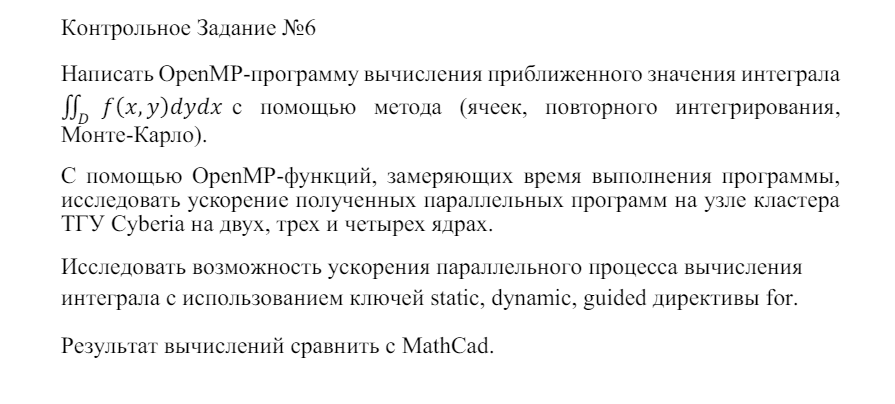
Отчет №6

Гренц Елизавета Алексеевна группа 932220



**Описание работы:**

В данном задании подсчет интеграла происходит так ж как и в предидущем, но уже мы работаем с OpenMP

#pragma omp parallel: эта директива запускает параллельную область, в которой несколько потоков будут параллельно выполнять вложенный код.(нам уже не нужно самостоятельно разделять область между потоками)

default(shared) private(x, y) shared(n, x\_lower, x\_upper, y\_lower, y\_upper, hx, hy): указывает атрибуты совместного использования переменных по умолчанию ( sharedили private). X и y становятся частными для каждого потока, в то время как другие переменные являются общими для всех потоков.

reduction(+:sum): указывает, что переменную sum следует рассматривать как частную переменную для каждого потока

#pragma omp single: гарантирует, что следующий блок кода выполняется в одном потоке

#pragma omp for schedule(static,2): Распределяет итерации внешнего цикла между потоками и размером фрагмента 2.

**Код для вставки :**

#include <stdio.h>

#include <math.h>

#include <omp.h>

double f(double x, double y) {

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

}

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

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

double y\_lower = -0.5, y\_upper = 0.5,y;

int n = 10000,size;

double integral, st, fin;

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

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

double sum = 0.0;

#pragma omp parallel default(shared) private(x, y) shared(n, x\_lower, x\_upper, y\_lower, y\_upper, hx, hy) reduction(+:sum)

{

#pragma omp single

{

st = omp\_get\_wtime();

size = omp\_get\_num\_threads();

}

#pragma omp for schedule(static,2)

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

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

x = x\_lower + (i + 0.5) \* hx;

y = y\_lower + (j + 0.5) \* hy;

if (fabs(y) < x && x < 1 - fabs(y)) {

sum += f(x, y);

}

}

}

}

integral=hx \* hy \* sum;

fin = omp\_get\_wtime();

