Отчёт по практическому заданию (2) в рамках курса «Суперкомпьютерное моделирование и технологии»

Вариант 5

Никифоров Никита Игоревич, гр. 621

nickiforov.nik@gmail.com

Октябрь 2022

1 Задача

Необходимо реализовать численный метод Монте-Карло нахождения значения интеграла в заданной области. Для реализации метода предлагается использовать языки программирования C/C++, с использованием библиотеки параллельного вычисления MPI.

Необходимо провести исследование реализованного численного метода для заданного интеграла, области и точности на параллельных вычислительных системах ВМК МГУ: IBM Blue Gene/P, IBM Polus

2 Математическая постановка задачи

Пусть функция $f(x, y, z) = x^3 * y^2 * z$ — непрерывна в ограниченной замкнутой области $G \subset \mathbb{R}^3$. Требуется вычислить определённый интеграл:

$$I = \iiint_G f(x, y, z)\, dxdydz = \iiint_G x^3 * y^2 * z\, dxdydz, \tag{1}$$

где область $G = (x, y, z) : -1 \le x \le 0,\ -1 \le y \le 0,\ -1 \le z \le 0$

3 Численный метод решения задачи (Монте-Карло)

Пусть заданная область G ограниченна параллелепипедом $P : a_1 \le x \le b_2,\ a_2 \le y \le b_2,\ a_3 \le z \le b_3$ Рассмотрим функцию определённую на параллелепипеде P:

$$F(x, y, z) = \begin{cases} f(x, y, z), & (x, y, z) \in G \\ 0, & (x, y, z) \notin G \end{cases} \tag{2}$$

Преобразуем искомый интеграл 1 - подставив функцию 2.

$$I = \iiint_G f(x, y, z)\, dxdydz = \iiint_P F(x, y, z)\, dxdydz, \tag{3}$$

Пусть $p_1 : (x_1, y_1, z_1),\ p_2 : (x_2, y_2, z_2), \ldots, p_n : (x_n, y_n, z_n)$ — случайные точки равномерно распределённые по области P. Тогда в качестве приближённого значения интеграла 3 предлагается использовать выражение:

$$I \approx |P| * \frac{1}{n} * \sum_{i=1}^{n} F(p_i), \tag{4}$$

где $|P|$ — объём параллелепипеда P, $|P| = (b_1 - a_1)(b_2 - a_2)(b_3 - a_3)$.

4 Аналитическое решение задачи

Найдём аналитически интеграл 1:

$$\begin{gathered} I = \iiint_G x^3 * y^2 * z\, dxdydz = \int_{-1}^{0} dx \int_{-1}^{0} dy \int_{-1}^{0} x^3 * y^2 * z\, dz = \\ \frac{x^4}{4}\Big|_{-1}^{0} \frac{y^3}{3}\Big|_{-1}^{0} \frac{z^2}{2}\Big|_{-1}^{0} = \frac{1}{24} * x^4 y^3 z^2\Big|_{-1}^{0} = \frac{1}{24} \approx 0.0416(6) \end{gathered} \tag{5}$$

5 Программная реализация

Для программной реализации предложенного метода используется язык C++, а также библиотека параллельного программирования MPI. Программа разделяется на независимые процессы, каждый из которых генерирует свой набор из 100 случайных точек, и считает свою частичную сумму. Принципиально алгоритм состоит из следующих шагов:

1. Каждый процесс генерируют набор из 100 случайных точек.
2. Каждый процесс считает сумму $S_j = \sum_{i=1}^{n} F(p_i)$, где j - ранк процесса, $n = 100$ - количество случайных точек, генерируемых процессом за итерацию.
3. С использованием функции MPI_AllReduce, происходит вычисление суммы по всем процессам: $S_{global} = \sum_{j=0}^{size-1} S_j.$, где $size$ - количество независимых MPI процессов.
4. В каждом процессе вычисляется приближённое значение интеграла: $I_{global} = |P| * \frac{1}{n} * S_{global}$.
5. Вычисляется ошибка: $err = |I_{global} - I_{analitical}|$.
6. Если ошибка меньше заданной вычисление прекращается, если больше, то алгоритм повторяется.

```
std::pair<double, int64_t>  monte_carlo(std::mt19937 &gen,
                                        std::uniform_real_distribution<> &dis,
                                        double eps, double solution, double volume,
                                        double (*func)(Point)) {
    double integ = 0.0;
    double err = std::abs(integ - solution);
    double global_sum = 0.0;
    int64_t point_counter = 0;

    while (err > eps) {
        double local_sum = 0;//global_sum;

        for (int i = 0; i < POINTS_PER_CPU; i++) {
            local_sum += func(Point(dis, gen));
        }

        double now_sum = 0.0;
        MPI_Allreduce(&local_sum, &now_sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
        global_sum += now_sum;

        point_counter += POINTS_PER_CPU * size;
        integ = volume * global_sum / (point_counter + 0.0);
        err = std::abs(integ - solution);

    }

    return {integ, point_counter};
}
```

