Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

Отчёт по заданию в рамках курса «Суперкомпьютерное моделирование и технологии» Численное интегрирование многомерных функций методом Монте-Карло

Выполнил:
Федяшкин Максим Алексеевич
627 группа
Вариант 16

21 Октября 2022 г.

1 Математическая постановка задачи

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

$$I = \iiint\limits_G f(x,y,z)\,dx\,dy\,dz \tag{1}$$

Для 16 варианта: $f(x,y,z) = x^2y^2z^2,\ G = \{(x,y,z) : |x| + |y| \le 1,\ -2 \le z \le 2\}$, и интеграл принимает вид:

$$I = \iiint\limits_G x^2y^2z^2\,dx\,dy\,dz \tag{2}$$

2 Аналитическое решение

$$I = \iiint\limits_G x^2y^2z^2\,dx\,dy\,dz = 2\int_0^1\int_{x-1}^{1-x}\int_{-2}^{2} x^2y^2z^2\,dx\,dy\,dz = 2\left(\frac{z^3}{3}\right)\Big|_{-2}^{2}\int_0^1\int_{x-1}^{1-x} x^2y^2\,dx\,dy =$$

$$= \frac{32}{3}\int_0^1 x^2\left(\frac{y^3}{3}\right)\Big|_{x-1}^{1-x}dx = \frac{32}{9}\int_0^1 2x^2\,(1-x)^3\,dx = \frac{64}{9}\int_0^1 x^2 - 3x^3 + 3x^4 - x^5\,dx =$$

$$= \frac{64}{9}\left(\frac{x^3}{3} - \frac{3x^4}{4} + \frac{3x^5}{5} - \frac{x^6}{6}\right)\Big|_0^1 = \frac{64}{9}\left(\frac{1}{3} - \frac{3}{4} + \frac{3}{5} - \frac{1}{6}\right) = \frac{16}{135} \tag{3}$$

3 Численный метод

Пусть область G ограниченна параллелепипедом: $\Pi : \begin{cases} a_1 \le x \le b_1 \\ a_2 \le y \le b_2 \\ a_3 \le z \le b_3 \end{cases}$ (4)

Рассмотрим функцию: $F(x,y,z) = \begin{cases} f(x,y,z), & (x,y,z) \subset G \\ 0, & (x,y,z) \not\subset G \end{cases}$ (5)

Преобразуем искомый интеграл:

$$\iiint\limits_G f(x,y,z)\,dx\,dy\,dz = \iiint\limits_\Pi F(x,y,z)\,dx\,dy\,dz$$

$p_1(x_1,y_1,z_1), p_2(x_2,y_2,z_2), ...$ - случайные точки, равномерно распределённые в П.В качестве приближённого значения интеграла можно использовать выражение:

$$I \approx |\Pi|\frac{1}{n}\sum_{n=1}^{n} p_i \tag{6}$$

где $|\Pi|$ - объём параллелепипеда П(4)

Для 16 варианта: $|\Pi| = 16,\ \Pi : \begin{cases} -1 \le x \le 1 \\ -1 \le y \le 1 \\ 2 \le z \le 2 \end{cases}$

4 Программная реализация

Полный код решения доступен по ссылке GitHub.

В основе алгоритма лежит вычисление частичных сумм $\sum_{i=(M-1)(N)}^{(M)(N)-1} F(p_i)$,

где $M \in [1; scale]$; $scale$ - кол-во частичных сумм изначально равное 1; N - кол-во суммируемых точек в частичной сумме (является фиксированным числом).

Тогда приблизительно вычисленный интеграл I_{scale} можно представить как:

$$I_{scale} = |\Pi| \frac{1}{scaleN} summ,\ summ = \sum_{M=1}^{scale} \sum_{(M-1)(N)}^{(M)(N)-1} F(p_i).$$

Пока не получена необходимая точность $epsilon$ по метрике $|I - I_{scale}|$: пересчитывается I_{scale}, затем $scale$ увеличивается на 1, после проверяется метрика. Если нужная точность достигнута, то I_{scale} искомое приближенное значение, а кол-во сгенерированных точек $(scale - 1) * N$. Иначе повторяем пересчет I_{scale}, увеличение $scale$, проверку метрики. Соотвественно данная часть алгоритма реализована посредством цикла $while$.