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

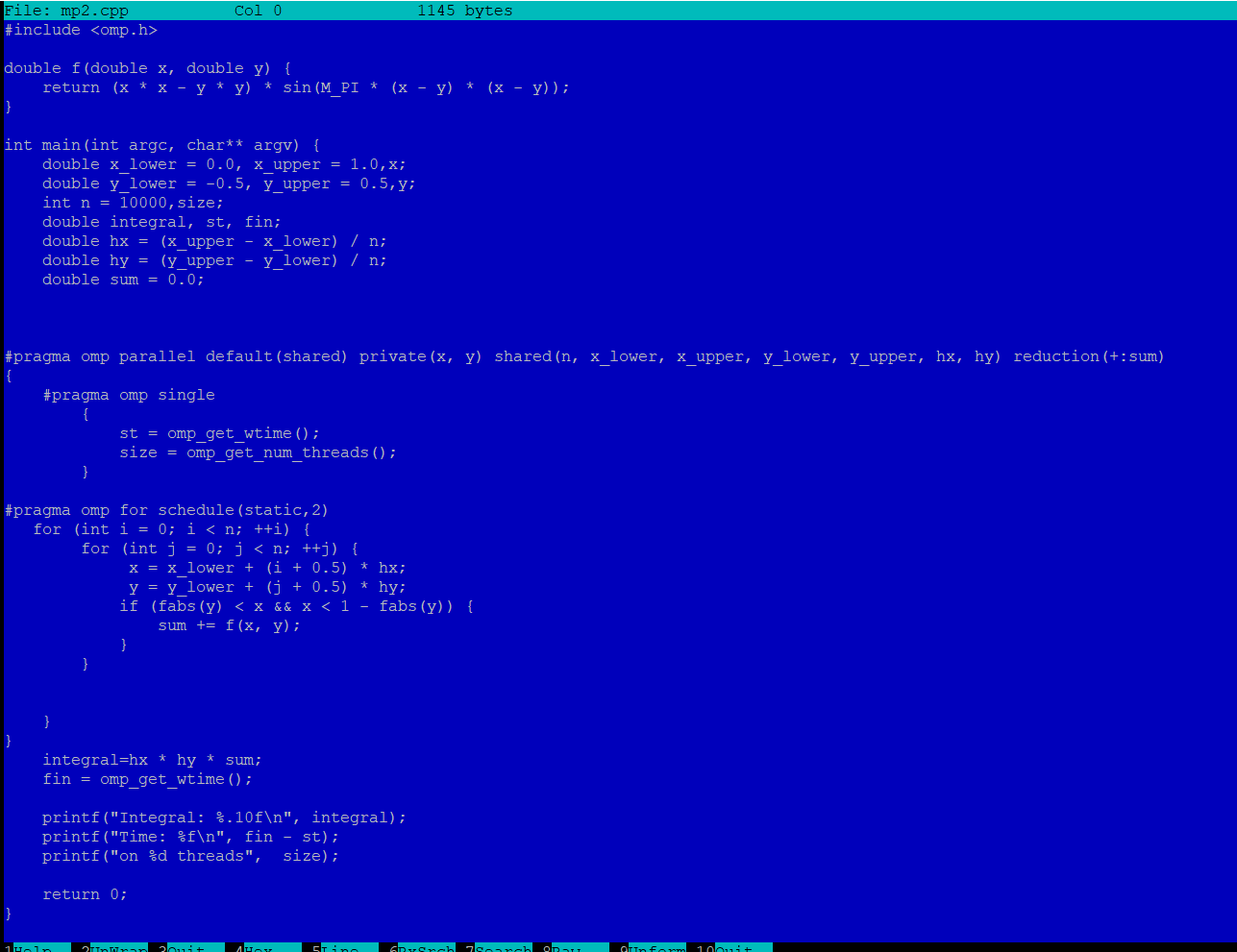
printf("Time: %f\n", fin - st);

printf("on %d threads", size);

return 0;

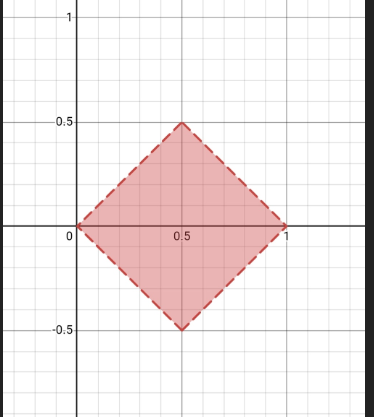
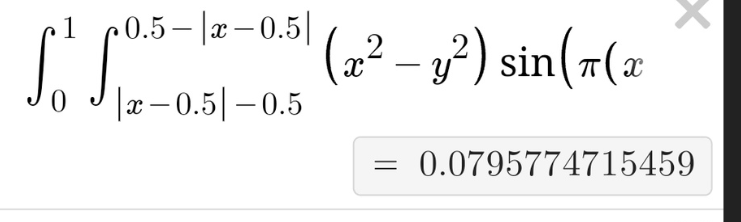
}