6 Исследование программной реализации

Для исследования использовалась система Polus и домашний компьютер. Характеристики узла системы параллельного вычисления Polus (на данный момент имеет в работе 3 узла):

- 2 десятиядерных процессора IBM POWER8 (каждое ядро имеет 8 потоков) всего 160 потоков
- Общая оперативная память 256 Гбайт (в узле 5 оперативная память 1024 Гбайт) с ECC контролем
- 2 x 1 ТБ 2.5” 7K RPM SATA HDD
- 2 x NVIDIA Tesla P100 GPU, 16Gb, NVLink
- 1 порт 100 ГБ/сек

Характеристики домашнего PC:

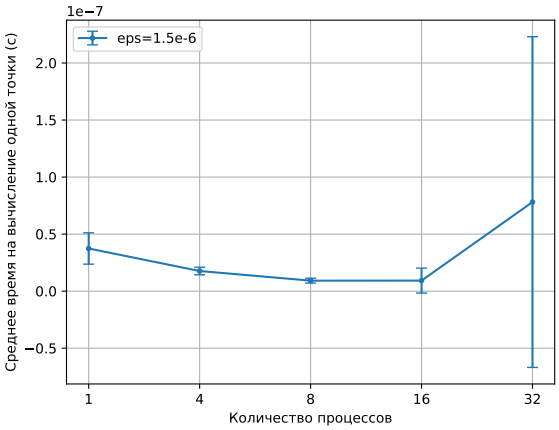
- 1 шестнадцати ядерный процессор AMD Ryzen 5950X, всего 32 потока на частоте 4.4Ghz
- Оперативная память 32 Гбайт
- SSD NVMe Samsung 500 ГБ
- AMD RX 5600XT

Для каждого значения требуемой точности $eps = \{1.5e-6, 5.0e-6, 3.0e-5\}$ и каждого количества процессов $cpus = \{1, 4, 8, 16, 32\}$ проводилось по 10 запусков, значения усреднялись.

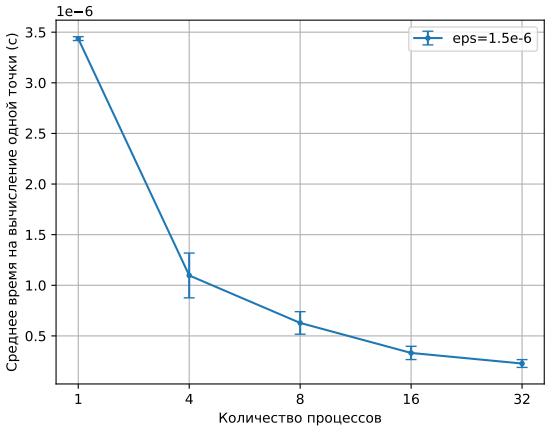
Ускорение считалось как отношение времени вычисления одной точки при разном количестве MPI процессов, что позволяет исключить случайность процесса при вычислении ускорения. А также как отношение общего времени выполнения на одном MPI-процессе ко времени вычисления на заданном количестве MPI-процессов. Составим таблицу для системы Polus:

Таблица 1: Результаты исследования на машине Polus

Точность	**число MPI процессов**	**Время работы (с)**	**Ускорение по времени**	**Ускорение**	**Ошибка**
$3.0 * 10^{-5}$	1	0.024625	1	1	0.000012
$3.0 * 10^{-5}$	4	0.021700	1.13	3.09	0.000013
$3.0 * 10^{-5}$	8	0.053036	0.46	4.83	0.000018
$3.0 * 10^{-5}$	16	0.081705	0.30	7.56	0.000019
$3.0 * 10^{-5}$	32	0.093178	0.26	12.27	0.000017
$5.0 * 10^{-6}$	1	0.745260	1	1	0.000003
$5.0 * 10^{-6}$	4	0.068947	10.8	3.09	0.000002
$5.0 * 10^{-6}$	8	0.345431	2.15	4.76	0.000003
$5.0 * 10^{-6}$	16	0.043465	17.14	9.84	0.000003
$5.0 * 10^{-6}$	32	0.072755	10.24	14.28	0.000003
$1.5 * 10^{-6}$	1	27.415805	1	1	8.013453e-07
$1.5 * 10^{-6}$	4	2.434363	11.26	3.13	9.394432e-07
$1.5 * 10^{-6}$	8	0.223442	122.69	5.46	8.698047e-07
$1.5 * 10^{-6}$	16	4.428387	6.19	10.35	7.874600e-07
$1.5 * 10^{-6}$	32	0.192503	142.41	15.12	6.611632e-07

