Московский государственный университет имени М.В. Ломоносова

Факультет Вычислительной математики и кибернетики

Кафедра Системного программирования

Здание по курсу

«Суперкомпьютерноемоделирование и технологии»

Численное интегрирование многомерных функцийметодом Монте-Карло

Вариант 5

Выполнил:

студент 627 группы

Герасимов Денис Юрьевич

Москва, 2022

Введение

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 кластере Polus

Математическая постановка задачи

Функция f(x, y, z)— непрерывна в ограниченой замкнутой областиG⊂R3.Требуется вычислить определённый интеграл:

I= $\iiint f(x,y,z)dxdydz$ на области G

Численный метод решения задачи Метод Монте-Карло.

Пусть область G ограниченна параллелепипедом P:{ $a_1 < x < a_2; b_1 < y < b_2; c_1 < z < c_2$}

Рассмотрим на неём функцию:F(x, y, z) = (f(x, y, z), (x, y, z)∈G; 0, (x, y, z) ∉G)

Преобразуем искомый интеграл:

I= $\iiint f(x,y,z)dxdydz$ по G

->

I= $\int_{c1}^{c2} \int_{b1}^{b2} \int_{a1}^{a2} F(x,y,z)dxdydz$

Пусть p(x, y, z) — случайные точки, равномерно распределённые в П.

Возьмём n таких случайных точек. В качестве приближённого значенияинтеграла предлагается использовать выражение:

$I \approx |П| * \frac{1}{n} \sum_{i=1}^{n} F(p_i)$, где |П| — объём параллелепипеда П (1)

Требуется реализовать параллельную MPI-программу,которая принимает на вход требуемую точность и генерирует случайные точки до техпор, пока требуемая точность не будет достигнута. Программа вычисляет точность как модуль разности между приближённым значением, полученным методом Монте-Карло, и точным значением, вычислинным аналитически.

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

- Посчитанное приближённое значение интеграла
- Ошибка посчитанного значения: модуль разности между приближённым и точ-ным значениями интеграла
- Количество сгенерированных случайных точек
- Время работы программы в секундах.
  Время работы программы измеряется следующим образом: Каждый MPI-процесс измеряет своё время выполнения, затем среди полученных значений берётся максимум.

В моём варианте параллельные процессы генерируют случайные точки независимо друг от друга. Все вычисляют свою часть суммы в формуле (1). 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.

Для обеспечения генерации разных последовательностей точек в разных MPI-процессах необходимо инициализировать генератор псевдослучайных чисел разными числами.

Мой вариант:

$$I = \iiint\limits_{\substack{-1<x<0 \\ -1<y<0 \\ -1<z<0}} x^3 y^2 z \, dxdydz$$

Аналитическое решение

$$I = \iiint\limits_{\substack{-1<x<0\\-1<y<0\\-1<z<0}} x^3y^2\,z\,dxdydz = \iint\limits_{\substack{-1<x<0\\-1<y<0}} \frac{x^3y^2}{2}dxdy = \int\limits_{-1<x<0} \frac{x^3}{6}dx = \frac{1}{24}$$

Описание реализации

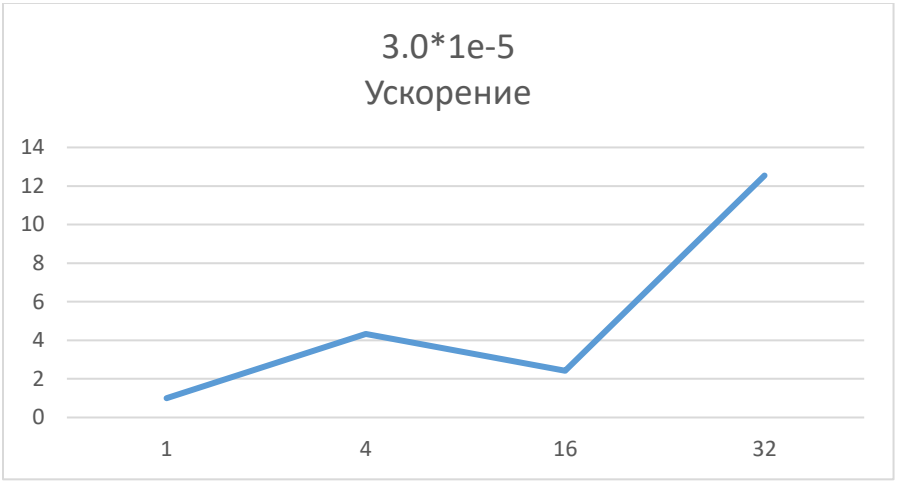
Программа получает на вход требуемую точности, затем каждый процесс инициализирует генератор случайных чисел (для воспроизводимости генератор инициализируется в зависимости от ранка процесса, но без сторонних случайных величин типа таймера)

Изначальный размер порции точек был выбран 10000, при увеличении кол-ва процессов порция для одного процесса = [10000/size]

Процессы генерируют точки, считают локальные суммы значений подинтегральной функции а затем с помощью редукции собирают общую сумму на 0-м процессе. 0-й процесс затем подсчитывает приближенное значение интеграла и сравнивает с ответом.

Если точность не достигнута, то 0-й процесс рассылает всем флаг о том, что нужно продолжать вычисление и процессы начинают новую итерацию.

Результаты


3.0*1e-5
Ускорение
14
12
10
8
6
4
2
0
1
4
16
32


5.0*1e-6
Ускорение
12
10
8
6
4
2
0
1
4
16
32


1.5*1e-6
Ускорение
30
25
20
15
10
5
0
1
2
3
4

Точность	Число процессор	Время работы	Ускорение	Ошибка
3.0*1e-5	1	0,151941	1	4.39941e-06
	4	0,0351478	4,32	2.53543e-06
	16	0,0626483	2,43	2.10131e-05
	32	0,0121141	12,54	3.97602e-06
5.0*1e-6	1	0,171923	1	4.39941e-06
	4	0,0344056	5,00	2.53543e-06
	16	0,0510162	3,37	1.92411e-06
	32	0,0161349	10,66	3.97602e-06
1.5*1e-6	1	3,05538	1	1.00631e-06
	4	2,64915	1,15	1.19073e-06
	16	1,47162	2,08	1.22252e-06
	32	0,116549	26,22	9.68072e-08

Хоть результаты и показывают какое-то ускорение и сам метод выглядит, так будто мы будем генерировать больше точек параллельно, а значит приходить к желаемому результату быстрее. Но на практике всё упиралось в генератор случайных чисел и его инициализацию. Запросто возникала ситуация, когда на одном процессе за пару итераций удавалось достигать заданой точности, при этом на большем числе процессов из-за другой последовательности и большей массивности точек точности удавалось достигать спустя десятки итераций, что в параллель даёт еще и значительно большее число сгенерированных точек.