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

Задание 1. Методы Монте-Карло

Отчёт
О выполненном задании

Выполнил:
студент 523 группы
Латыпов Ш. И.
latypovshamil2001@gmail.com

Москва
2023

Описание задачи

Требуется написать параллельную программу, реализующую метод Монте-Карло для модели случайных блужданий. Программа должна принимать на вход (через аргументы командной строки) пять параметров:

- a, b — границы интервала,
- x — начальная позиция,
- p — вероятность перехода частицы вправо,
- N — число частиц.

Результатом работы программы должны быть два файла. В файле output.txt должны быть записаны полученные данные (вероятность достижения состояния b и среднее время жизни одной частицы t). В файле stat.txt должно быть сохранено время T работы основного цикла программы с указанием всех параметров запуска, в том числе параметр P — число параллельных процессов.

Код программы

```
#include <mpi.h>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iostream>

using namespace std;

double frand(double a, double b) {
  return a + (b - a) * (rand() / double(RAND_MAX));
}

int do_walk(int a, int b, int x, double p, double& t) {
  int step = 0;
  while (x > a && x < b) {
    if (frand(0, 1) < p)
      x += 1;
    else
      x -= 1;
    t += 1.0;
    step += 1;
  }
  return x;
}

void run_mc(int a, int b, int x, double p, int N) {
  int numprocs, rank;
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  srand(2 * (rank + 1));
```

```
  double t = 0.0;
  double w = 0.0;

  int old_N = N;
  N = N / numprocs + ((rank < N % numprocs) ? 1 : 0);

  double start, stop, all_time;
  start = MPI_Wtime();
  for (int i = 0; i < N; i++) {
    int out = do_walk(a, b, x, p, t);
    if (out == b)
      w += 1;
  }
  stop = MPI_Wtime();
  all_time = stop - start;

  double global_t, global_w;
  double max_time;

  MPI_Reduce(&t, &global_t, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
  MPI_Reduce(&w, &global_w, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
  MPI_Reduce(&all_time, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);

  MPI_Barrier(MPI_COMM_WORLD);
  if (rank == 0) {
    ofstream f("output.txt");
    f << global_w / old_N << " " << global_t / old_N << endl;
    f.close();

    ofstream s("stat.txt");
    s << "Time = " << max_time << endl;
    s << "Num of Processes = " << numprocs << endl;
    s << "mpisbmit.pl -p " << numprocs << " main -- " << a << " " << b << " "
      << x << " " << p << " " << old_N << endl;
    s.close();
  }
  MPI_Finalize();
}

int main(int argc, char** argv) {
  MPI_Init(&argc, &argv);

  int a = atoi(argv[1]);
  int b = atoi(argv[2]);
  int x = atoi(argv[3]);
  double p = atof(argv[4]);
  int N = atoi(argv[5]);

  run_mc(a, b, x, p, N);

  return 0;
}
```

