Задание 1. Методы Монте-Карло

Студеникина Мария Александровна
studenikina.marya@yandex.ru

T(N) (время), S(N) (ускорение) и E(N) (эффективность) при фиксированном значении P.

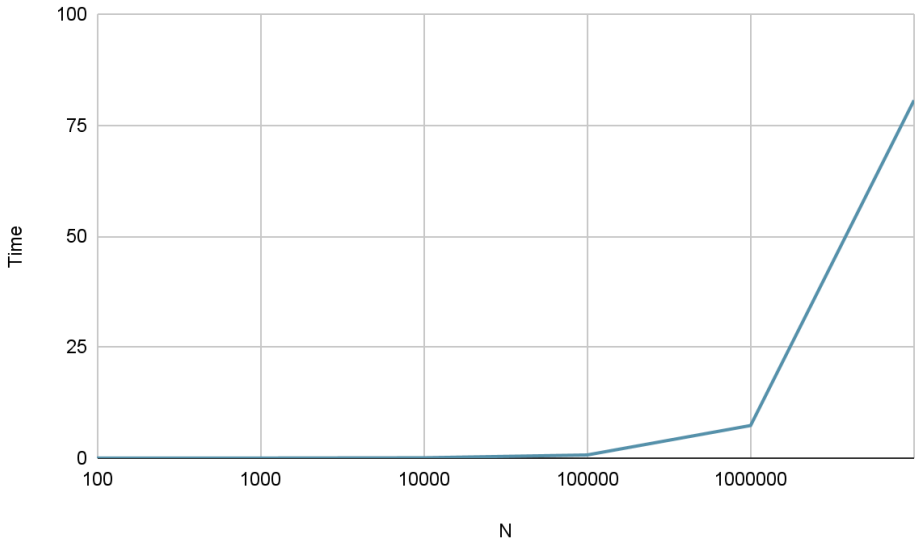

Рис. 1: Зависимость времени работы программы от количества частиц. Используется 8 процессов.

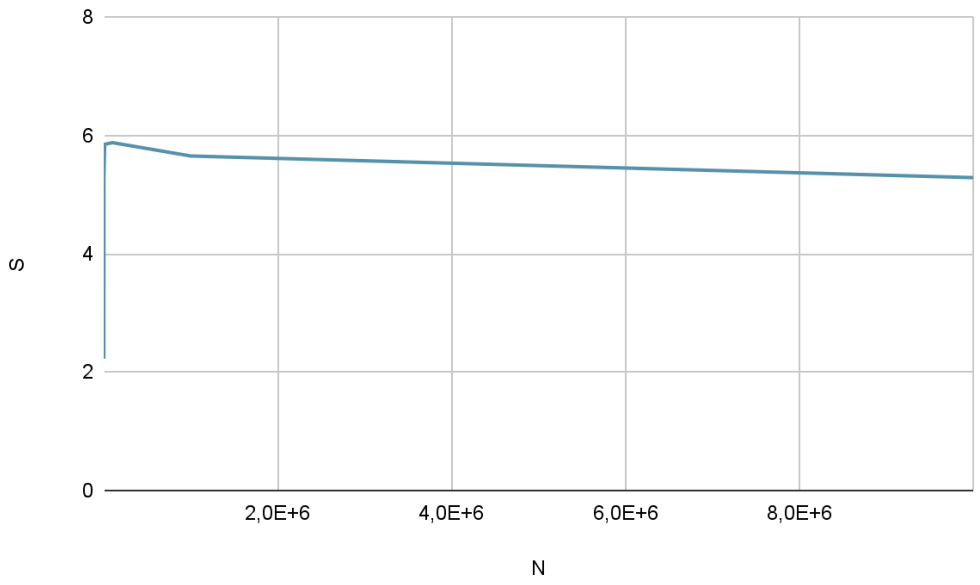

Рис. 2: Зависимость ускорения программы от количества частиц относительно не параллельной версии. Используется 8 процессов.

