Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Отчёт по заданию №2 в рамках курса «Суперкомпьютерное моделирование и технологии»

Выполнил: студент 621-ой группы
Рябченков Владимир Михайлович
Вариант: 15

Москва 2022

1 Описание задачи

Данная работа посвящена решению задачи вычисления многомерного интеграла методом Монте-Карло и исследованию параллельной MPI-программы, реализующей данный метод.

Входными данными для данной задачи является тройной интеграл с заданной областью интегрирования. Требуется:

- Выполнить программную реализацию метода Монте-Карло на языке C или C++ для вычисления заданного интеграла с использованием библиотеки MPI;
- Исследовать масштабируемость параллельной MPI-программы на следующих вычислительных системах ВМК МГУ: IBM Blue Gene/P, IBM Polus.

2 Математическая постановка задачи

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

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

3 Численный метод решения задачи

В данном разделе представлено описание метода Монте-Карло для численного интегрирования.

Пусть область G ограничена параллелепипедом: $\Pi = \begin{cases} a_1 \le x \le b_1, \\ a_2 \le y \le b_2, \\ a_3 \le z \le b_3 \end{cases}$

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:

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

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

$$I \approx |\Pi| \cdot \frac{1}{n} \sum_{i=1}^{n} F(p_i) \tag{3.1}$$

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

4 Исходный интеграл и его точное значение

$$I = \iiint\limits_G \sqrt{y^2 + z^2}\, dx\, dy\, dz \tag{4.1}$$

где область $G = \{(x, y, z) : \ 0 \leq x \leq 2,\ y^2 + z^2 \leq 1\}$

$$\begin{aligned} I = \{x = x,\ y = r \cdot cos\phi,\ z = r \cdot sin\phi,\ J = r\} = \\ = \int_0^2 dx \int_0^{2\pi} d\phi \int_0^1 r^2 dr = \\ = \int_0^2 dx \int_0^{2\pi} \frac{1}{3}\, d\phi = \\ = \int_0^2 \frac{2\pi}{3}\, dx = \frac{4}{3}\pi = 4.1887902048 \end{aligned}$$

5 Программная реализация

В рамках данной работы была реализована параллельная MPI-программа, которая вычисляет интеграл 4.1 с помощью метода Монте-Карло, описанного в разделе 3 (программа доступна по ссылке https://github.com/vultar150/parallel-dz2.git). 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 приближённым значением, полученным методом Монте-Карло, и точным значением I, вычисленным аналитически 4.

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 точным 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 качестве варианта распараллеливания используется парадигма «мастер-рабочие»: один из процессов («мастер») генерирует случайные точки и передаёт каждому из остальных процессов («рабочих») отдельный, предназначенный для него, набор сгенерированных случайных точек.

В данной программе процессом «мастером» является процесс с номером 0. Передача точек процессам-рабочим осуществляется с помощью функции MPI_Scatterv. В про-

грамме предварительно происходит инициализация некоторых параметров для данной функции (sendcounts и displs).

Процессы-рабочие вычисляют свою часть суммы по формуле 3.1. Затем процесс-мастер с помощью операции редукции MPI_Reduce вычисляет общую сумму. После чего вычисляется ошибка (разность между посчитанным значением и точным значением I, вычисленным аналитически). В случае если ошибка выше требуемой точности ϵ, которую подали на вход программе, то генерируются дополнительные точки и расчёт продолжается.

На каждой итерации цикла процесс-мастер генерирует N точек, где N задано в виде параметра. Каждый процесс-рабочий получает свою часть точек $\frac{N}{P}$, где P — количество процессов-рабочих. Если N не кратно P, то первые $N \ mod \ P$ процессов-рабочих получают на одну точку больше, чем остальные.

После операции редукции MPI_Reduce и вычисления ошибки, процесс-мастер проверяет достигнута ли заданная точность или нет. Если точность достигнута, то флаг needBreak = 1, иначе needBreak = 0. После чего мастер-процесс отправляет значение заданного флага процессам-рабочим с помощью операции MPI_Bcast. Цикл продолжается до тех пор, пока флаг needBreak = 0.

После завершения основного цикла, процесс-мастер выводит результаты.