Результаты работы программы

1. Фиксированное значение P (количество процессов), число N меняется

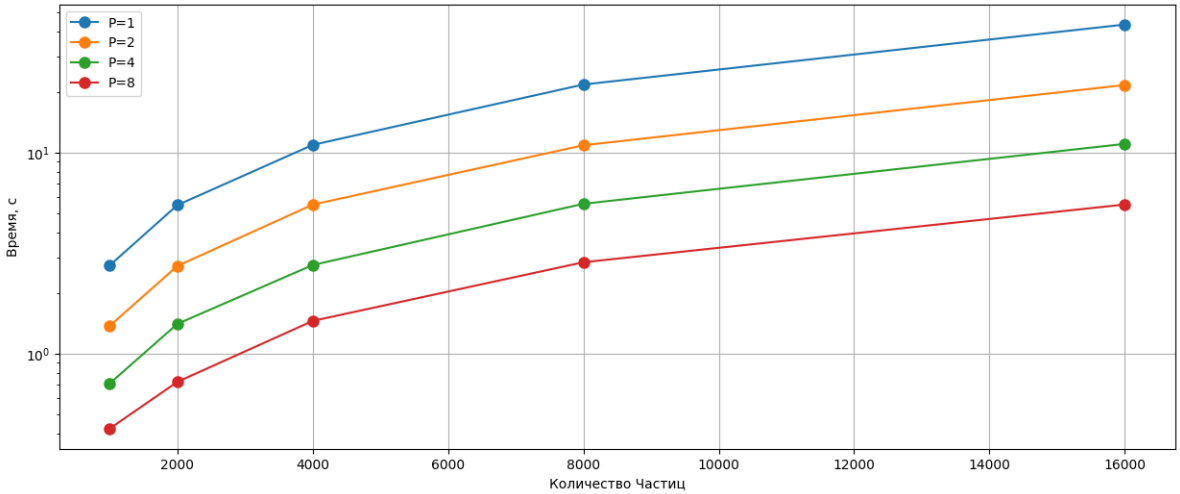

Рис. 1: Время работы

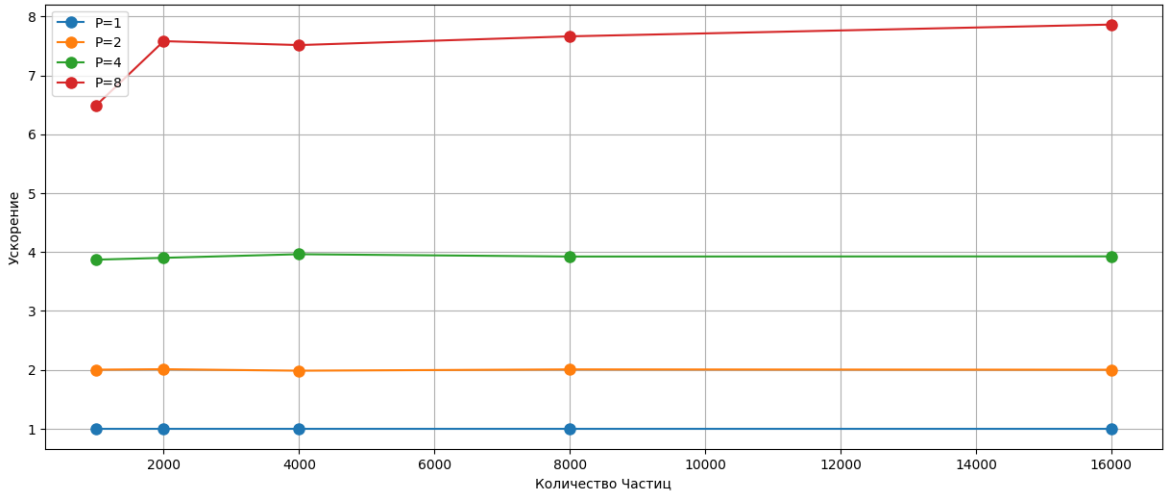