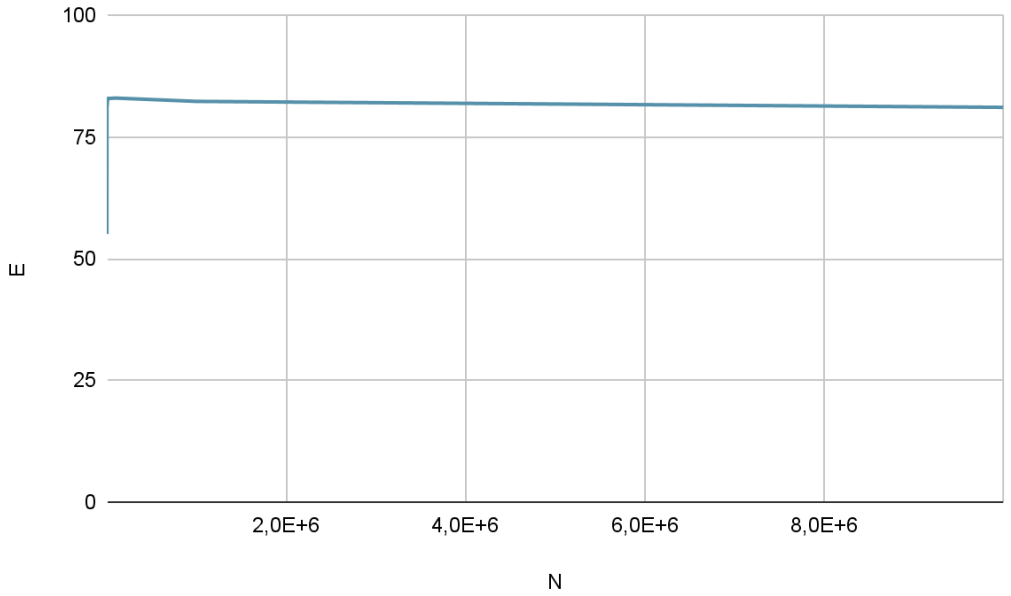

Рис. 3: Зависимость эффективности программы от количества частиц относительно не параллельной версии. Используется 8 процессов.

Нестабильность последних двух графиков в начале экспериментов обусловлена тем, что при небольших значениях N (10 и 100) время, затрачиваемое на выполнение основного цикла программы, становится сопоставимым с накладными расходами на инициализацию библиотеки MPI.

T(P), S(P) и E(P) при фиксированном значении N.

