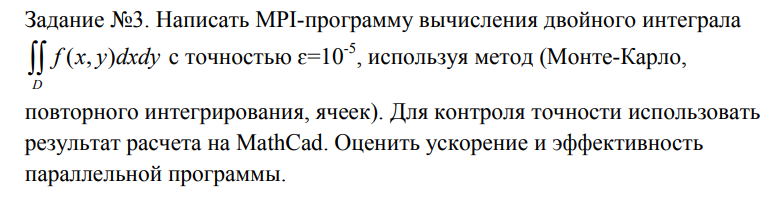
Отчет №5

Гренц Елизавета Алексеевна группа 932220



Описание работы:

Функция **f(x, y)**представляет собой подынтегральную функцию 

**Двойной интеграл-расчет** ( **double\_integral** функция):

Эта функция вычисляет двойной интеграл заданной функции в указанной прямоугольной области. D

Ей ребуется четыре аргумента:

* + - **x\_lower**, **x\_upper**: Нижняя и верхняя границы области интегрирования вдоль оси x.
    - **y\_lower**, **y\_upper**: Нижняя и верхняя границы области интегрирования по оси y.
    - **n**: количество интервалов для каждого измерения (по осям x и y).

Он делит область интеграции *n*×*n* маленькие прямоугольники и аппроксимирует интеграл, используя правило средней точки.

Для каждого прямоугольника он вычисляет координаты средней точки **(x, y)** и оценивает функцию **f(x, y)**.

Важно заметить: Область интегрирования по оси Y поровну разделена между всеми процессами MPI, гарантируя, что каждый процесс вычисляет интеграл по одной и той же части области.

Такой подход гарантирует, что результат интегрирования не зависит от количества процессов, используемых для вычислений.

Функция MPI\_Reduce используется для суммирования локальных интегралов всех процессов и получения окончательного интегрального результата для процесса 0.

Только процесс 0 печатает окончательное интегральное значение и время, затраченное на расчет.

**Код для вставки :**

#include <stdio.h>

#include <mpi.h>

#include <math.h>

double f(double x, double y) {

return (x \* x - y \* y) \* sin(M\_PI \* (x - y) \* (x - y));

}

double double\_integral(double x\_lower, double x\_upper, double y\_lower, double y\_upper, int n, int size) {

int local\_n=n/size;

double hx = (x\_upper - x\_lower) / n;

double hy = (y\_upper - y\_lower) / local\_n;

double sum = 0.0;

for (int i = 0; i < n; ++i) {

double x = x\_lower + (i + 0.5) \* hx; // СЃСЂРµРґРЅСЏСЏ С‚РѕС‡РєР° РґР»СЏ РєР°Р¶РґРѕРіРѕ РёРЅС‚РµРіСЂР°Р»Р° x

for (int j = 0; j < local\_n; ++j) {

double y = y\_lower + (j + 0.5) \* hy; // СЃСЂРµРґРЅСЏСЏ С‚РѕС‡РєР° РґР»СЏ РєР°Р¶РґРѕРіРѕ РёРЅС‚РµРіСЂР°Р»Р° y

if (fabs(y) < x && x < 1 - fabs(y)) { sum += f(x, y); } }

}

return hx \* hy \* sum;

}

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

int rank, size;

double x\_lower = 0.0, x\_upper = 1.0;

double y\_lower = -0.5, y\_upper = 0.5; // РѕР±Р»Р°СЃС‚СЊ y РѕРїСЂРµРґРµР»РµРЅРЅР°СЏ РІ D

int n = 10000;

double integral, st, fin;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

st = MPI\_Wtime();

// СЂР°Р·РґРёР»РµРЅРёРµ y РјРµР¶РґСѓ РїСЂРѕС†РµСЃСЃР°РјРё

double y\_segment = (y\_upper - y\_lower) / size;

double local\_y\_lower = y\_lower + rank \* y\_segment;

double local\_y\_upper = local\_y\_lower + y\_segment;

double local\_integral = double\_integral(x\_lower, x\_upper, local\_y\_lower, local\_y\_upper, n, size);