Рис. 2: Ускорение

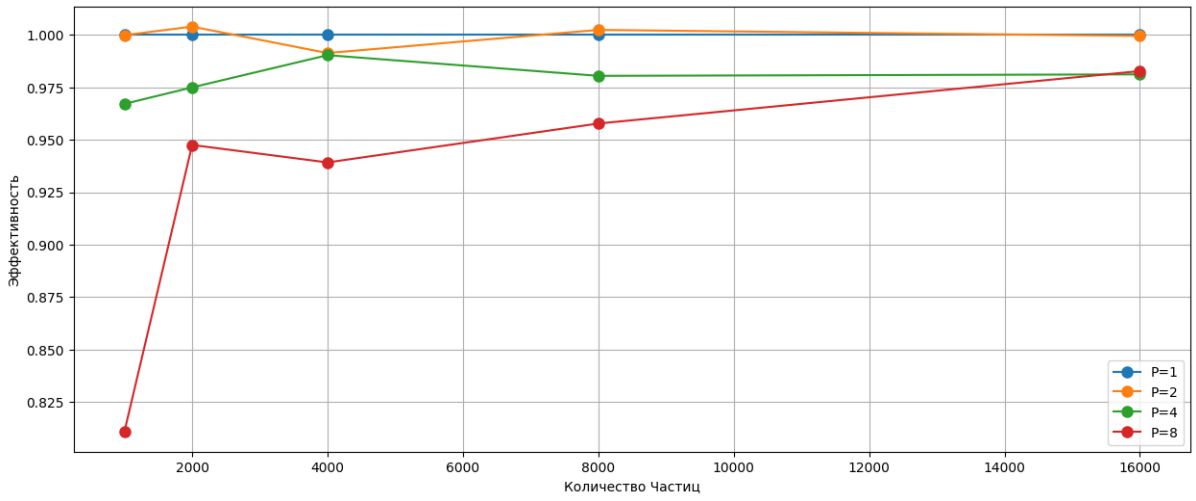

Рис. 3: Эффективность

2. Фиксированное значение N, число P меняется

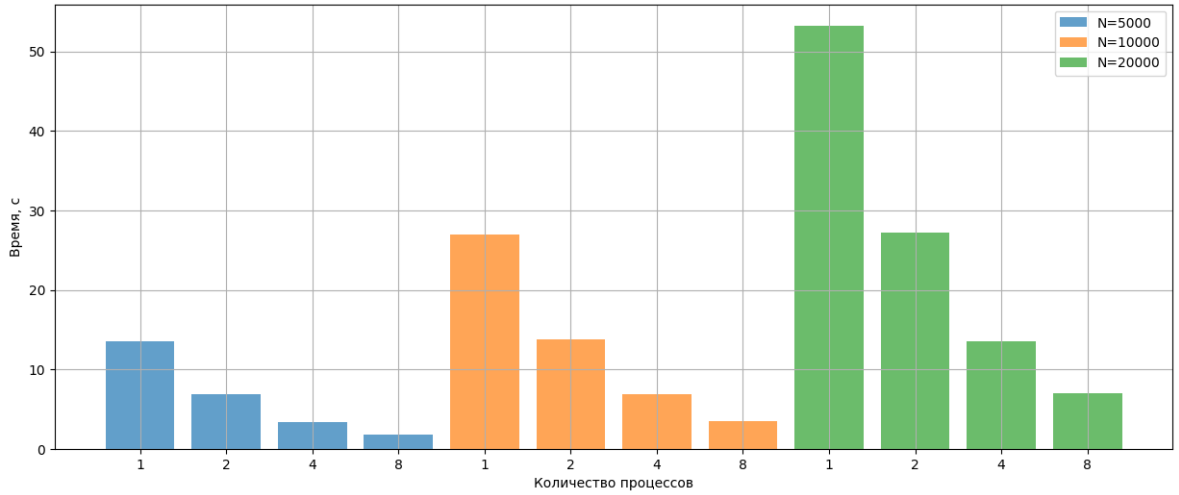