Для вычисления частичной суммы $summ$ на каждой итерации цикла $while$ генерируется N точек, которые хранятся в массиве $P[N]$, при каждой новой итерации, предыдущие точки затираются. При этом $summ$ сохраняет значение с предыдущий итерации, что позволяет сэкономить на вычислениях:

$$summ = summ + \sum_{i=0}^{N-1} F(p_i)$$

где p_i - точки сгенерированные на новой итерации, номер которой $scale$(на момент вычисления приближенного значения интеграла и частичной суммы $while$ на 1, потом $scale - 1$).

Точки геннирирутся посредство $rand()$ так, чтобы они лежали в Π. Для упрощения замеров для $srand$ выбирается всегда одинаковое неизменяемое число. Вычисление частисной суммы оформлено в виде цикла for, который проходит по всем точкам $P[N]$.

Важно отметить эвристическое наблюдение, что при увеличении N увеличивается и общее кол-во сгенерированных точек. Объяснить это можно тем, что прирост частичной суммы на каждой итерации оказывает большее влияние на значение приближенного интеграла и ему сложнее попасть в $epsilon$ окрестность точного значения.

4.1 Последовательная реализация

Последовательная реализация представлена в файле с именем: p2_seq.cpp;

В данном решении полностью реализован алгоритм описанный выше. Единственно стоит отметить, что наиболее эффективным является вычисление интеграла при $N = 1$;

4.2 Параллельная реализация

Параллельная реализация представлена в файле с именем: p2_par.c;