// СЃСѓРјРјРёСЂСѓРµРј Р»РѕРєР°Р»СЊРЅС‹Рµ РёРЅС‚РµРіСЂР°Р»С‹

MPI\_Reduce(&local\_integral, &integral, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

fin = MPI\_Wtime();

printf("Integral: %.10f\n", integral);

printf("Time: %f\n", fin - st);

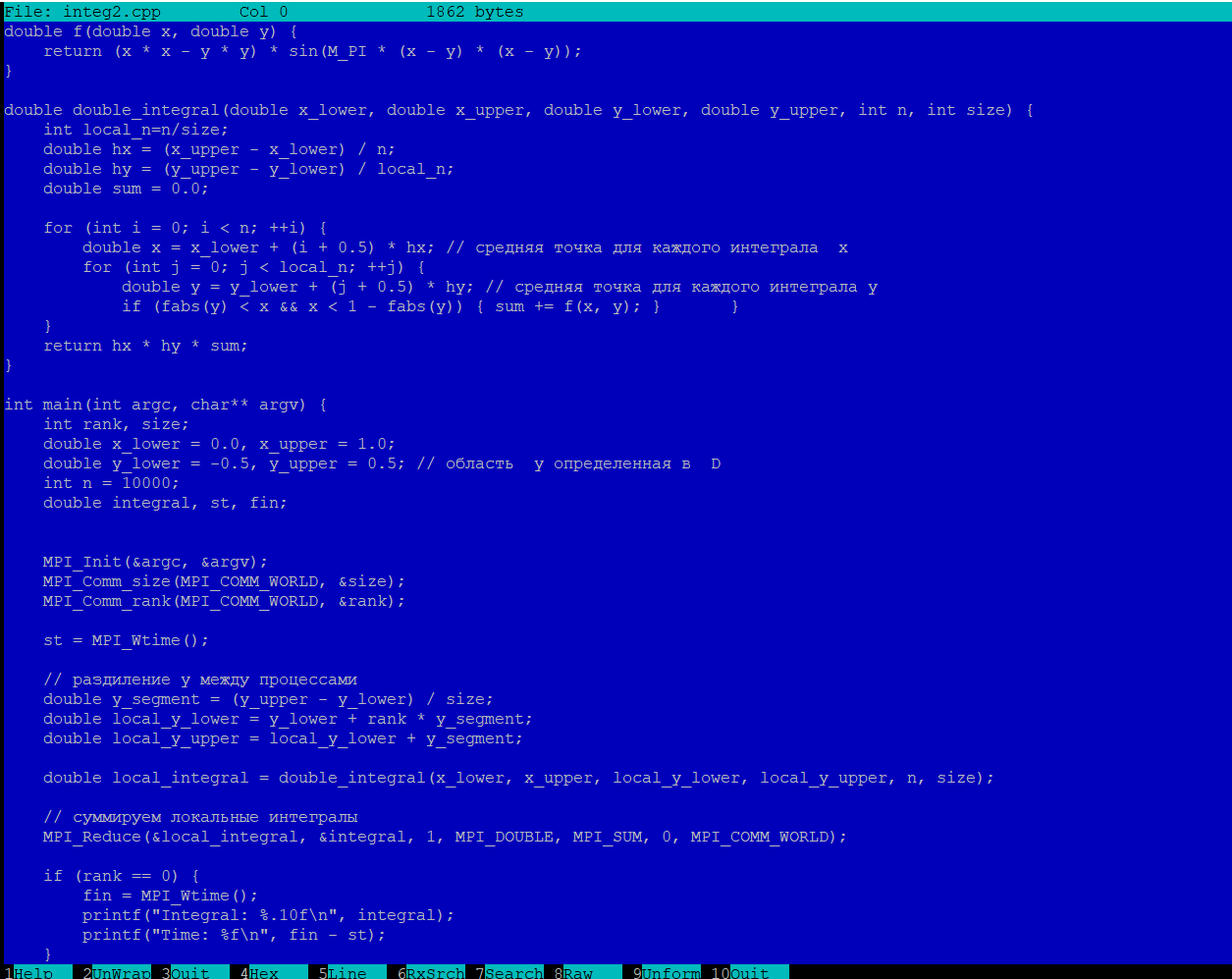
}

MPI\_Finalize();

return 0;