Рис. 4: Время работы

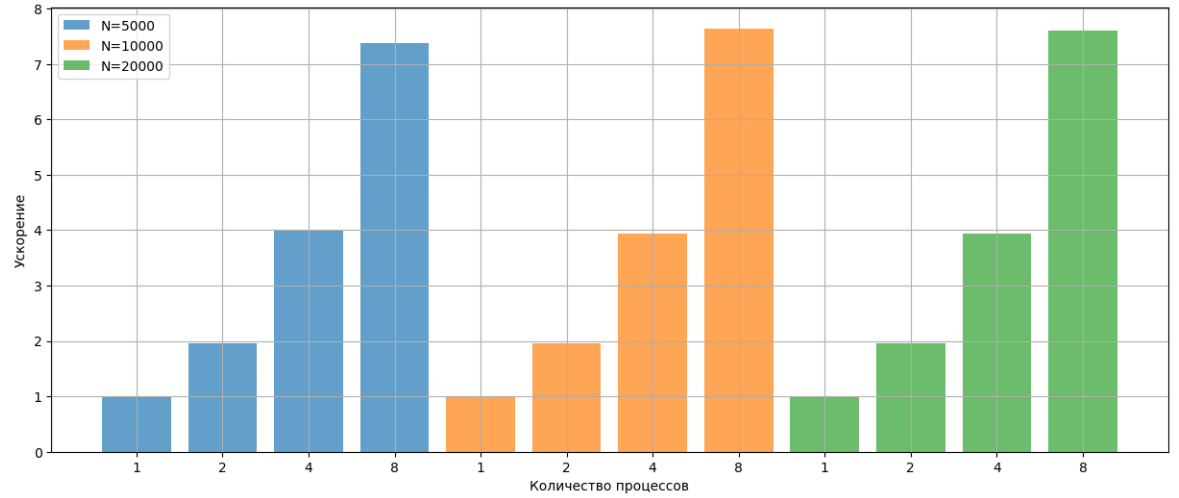

Рис. 5: Ускорение

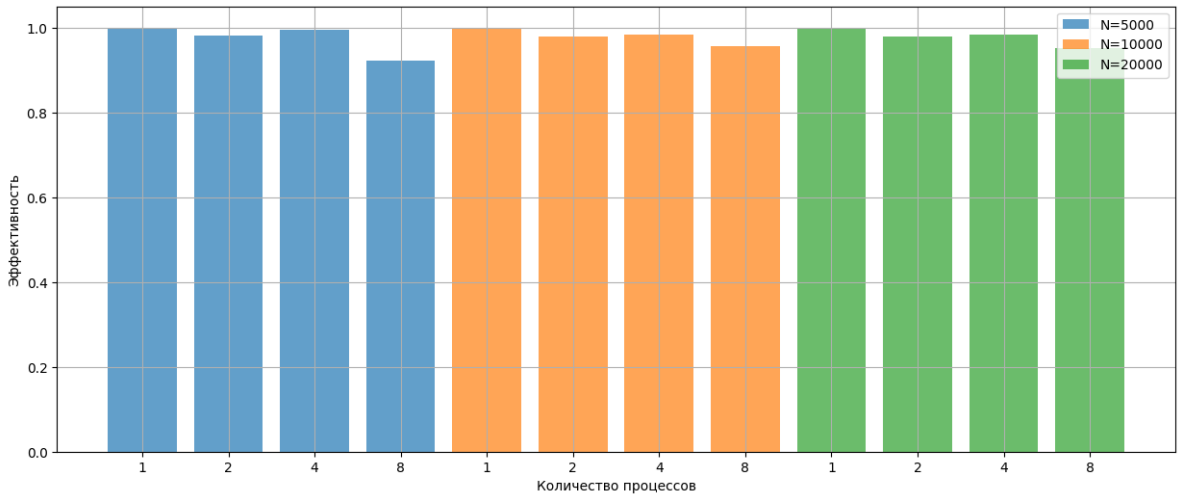

Рис. 6: Эффективность

3. Значение N = 1000P

