффект можно запуском на удачных точках, позволяющих оценить интеграл с данной точностью. Поскольку выбирается сразу достаточно точек, то один процесс справляется с задачей вычисления быстрее, чем большее число.
- На системе Polus видна обратная зависимость по ускорению на большем числе узлов. С ростом числа процессов ускорение растёт (см. график 2с). Этот эффект может объясняться более эффективной топологией при взаимодействии «мастера» с «рабочими».



(a)



(b)



(c)

Рис. 2: Сравнение получаемых характеристик запусков на Polus (абсолютная ошибка – а, число требуемых точек для достижения точности – б, ускорение относительно запуска на 2-х процессах – в)

5 Заключение

В ходе экспериментов с реализацией не выявлено каких-либо существенных закономерностей по точности вычисления. Однако на приведённых графиках можно отследить различие архитектур Blue Gene/P и Polus.

Также можно отметить, что приведённые результаты не показывают значимых различий из-за попытки задать оптимальное (с некоторой теоретической стороны) начальное число точек. При меньшем числе точек скорости выполнения имели бы более существенные отклонения.

6 Литература

Источники

- [1] Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. ЧИСЛЕННЫЕ МЕТОДЫ. - М.: Наука, 1987.

7 Приложение 1. Код программы

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "mpi.h"
4 #include <time.h>
5 #include <math.h>
6
7
8 double func(double x, double y, double z){
9     if ((x*x + y*y <= z*z) && ((z >= 0) && (z <= 1)))
10         return sqrt(x*x + y*y);
11     return 0.0;
12 }
13
14
15 int main(int argc, char *argv[]){
16     if (argc != 2){
17         printf("Program receive %d numbers. Should be 1: epsilon\n", argc);
18         return -1;
19     }
20
21     double epsilon = atof(argv[argc-1]);
22     if (epsilon <= 0){
23         printf("Epsilon should be > 0!!!\n");
24         return -1;
25     }
26
27     int N;
28     if (epsilon < 1e-7){
29         printf("Set epsilon < 1e-7!");
30         return -1;
31     }
32     N = roundf(1 / epsilon);
33
34     int myrank, nprocess;
35
36     MPI_Init(&argc, &argv);
37     MPI_Comm_size(MPI_COMM_WORLD, &nprocess);
38     if (nprocess < 2){
39         printf("Too few processes!");
40         MPI_Finalize();
41         return -1;
42     }
43     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
44
45     int block_size = N / (nprocess-1);
46     double (*p)[3] = malloc(sizeof(double[block_size][3]));
47     //double *p = malloc(sizeof(double) * 3 * block_size);
48
49     double global_integral;
50     double current_sum = 0.;
51     int converge = 0;
52     double error;
53     double total_points_amount = 0.;
54     double whole_time = 0.;
55     double start, finish, slave_time;
56     start = MPI_Wtime();
57
58     while (!converge) {
59         if (!myrank) {
60             total_points_amount += ((nprocess - 1) * block_size);
```

```

61 // srand(time(NULL)); //Unnecessary string
62     for (int i = 1; i < nprocess; i++) {
63         for (int j = 0; j < block_size; j++) {
64             p[j][0] = 2 * (double) rand() / RAND_MAX - 1;
65             p[j][1] = 2 * (double) rand() / RAND_MAX - 1;
66             p[j][2] = (double) rand() / RAND_MAX;
67         }
68
69         MPI_Send(p, block_size * 3, MPI_DOUBLE, i, i, MPI_COMM_WORLD);
70     }
71
72     double total_sum = 0.0;
73 // double local_sum;
74 // MPI_Status status;
75 // for (int i = 1; i < nprocess; total_sum += local_sum, ++i)
76 // MPI_Recv(&local_sum, 1, MPI_DOUBLE, i, i, MPI_COMM_WORLD, &status);
77
78     MPI_Reduce(MPI_IN_PLACE, &total_sum, 1, MPI_DOUBLE, MPI_SUM, 0,
79               MPI_COMM_WORLD);
80
81     MPI_Reduce(MPI_IN_PLACE, &whole_time, 1, MPI_DOUBLE, MPI_MAX, 0,
82               MPI_COMM_WORLD);
83
84     current_sum += total_sum;
85
86     global_integral = current_sum / total_points_amount;
87     global_integral *= 4.;
88
89     error = fabs(global_integral - (M_PI / 6));
90     // whole_time += max_time;
91     if (error < epsilon)
92         converge = 1;
93
94     for (int i = 1; i < nprocess; i++) {
95         MPI_Send(&converge, 1, MPI_INT, i, i, MPI_COMM_WORLD);
96     }
97 } else {
98     MPI_Status status;
99     MPI_Recv(p, block_size * 3, MPI_DOUBLE, 0, myrank, MPI_COMM_WORLD,
100             &status);
101
102     double sum = 0.0;
103     for (int j = 0; j < block_size; j++)
104         sum += func(p[j][0], p[j][1], p[j][2]);
105
106     // MPI_Send(&sum, 1, MPI_DOUBLE, 0, myrank, MPI_COMM_WORLD);
107     MPI_Reduce(&sum, NULL, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
108
109     finish = MPI_Wtime();
110     slave_time = finish - start;
111
112     MPI_Reduce(&slave_time, NULL, 1, MPI_DOUBLE, MPI_MAX, 0,
113               MPI_COMM_WORLD);
114     MPI_Recv(&converge, 1, MPI_INT, 0, myrank, MPI_COMM_WORLD, &status);
115 }
116
117 if (!myrank){
118     printf("True integral value=%.10f\n", M_PI/6);
119     printf("Total Monte Carlo estimate=%.10f\n", global_integral);
120     printf("N points=%d\n", (int)total_points_amount);
121     printf("Final absolute error=%.10f\n", error);

```

```
118     printf("Total time=%.10f sec\n", whole_time);
119 }
120 free(p);
121 MPI_Finalize();
122 return 0;
123 }
```

Листинг 1: mc_master_slave.c