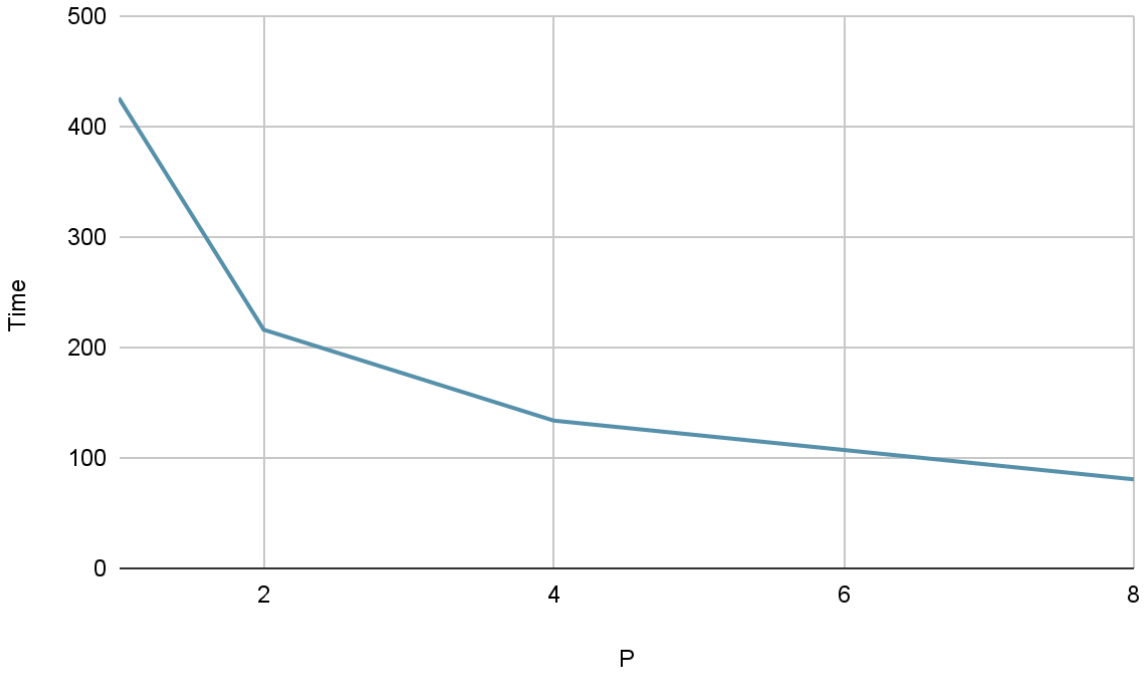

Рис. 4: Зависимость времени работы программы от количества процессов. N = 10^7.

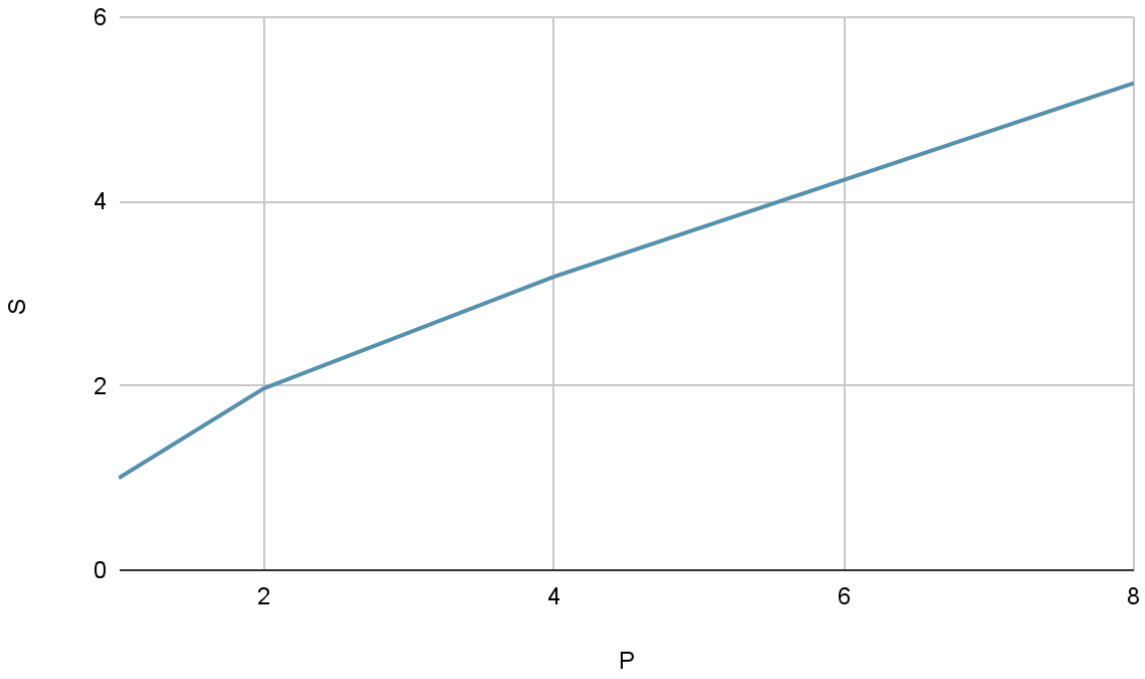

Рис. 5: Зависимость ускорения программы от количества процессов. N = 10^7.

