Московский Государственный Университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Численное интегрирование многомерных функций методом Монте-Карло

Выполнил: Мякшин Владислав Эдуардович, 614 группа
6 вариант

Москва, 2022 г.

Введение

В качестве модельной задачи предлагается задача вычисления многомерного интеграла методом Монте-Карло.

Программная реализация должна быть выполнена на языке C++ с использованием библиотеки параллельного программирования MPI.

Требуется исследовать масштабируемость параллельной MPI-программы на следующей параллельной вычислительной системе ВМК МГУ: IMB Polus.

Математическая постановка задачи

Функция $f(x,y,z)$ – непрерывная в ограниченной замкнутой области $G \subset \mathbb{R}^3$. Требуется вычислить определенный интеграл:

$$I = \iiint\limits_G f(x,y,z)dxdydz,$$

где функция $f(x) = sin(x^2+z^2)\cdot y$, $G = \{(x,y,z): x^2+y^2+z^2 \le 1, x \ge 0, y \ge 0, z \ge 0\}$

Нахождение точного значения интеграла аналитически

$$I = \iiint\limits_G y\cdot sin(x^2+z^2)dxdydz = \begin{pmatrix} \mathrm{x} = \mathrm{r}\cdot\sin\phi \\ \mathrm{z} = \mathrm{r}\cdot\cos\phi \\ \mathrm{dxdz} = \mathrm{r}\cdot\mathrm{drd}\phi \\ \mathrm{x}^2+z^2 = r^2 \\ x^2+y^2+z^2 \Leftrightarrow \mathrm{r}^2+y^2 \le 1 \end{pmatrix} = \iiint\limits_{\substack{\mathrm{y,r}\ge 0 \\ 0\le\phi\le\frac{\pi}{2} \\ r^2+y^2\le 1}} y\cdot sin(r^2)\cdot \mathrm{r}dydrd\phi =$$

$$= \int\limits_{0\le\phi\le\frac{\pi}{2}} \left[\iint\limits_{\substack{\mathrm{y,r}\ge 0 \\ r^2+y^2\le 1}} r\cdot y\cdot sin(r^2)dydr \right] \mathrm{d}\phi = \int\limits_{0\le\phi\le\frac{\pi}{2}} \mathrm{d}\phi \iint\limits_{\substack{\mathrm{y,r}\ge 0 \\ r^2+y^2\le 1}} r\cdot y\cdot sin(r^2)dydr =$$

$$= \frac{\pi}{2} \iint\limits_{\substack{\mathrm{y,r}\ge 0 \\ r^2+y^2\le 1}} r\cdot y\cdot sin(r^2)dydr = \begin{pmatrix} r^2 = t \\ 2\mathrm{rdr} = \mathrm{dt} \\ 0 \le r^2 \le 1 \Leftrightarrow 0 \le t \le 1 \end{pmatrix} =$$

$$= \frac{\pi}{2} \iint\limits_{\substack{\mathrm{y,t}\ge 0 \\ \mathrm{t}+y^2\le 1}} y\cdot sin(\mathrm{t})dy\frac{dt}{2} = \frac{\pi}{4} \iint\limits_{\substack{\mathrm{y,t}\ge 0 \\ \mathrm{t}+y^2\le 1}} y\cdot sin(\mathrm{t})dy\mathrm{dt} = \begin{pmatrix} y^2 = s \\ 2\mathrm{ydy} = \mathrm{ds} \\ 0 \le y^2 \le 1 \Leftrightarrow 0 \le s \le 1 \end{pmatrix} =$$

$$= \frac{\pi}{4} \iint\limits_{\substack{\mathrm{s,t}\ge 0 \\ 0\le\mathrm{t}+s\le 1}} sin(\mathrm{t})\frac{ds}{2}\mathrm{dt} = \frac{\pi}{8} \iint\limits_{0\le\mathrm{t}+s\le 1} sin(\mathrm{t})\mathrm{dsdt} = \frac{\pi}{8} \int\limits_{0\le s\le 1} \left[\int\limits_{0\le\mathrm{t}\le 1-\mathrm{s}} sin(t)dt \right] \mathrm{ds} =$$

$$= \frac{\pi}{8}\int\limits_0^1 \left[\left(-cos(t)\right)\Big|_0^{1-s}\right] ds = \frac{\pi}{8}\int\limits_0^1 [-cos(1-s)+cos(0)]\mathrm{ds} = \frac{\pi}{8}\int\limits_0^1 [1-cos(1-s)]\mathrm{ds} =$$

$$= \frac{\pi}{8}\left[\left(s+sin(1-s)\right)\Big|_0^1\right] = \frac{\pi}{8}\left(1+sin(0)-0-sin(1)\right) = \frac{\pi}{8}\left(1-sin(1)\right)$$

Численный метод решения задачи

Пусть область G ограничена параллелепипедом

$$\text{П:}\begin{cases} a_1 \le x \le b_1 \\ a_2 \le \text{y} \le b_2 \\ a_3 \le \text{z} \le b_3 \end{cases}$$

Рассмотрим функцию:

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

Преобразуем искомый интеграл:

$$I = \iiint\limits_{\text{G}} f(x,y,z)dxdydz = \iiint\limits_{\text{П}} F(x,y,z)dxdydz$$

Пусть $p_1(x_1,y_1,z_1), p_2(x_2,y_2,z_2)$, … - случайные точки, равномерно распределенные в П. Возьмем n таких случайных точек. В качестве приближенного значения интеграла предлагается использовать выражение:

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

где |П| - объем параллелепипеда П. $|\text{П}| = (b_1 - a_1)(b_2 - a_2)(b_3 - a_3)$.