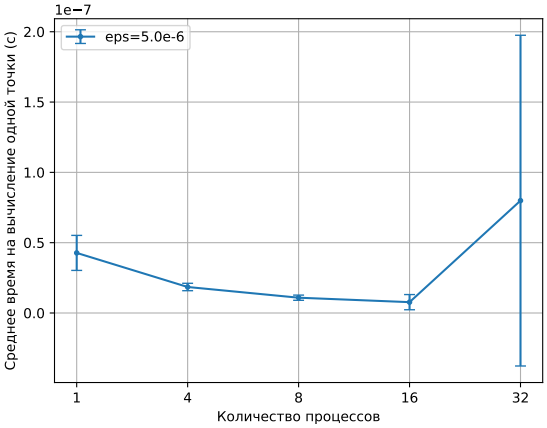


(a) На домашнем компьютере

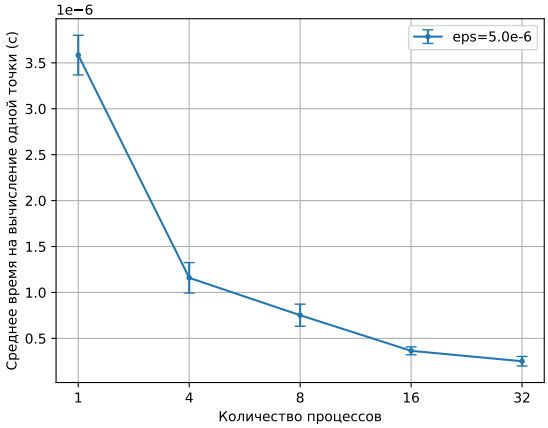


(b) На системе Polus

Рис. 1: График зависимость времени вычисления одной точки от количества процессов MPI

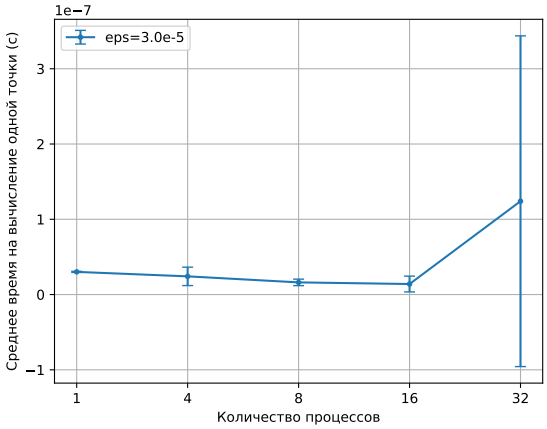


(a) На домашнем компьютере

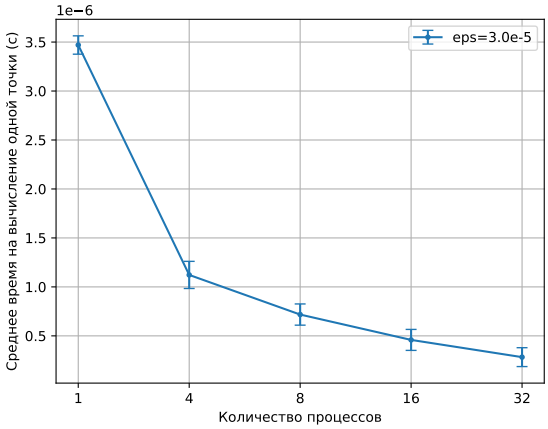


(b) На системе Polus

Рис. 2: График зависимость времени вычисления одной точки от количества процессов MPI