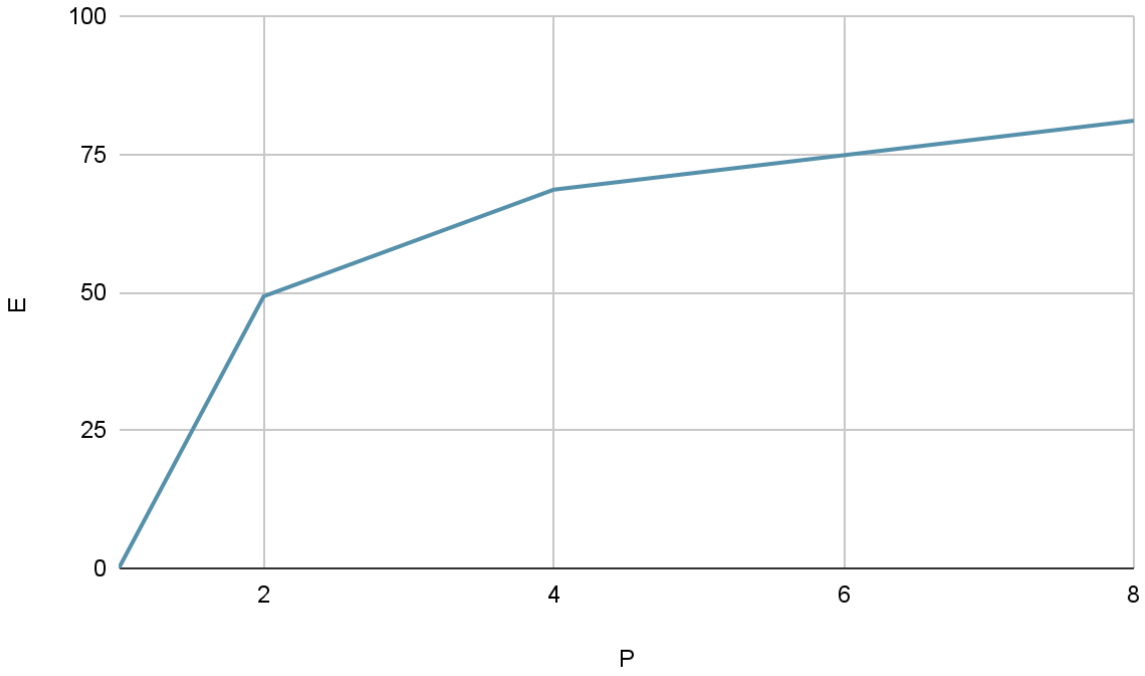

Рис. 6: Зависимость эффективности программы от количества процессов. N = 10^7.

Графики зависимостей T(P), S(P), E(P) от количества используемых процессов P при количестве частиц N = 10^3*P