В нашем случае $|\text{П}| = 1$

Описание программной реализации

Параллельная MPI-программа принимает на вход требуемую точность и генерирует случайные точки до тех пор, пока требуемая точность не будет достигнута. Программа вычисляет точность как модуль разности между приближенным значением, полученным методом Монте-Карло, и точным значением, вычисленным аналитически.

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

- Посчитанное приближенное значение интеграла
- Ошибка посчитанного значения: модуль разности между приближенным и точным значениями интеграла
- Количество сгенерированных случайных точек
- Время работы программы в секундах

Время работы программы измеряется следующим образом. Каждый MPI-процесс измеряет свое время выполнения, затем среди полученных значений берется максимум.

В программе реализован вариант, когда параллельные процессы генерируют случайные точки независимо друг от друга. Все процессы в этом случае вычисляют свою часть суммы в формуле. Затем вычисляется общая сумма с помощью операции редукции.

В этом варианте параллельной реализации для обеспечения генерации разных последовательностей точек в разных MPI-процессах инициализирован генератор псевдослучайных чисел (в случае использования стандартного генератора – функцией *srand()*) разными числами.

Код программы с небольшими комментариями можно найти в Приложении 1.

Исследование масштабируемости программы на системе Polus

На основе реализованного алгоритма были проведены эксперименты на системе Polus.

Для большей достоверности результатов было принято решение усреднять время работы программы и итоговой ошибки вычисления по 5 запускам.

Согласно постановке задачи был произведен расчет на 1, 4, 16 MPI-процессах.

Точность ϵ	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	1	**0,266315**	**1**	**2,83494E-05**
	4	**0,1349202**	**1,97387048**	**2,65136E-05**
	16	**0,0016872**	**157,8443575**	**1,90915E-05**
$5.0 \cdot 10^{-6}$	1	**0,3119198**	**1**	**4,53339E-06**
	4	**0,1425652**	**2,187909812**	**2,28972E-06**
	16	**0,1555218**	**2,005633937**	**4,25364E-06**
$1.5 \cdot 10^{-6}$	1	**1,39383**	**1**	**1,38378E-06**
	4	**0,1300824**	**10,71497758**	**1,04932E-06**
	16	**0,1918542**	**7,265048146**	**1,16116E-06**

Графики зависимости ускорения в зависимости от числа MPI-процессов для заданной точности можно найти в Приложении 2.

Вывод

Метод Монте-Карло сильно зависит от количества точек, которые будут сгенерированы, так как от этого зависит точность, которую мы получаем. Таким образом, запуски с большим количеством процессов могут работать медленнее, потому что используется другое количество точек для подсчета значения интеграла.

Приложение 1

```
#include <iostream>
#include <cmath>
#include <random>
#include "mpi.h"

const double DEFAULT_INTEGRAL = M_PI / 8 * (1 - sin(1)); // аналитическое значение интеграла

double f(double x, double y, double z) { // подынтегральная функция
  return y * sin(x * x + z * z);
}

bool G(double x, double y, double z) { // принадлежность области определения
  return x >= 0 && y >= 0 && z >= 0 && x * x + y * y + z * z <= 1;
}

double F(double x, double y, double z) { // считаем функцию, если в области определения
  return (G(x, y, z)) ? f(x, y, z) : 0.0;
}

double MonteCarlo(int n) { // сумма случайных точек подынтегральной функции, входящих в область определения. Вход: кол-во точек
  double sum = 0.0;
  for (int i = 0; i < n; ++i) {
    double x = double(rand()) / RAND_MAX;
    double y = double(rand()) / RAND_MAX;
    double z = double(rand()) / RAND_MAX;
    sum += F(x, y, z);
  }
  return sum;
}

double CalculateIntegral(double sum, int n) { // численное нахождение интеграла. Вход: сумма и кол-во точек
  double volume = 1.0 * 1.0 * 1.0;
  return volume * sum / n;
}

int main(int argc, char** argv) {
  if (argc != 2) {
    std::cout << "Неверное число аргументов" << std::endl;
    return -1;
  }
  double eps = atof(argv[1]);
  MPI_Init(&argc, &argv);
  int N_processes, process_id;
  MPI_Comm_size(MPI_COMM_WORLD, &N_processes);
  MPI_Comm_rank(MPI_COMM_WORLD, &process_id);
  srand(time(NULL) + process_id * N_processes);
  double start_time = MPI_Wtime(); // засекаем время
  double integral = 0.0; // переменная интеграла
  int n_points = 10000 / N_processes; // кол-во точек на процесс
  double local_sum = 0; // локальная сумма для одного процесса
  double global_sum = 0; // глобальная сумма для всех процессов
  int cnt_points = 0; // суммарное кол-во точек
  do {
    local_sum += MonteCarlo(n_points); // каждый процесс считает локальную сумму
    MPI_Allreduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD); // прибавляет своё значение к глобальной сумме
    cnt_points += n_points * N_processes; // считаем общее кол-во точек
    integral = CalculateIntegral(global_sum, cnt_points); // считаем интеграл по полученной глобальной сумме и общему кол-ву точек
  } while (fabs(integral - DEFAULT_INTEGRAL) > eps); // проверяем соответствие с точностью

  double end_time = MPI_Wtime(); // стоп таймер
  if (process_id == 0) {
    std::cout << "Приближенное значение интеграла:\t" << integral << std::endl;
    std::cout << "Ошибка посчитанного значения:\t" << fabs(integral - DEFAULT_INTEGRAL) << std::endl;
    std::cout << "Количество сгенерированных случайных точек:\t" << cnt_points << std::endl;
    std::cout << "Время работы выполнения:\t" << end_time - start_time << std::endl;
  }
  MPI_Finalize();
  return 0;
}
```

Приложение 2