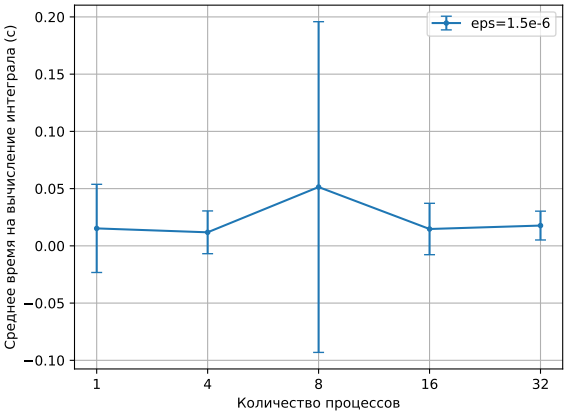


(a) На домашнем компьютере

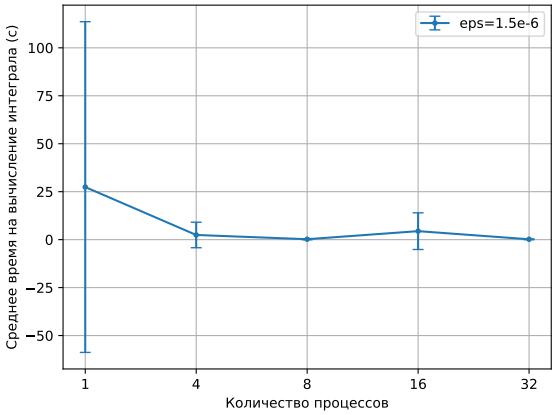


(b) На системе Polus

Рис. 3: График зависимость времени вычисления одной точки от количества процессов MPI

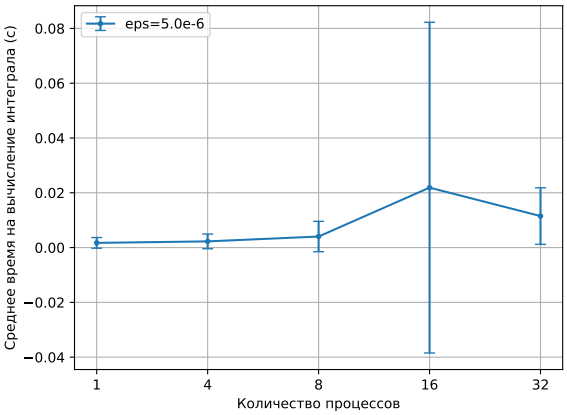


(a) На домашнем компьютере

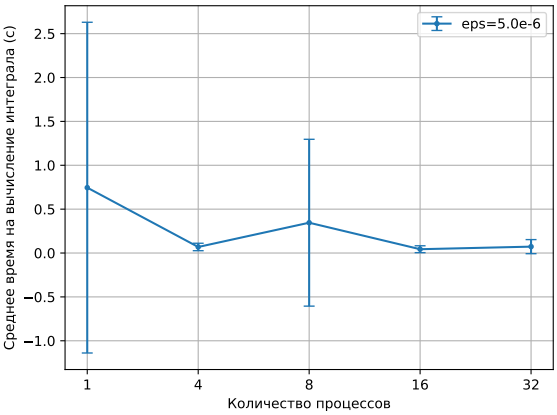


(b) На системе Polus

Рис. 4: График зависимость времени вычисления значения интеграла от количества процессов MPI

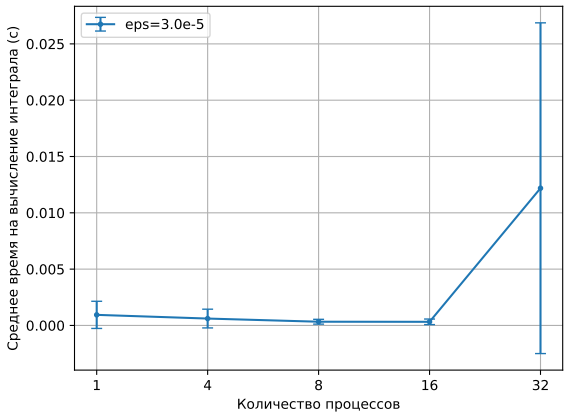


(a) На домашнем компьютере

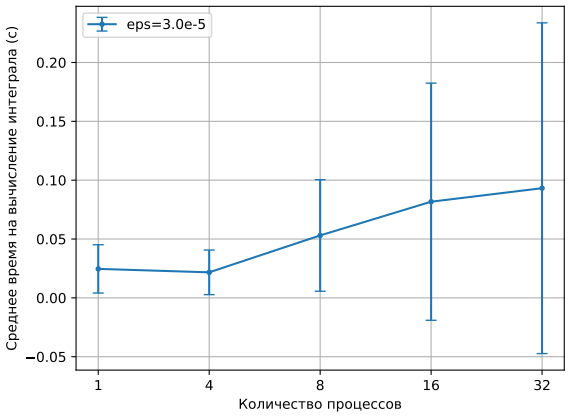


(b) На системе Polus

Рис. 5: График зависимость времени вычисления значения интеграла от количества процессов MPI



(a) На домашнем компьютере



(b) На системе Polus

Рис. 6: График зависимость времени вычисления значения интеграла от количества процессов MPI