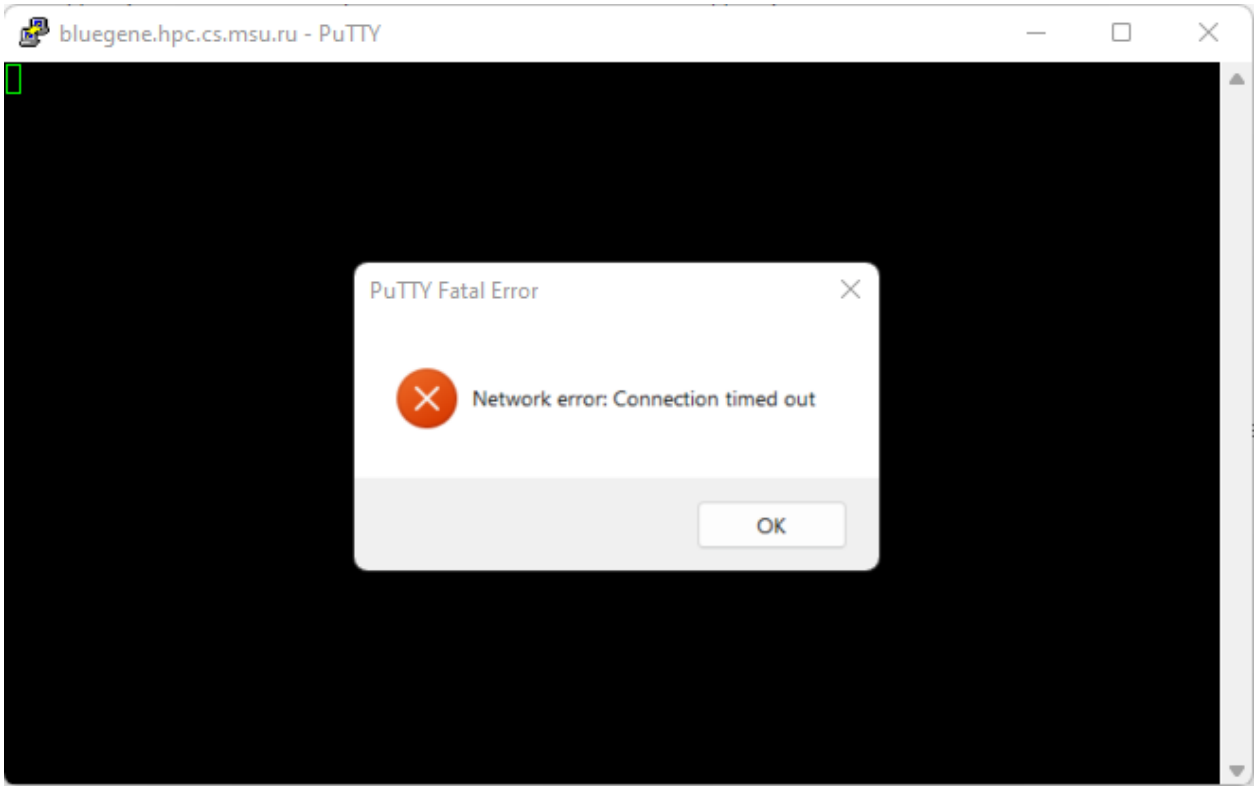
В данном решении представлена распараллеленая версия алгоритма описанного выше. Для увелечения скорости подсчет частичных сумм реализован на дополнительных процессорах. Пусть P - кол-во процессоров, тогда процессоров для подсчета частичных сумм $P-1$ (далее рабочие), 0 процесс считается мастером. Аналогично алгоритму кол-во генерируемых точек на итерации *while* задается через N. Определим переменную len как ближайшее целое большее или равное N и кратное $P-1$. Тогда каждый рабочий обрабатывает $\frac{len}{P-1}$ точек. Каждую итерацию мастер начинает с выполнения $MPI\_Scatterv()$, после которой начинает генерировать новую партию точек, не дожидаясь пока рабочие закончат свои вычисления, сгенерировав новые точки, мастер дожидается завершения вычислений рабочих и выполняет MPI_Reduce, после этого работа выполняется согласно общему алгоритму (вычисление частичной суммы, перерасчет приближенного интеграла, увеличение scale, проверка точности). На каждой итерации *while* мастер генерирует N точек, при этом если $len > N$, то в массив кладутся точки с нулевыми координатами (F(0, 0, 0) = 0). Рабочий не знает сколько итераций выполнит *while* мастера, поэтому использует $while(true)$. Рабочий на начало итерации находится в ожидании приема точек с помощью $MPI\_Scatterv$, получив точки он вычисляет сумму значений функции в точках, после встает в режим ожидания выполнения MPI_Reduce, когда reduce начинается новая итерация. Чтобы выйти из $while(true)$ рабочий должен получить набор точек, у которого 1 элемент имеет координату x равную заранее выбранному числу $B \notin [-1;1]$. Соответсвенно мастер после того как вычислил интеграл с нужной точностью должен послать еще один набор точек каждому рабочему, чтобы они завершили работу. При этом после получения B, рабочие сразу выходят из цикла и завершают работу.

5 Исследование масштабируемости

Для точности измерений зафиксируем параметры:

1. $N = 1000$ - кол-во точек генерируемых мастером на каждой итерации цикла *while*
2. $B = -123$ - магическое число для остановки рабочих.
3. $srand(z)$ - где $z = 1$ всегда одинаковая константа.