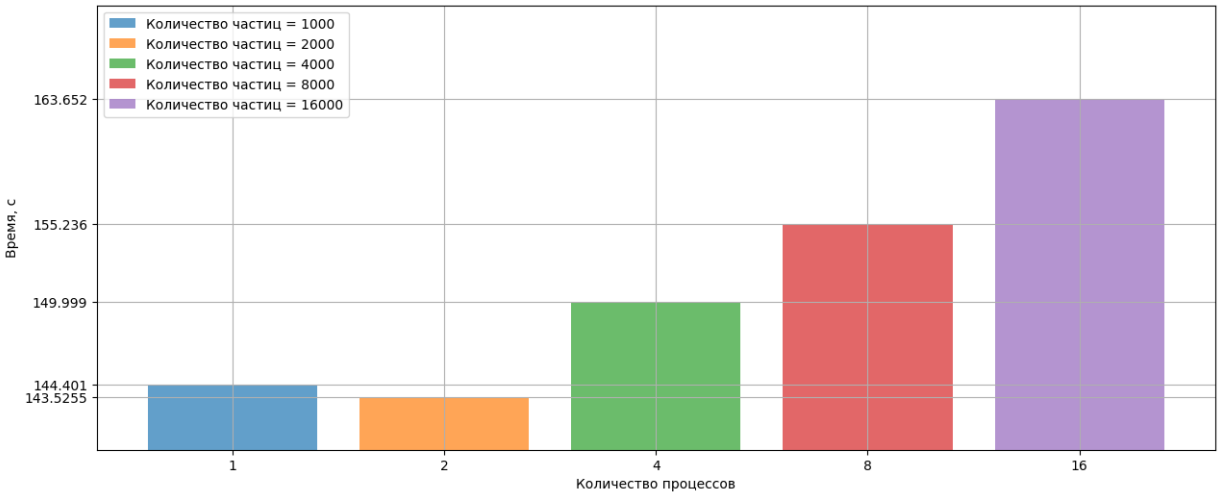

Рис. 7: Время работы

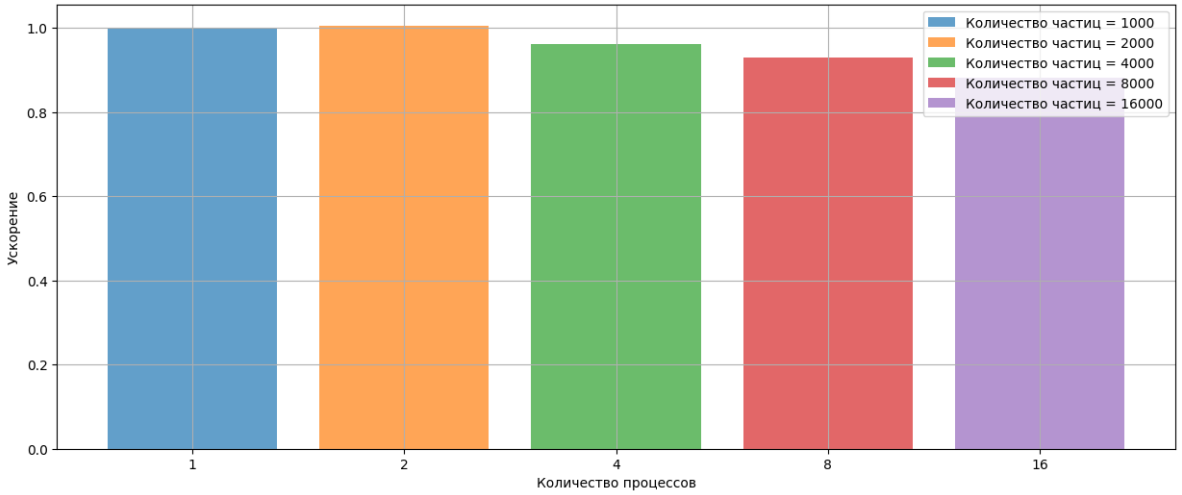

Рис. 8: Ускорение

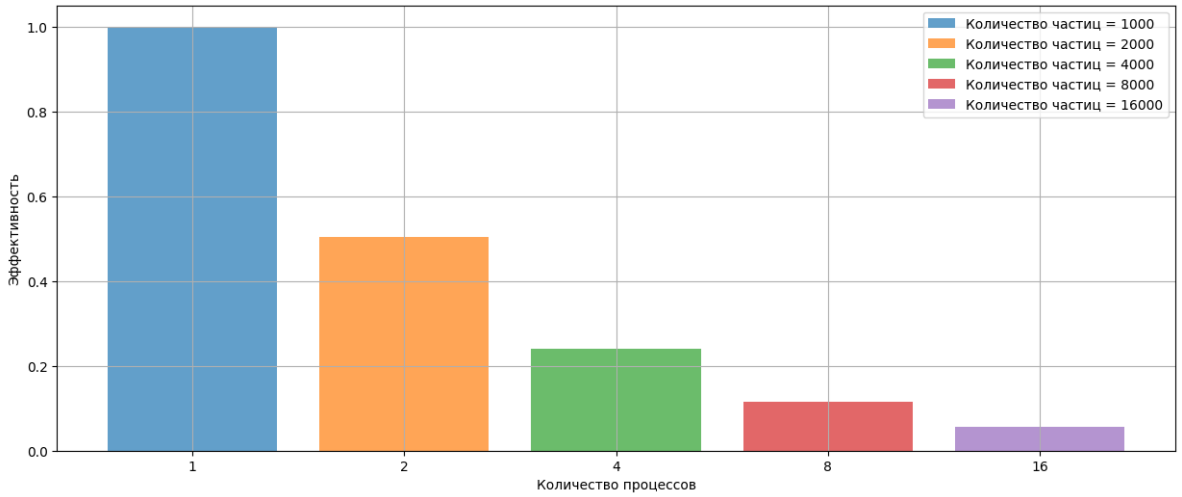

Рис. 9: Эффективность