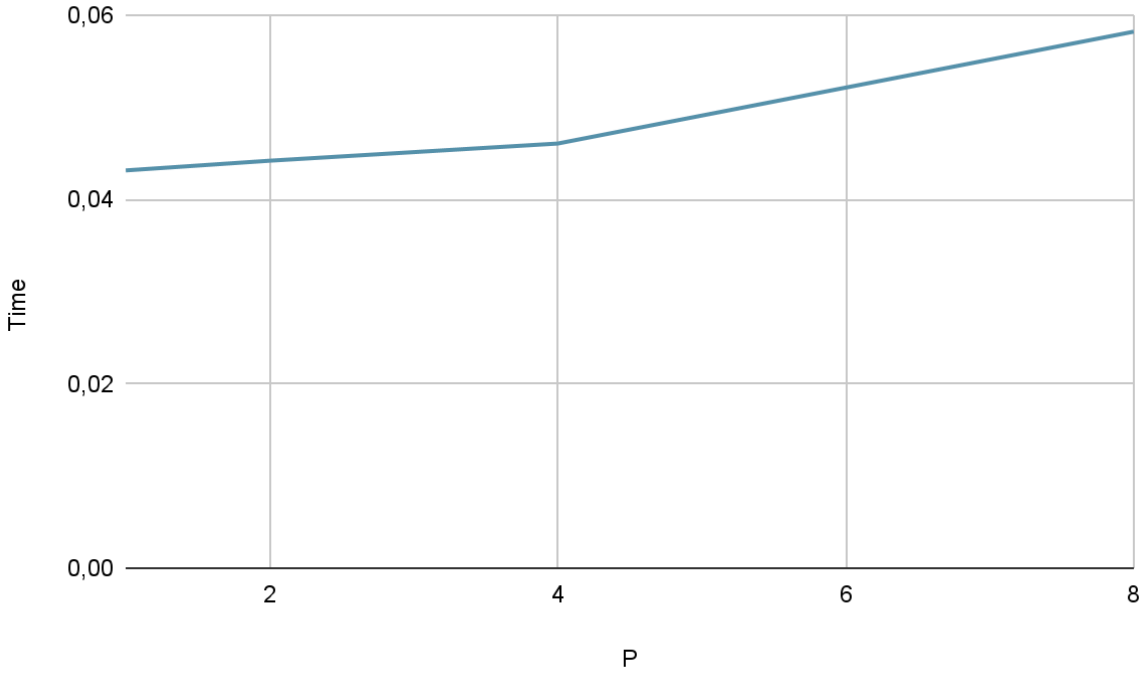

Рис. 7: Зависимость времени работы программы от количества используемых процессов. Количество частиц равно N = 10^3*P.

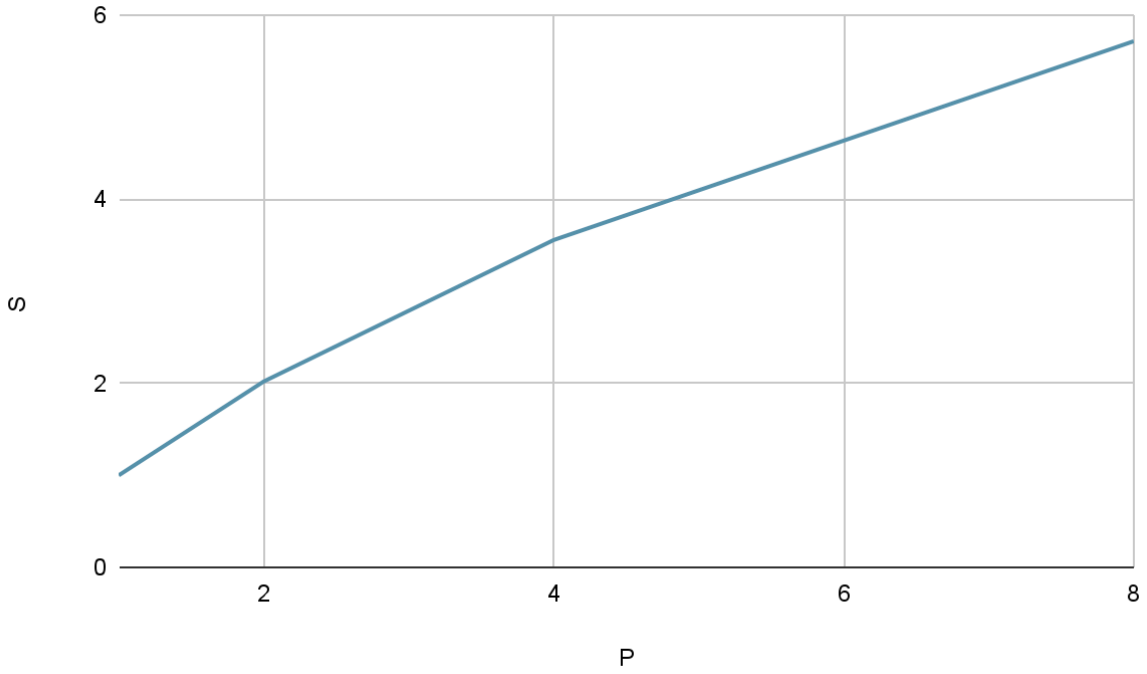

Рис. 8: Зависимость ускорения программы от количества используемых процессов. Количество частиц равно N = 10^3*P.

