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

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

Выполнила:
Лапенко Юлия Андреевна
Группа 627
Вариант 1

1 Описание условия

С помощью метода Монте-Карло требуется численно вычислить интеграл:

$$I = \iiint_G xy^2z^3 dxdydz,$$

где область G ограничена поверхностями $z = xy$, $y = x$, $x = 1$, $z = 0$.

2 Описание метода решения

Посчитаем интеграл аналитически:

$$I = \int_0^1 xdx \int_0^x y^2dy \int_0^{xy} z^3dz = \frac{1}{364}$$

Пусть область G ограничена параллелепипедом $\Pi : \{0 \le x \le 1, 0 \le y \le x, 0 \le z \le xy\}$. Рассмотрим функцию $F(x,y,z) = \begin{cases} xy^2z^3, & (x,y,z) \in G \\ 0, & (x,y,z) \notin G \end{cases}$. Приближенно будем вычислять интеграл следующим образом:

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

где n – количество точек.

Опишем программную реализацию. Пусть на каждом процессоре независимо генерируется по $n = 1000$ точек, пока нужная точность ε не будет достигнута. На каждом процессоре будет инициализироваться генератор псевдослучайных чисел:

```
std::srand(std::time(NULL));
```

Каждый процессор генерирует по 1000 случайных точек p_i, вычисляет значение $F(p_i)$ в каждой такой точке и суммирует полученные значения. Процессор с рангом 0 собирает частичные суммы по процессорам с помощью функции `MPI_Reduce`, а затем делит общую сумму на количество сгенерированных точек. Если желаемая точность достигнута, процесс с рангом 0 рассылает с помощью `MPI_Bcast` всем процессам значение 0 (прервать вычисления), иначе – 1 (сгенерировать еще по 1000 точек и продолжить).

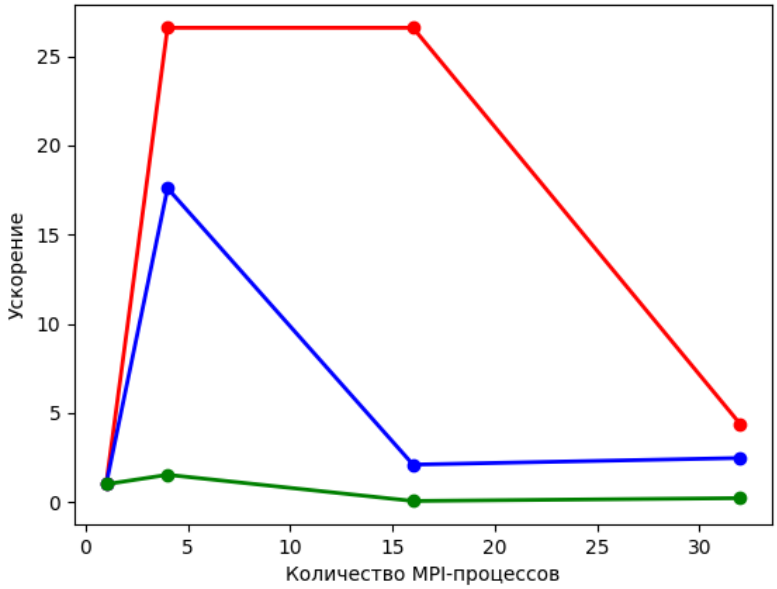
Время работы берется максимальным из отдельных времен работы процессоров: каждый процессор отсылает с помощью `MPI_Gather` свое время процессору с рангом 0.

3 Исследование масштабируемости

Поскольку доступ к Bluegene в момент написания отчета отсутствует, будем запускать код только на Polus.

ε	Число процессов	T, сек	S	Ошибка	Кол-во точек
$3.0 \cdot 10^{-5}$	1	0.0346	1	$2.6 \cdot 10^{-5}$	306000
	4	0.0013	26.6	$1.7 \cdot 10^{-5}$	28000
	16	0.0013	26.6	$2.3 \cdot 10^{-5}$	16000
	32	0.0079	4.37	$1.97 \cdot 10^{-5}$	512000
$5.0 \cdot 10^{-6}$	1	0.0512	1	$3.6 \cdot 10^{-6}$	458000
	4	0.0029	17.6	$2.1 \cdot 10^{-6}$	100000
	16	0.0243	2.1	$4.43 \cdot 10^{-7}$	848000
	32	0.0207	2.47	$3.1 \cdot 10^{-5}$	1248000
$1.5 \cdot 10^{-6}$	1	0.0216	1	$7.8 \cdot 10^{-7}$	190000
	4	0.0141	1.53	$1.3 \cdot 10^{-6}$	384000
	16	0.3329	0.06	$1.4 \cdot 10^{-7}$	37216000
	32	0.0975	0.22	$3.97 \cdot 10^{-7}$	7072000

График, показывающий зависимость ускорения от количества процессов (красным показана точность $\varepsilon = 3.0 \cdot 10^{-5}$, синим – $\varepsilon = 5.0 \cdot 10^{-6}$, зеленым – $\varepsilon = 1.5 \cdot 10^{-6}$):



Как видно, программа не очень хорошо масштабируется. Это связано с тем, что точки генерируются на каждом процессе независимо и случайно, что делает поведение недетерменированным: мы не можем знать наверняка, как быстро будет достигнута желаемая точность. Поэтому в некоторых случаях получившееся ускорение превышает количество процессов: нам повезло быстро достичь нужной точности.