6 Результаты

Результаты запусков программы на системе Polus для различного числа MPI-процессов и различных значений входного параметра ϵ приведены в таблице 1. Важно отметить, что результаты запусков программы на системе Blue Gene/P отсутствуют, так как при попытке соединения по SSH к данной системе возникает ошибка, представленная на рис. 1.

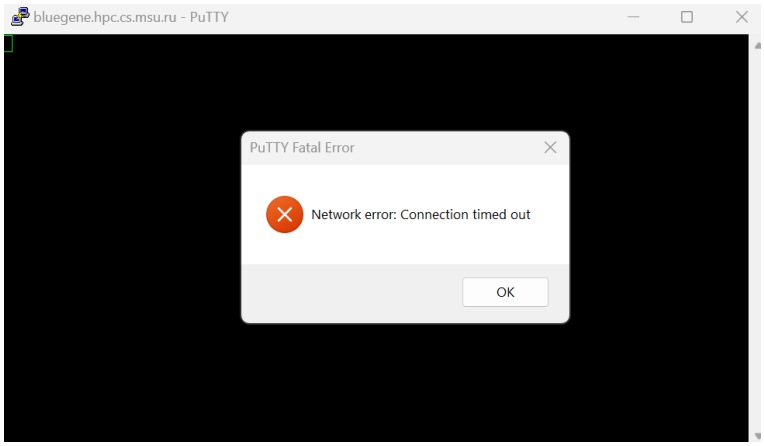

Рис. 1. Ошибка при подключении к системе Blue Gene/P

Таблица 1. Таблица с результатами расчётов для системы Polus

Точность ϵ	Число MPI-процессов	Время работы программы (с)	Ускорение	Ошибка
$3.0 * 10^{-5}$	2	0.117	1	$0.58 \cdot 10^{-5}$
	4	0.144	0.81	$0.58 \cdot 10^{-5}$
	16	0.098	1.19	$0.58 \cdot 10^{-5}$
	60	0.134	0.87	$0.58 \cdot 10^{-5}$
$5.0 * 10^{-6}$	2	10.425	1	$3.4 \cdot 10^{-6}$
	4	9.579	1.09	$3.4 \cdot 10^{-6}$
	16	8.705	1.20	$3.4 \cdot 10^{-6}$
	60	7.886	1.32	$3.4 \cdot 10^{-6}$
$1.5 * 10^{-6}$	2	10.680	1	$0.33 \cdot 10^{-6}$
	4	9.337	1.14	$0.33 \cdot 10^{-6}$
	16	8.747	1.22	$0.33 \cdot 10^{-6}$
	60	8.339	1.28	$0.33 \cdot 10^{-6}$

Графики зависимости ускорения от количества процессов для различных значений ϵ на системе Polus представлены рисунках 2–4.