5.1 Blue Gene



5.2 Polus

Точность ε	Число MPI-процессов	Время работы программы(с)	Ускорение	Ошибка
$3.0 \cdot 10^{-5}$	2	0.012	1	0.0000078373
	4	0.012	1	0.0000078373
	16	0.022	0.54(54)	0.0000078373
	32	0. 02	0.6	0.0000078373
$5.0 \cdot 10^{-6}$	2	0.162	1	0.0000015638
	4	0.16	1.0125	0.0000015638
	16	0.214	0.757	0.0000015638
	32	0.236	0.68644	0.0000015638
$15.0 \cdot 10^{-7}$	2	0.17	1	0.0000006108
	4	0.162	1.0493	0.0000006108
	16	0.21	0.80952	0.0000006108
	30	0.224	0.75892	0.0000006108

Таблица 1: Таблица с результатами расчётов для системы Polus

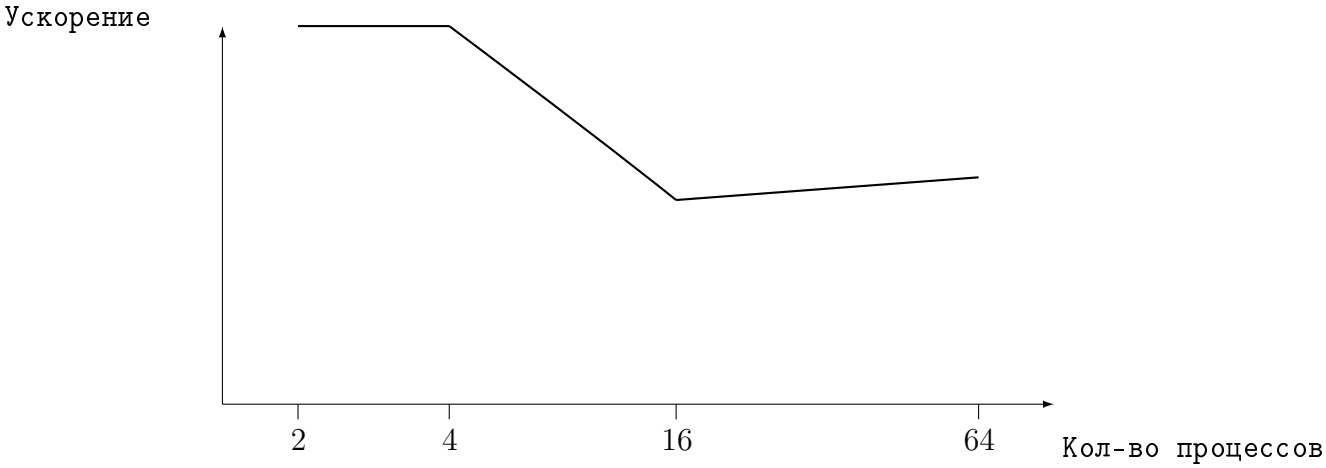


График 1: Polus, $\epsilon = 3.0e - 5$

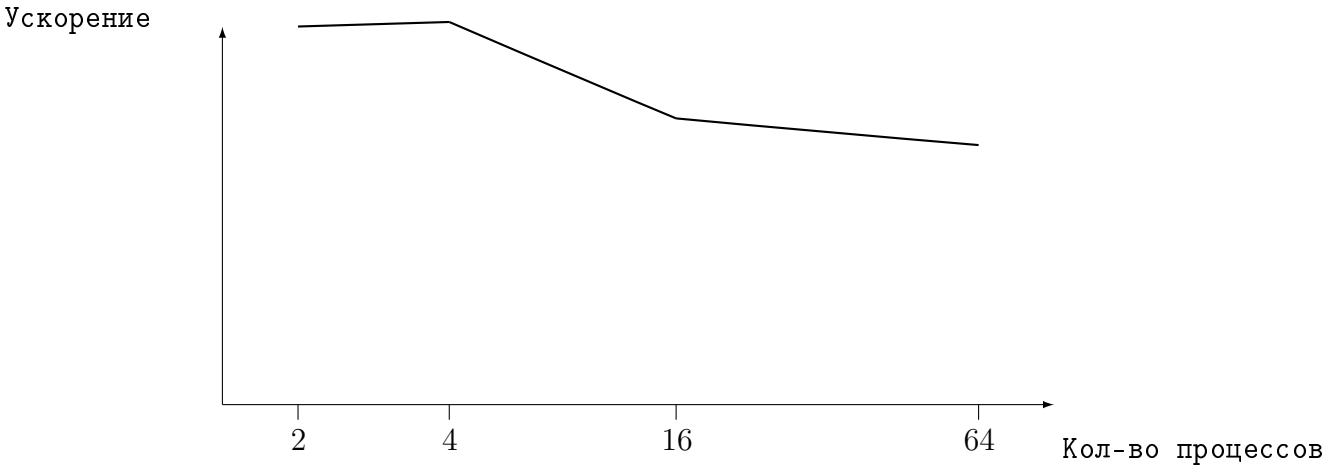


График 2: Polus, $\epsilon = 5.0e - 6$

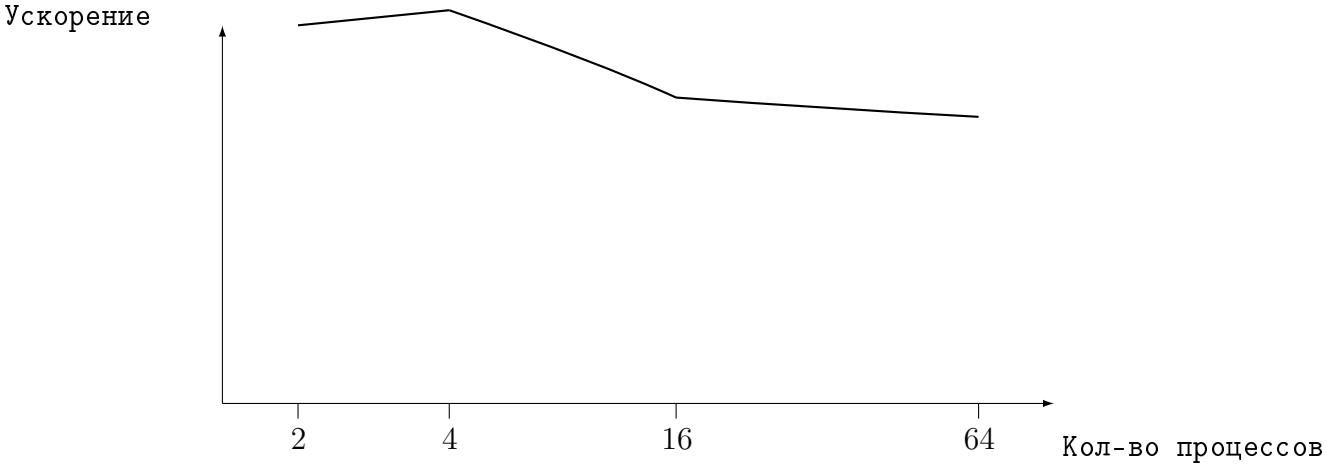


График 3: Polus, $\epsilon = 15.0e - 7$

6 Вывод

На основе результатов исследования можно сделать выводы, эффективность работы программы уменьшается при увеличении кол-ва процессов. При этом важно отметить, что благодаря фиксации параметров общее кол-во сгеннерированных точек и точность не меняются. Объяснить результат можно следующими наблюдениями:

1. Генерация точки занимает существенно больше времени, чем подсчет значения функции F. Из-за этого рабочие простаивают и ждут пока мастер сгенерирует точки для следующей рассылки

2. Операция рассылки и сбора очень дорогостоящие и, как показывает исследование на Polus, не окупают вычисление части частичной суммы на рабочем. Особенно это хорошо видно если увеличить вычислительную сложность F.

3. В замечание упоминается что при увеличении N, растет и кол-во точек необходимых для достижения нужной точности. При этом чтобы рабочие не делали бессмысленных вычислений необходимо чтобы $N > P - 1$. Что приводит к тому, что N необходимо брать достаточно большим.

4. Дополнительная потеря времени на пересылку дополнительных N точек и сбор времени работы каждого рабочего.

В результате можно сделать вывод, что предложенный алгоритм мастер-рабочий крайне плохо подходит к данной задаче, и последовательное решение эффективней.