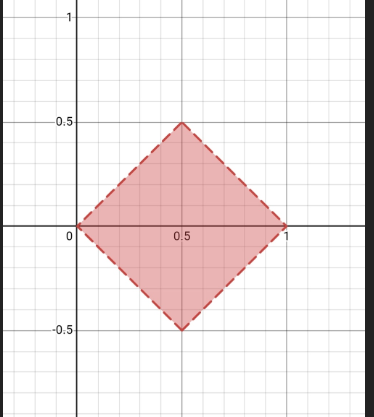
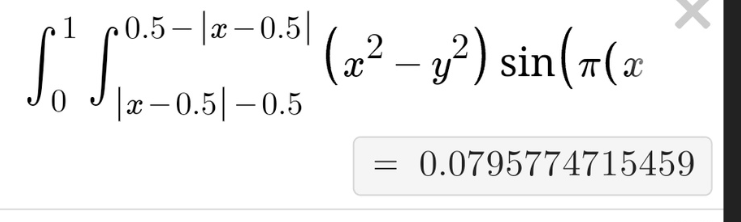
}

Скрин\_для \_просмотра



**Результаты**

Integral: 0.0795695125



**1 процесс**

Time: 1.753538

**2 процесса**

Time: 0.878843

**4 процесса**

Time: 0.625246

**6 процессов**

Time: 0.457939

**8 процессов**

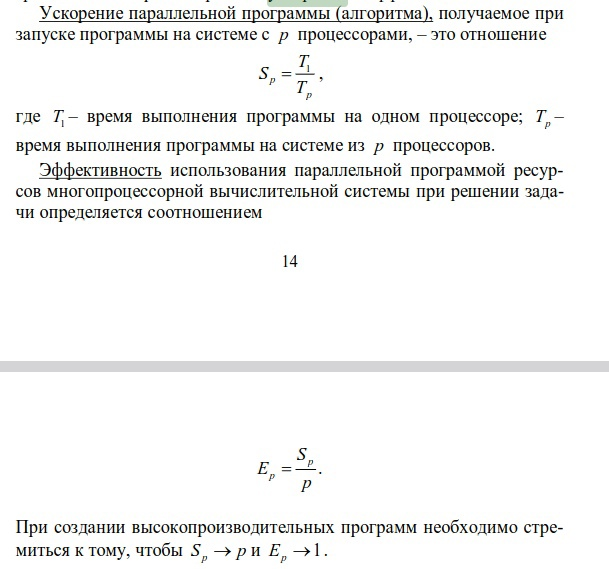
Time: 0.358862

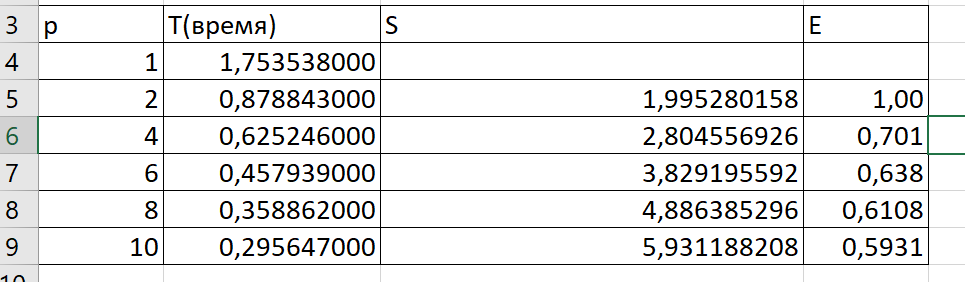
**10 процессов**

Time: 0.295647

**Вывод :**

Оценим Ускорение и Эффективность





С увеличением кол-ва процессов уменьшается время выполнения программы, но самая высокая эффективность достигается не на самом большом числе процессоров, моя программа лучше всего работает **на двух** процессах. Возможно по мере усложнения вычислений результат может измениться. Но это уже вторая программа в которой на 2-х процессах лучший результат.