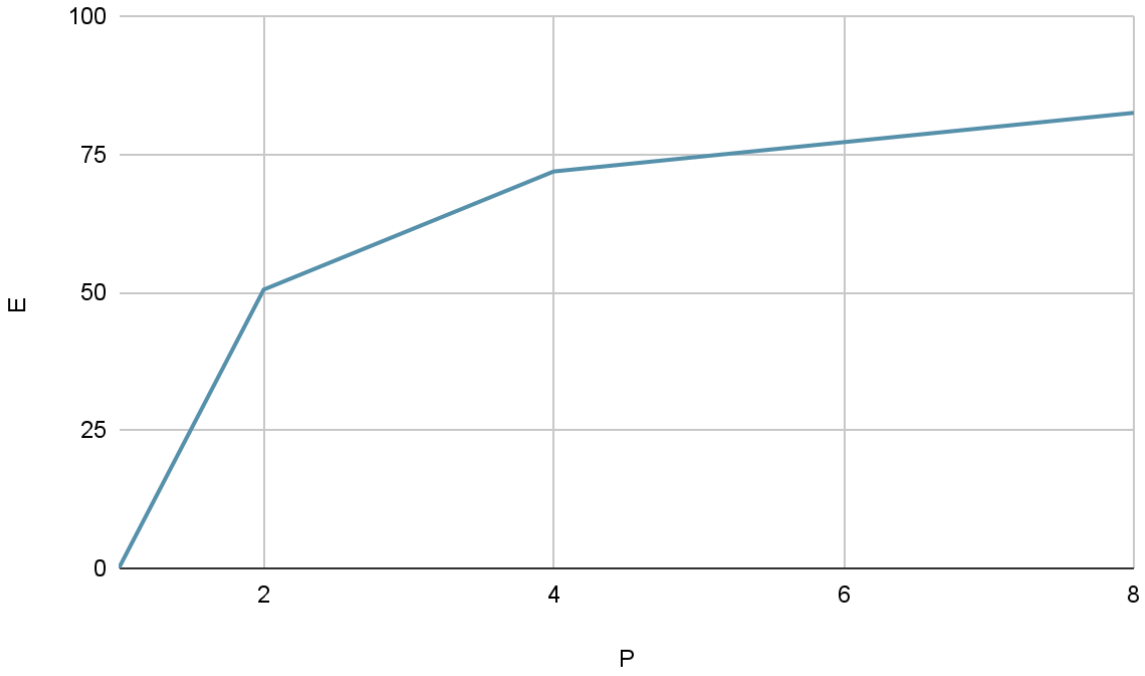

Рис. 9: Зависимость эффективности программы от количества используемых процессов. Количество частиц равно N = 10^3*P.

Код программы

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

using namespace std;

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

pair<int, long long 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 pair<int , long long int>(x, step);
}

void run_mc(int a, int b, int x, double p, int N, int rank, int size)
{
	// srand(time(0));
	int t = 0;
	int w = 0;

	int local_N = N / size;
	int local_count = 0;

	double start_time, end_time;
	MPI_Barrier(MPI_COMM_WORLD);
	start_time = MPI_Wtime();

	for( int i=0; i< local_N ; i++ )
	{
```

```
            pair<int, long long int> temp = do_walk(a, b, x, p, t);
            if(temp.first == b )
                  w += 1;
            t += temp.second;
      }

      MPI_Barrier(MPI_COMM_WORLD);
      end_time = MPI_Wtime();

      double time = end_time - start_time;

      double buf[2];
      double total[2];
      total[0] = 0;
      total[1] = 0;
      buf[0] = ((double) w) / local_N;
      buf[1] = ((double) t) / local_N;
      MPI_Reduce(buf, total, 2, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
      if (rank == 0) {
            ofstream fout("output_1.txt", std::ofstream::out |
std::ofstream::app);
            fout << total[0]/size << ' ' << total[1]/size << endl;
            fout <<
"--------------------------------------------------------------------------------
---------" << endl;
            fout.close();
            fout.open("stat_1.txt", std::ofstream::out | std::ofstream::app);
            fout << "time = " << time << " P = " << size << endl;
            fout << a << ' ' << b << ' ' << x << ' ' << p << ' ' << N <<endl;
            fout <<
"--------------------------------------------------------------------------------
---------" << endl;
            fout.close();
      }
}

int main(int argc, char** argv)
{

      if(argc != 6) {
            cout << "input: a b x p n" << endl;
            return 1;
      }

      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]);
        MPI_Init(&argc, &argv);
        int rank, size;
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);

        srand(time(NULL) + rank);

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

        return 0;
}
```