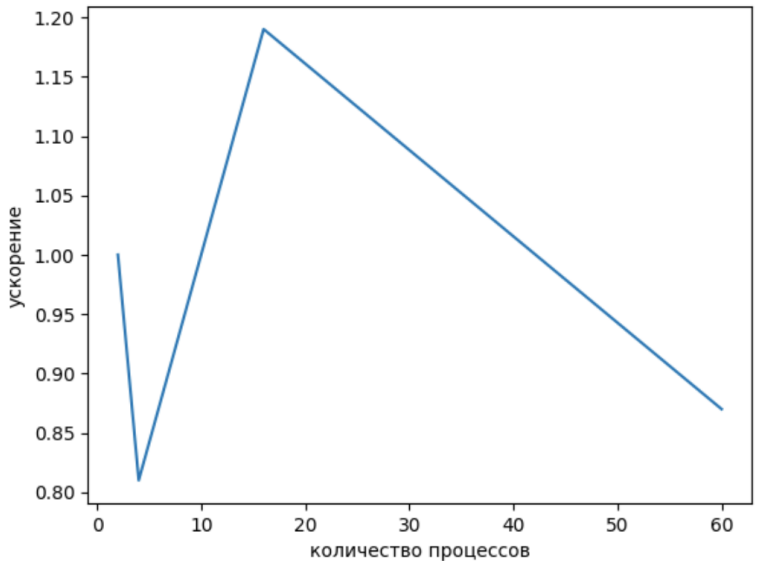

Рис. 2. Polus: точность $\epsilon = 3.0 \cdot 10^{-5}$

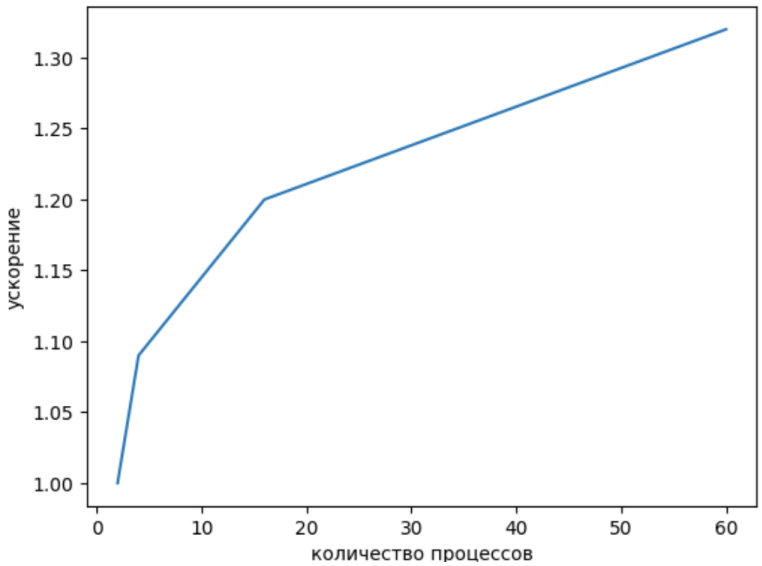

Рис. 3. Polus: точность $\epsilon = 5.0 \cdot 10^{-6}$

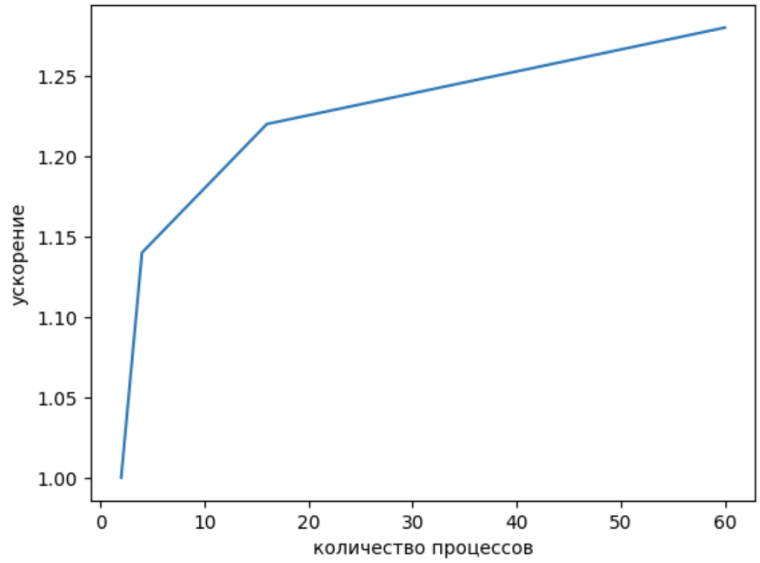

Рис. 4. Polus: точность $\epsilon = 1.5 \cdot 10^{-6}$

7 Выводы

Время работы параллельной MPI-программы при точности $\epsilon = 3.0 \cdot 10^{-5}$ может увеличиваться с ростом количества процессов. Это связано с тем, что время работы программы при данной точности не превышает 0.144 с, что соизмеримо со временем, затрачиваемым на накладные расходы (например, обмен данными между процессами).

Если рассматривать точность $\epsilon = 5.0 \cdot 10^{-6}$ или $\epsilon = 1.5 \cdot 10^{-6}$ (рис. 3, 4), то ускорение увеличивается с ростом количества процессов. Однако максимальное значение ускорения составляет 1.32 при 60-ти процессах, что показывает незначительный прирост скорости работы программы. Это можно объяснить тем, что последовательная часть программы (генерирование точек на процессе-мастере) занимает около 70% (исходя из проведенных экспериментов). По закону Амдала $S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$, где S_p — ускорение, p — количество процессов, α — доля последовательных вычислений. При $\alpha = 0.7$

максимальное значение ускорения составляет $S_p \approx 1.47$, что примерно и соответствует результатам представленным на рисунках 3, 4.