Скрин\_для \_просмотра



**Результаты**

Integral: 0.0795695125



Один из выводов

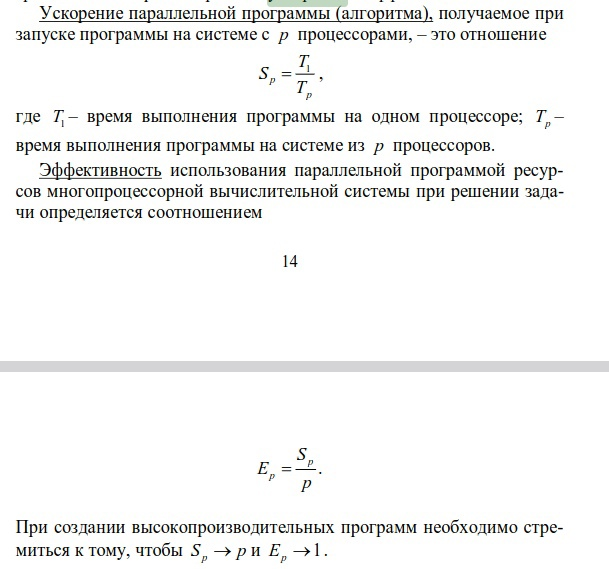
Integral: 0.0795695125

Time: 0.871740

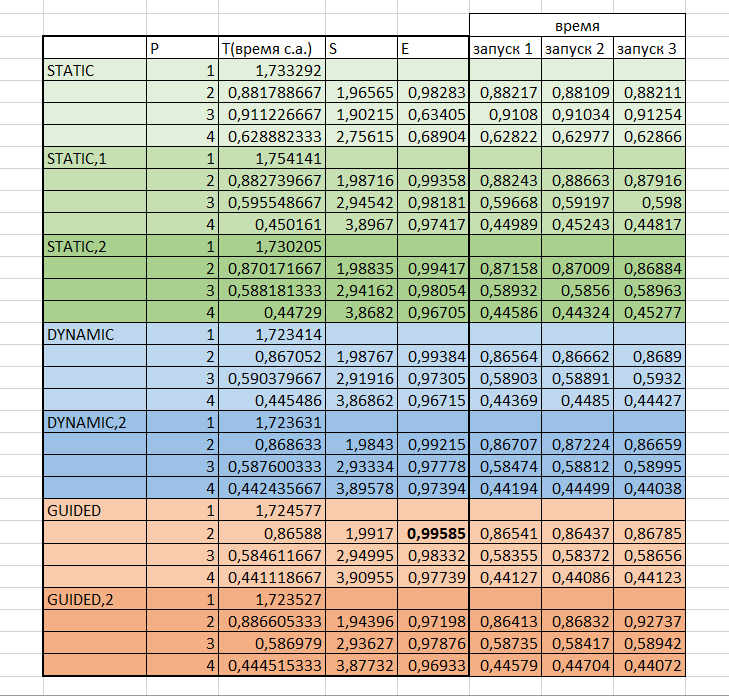
on 2 threads

**Вывод :**

Оценим Ускорение и Эффективность



\*жирным выделена самая большая эффективность



**#pragma omp for schedule(static, 1):**

Эта опция назначает последовательные итерации цикла потокам в циклическом порядке с размером фрагмента, равным 1.

Каждый поток обрабатывает одну итерацию за раз.

**Поскольку мои итерации цикла независимы и, вероятно, имеют одинаковые вычислительные затраты, такое детальное распределение может привести к хорошему балансу нагрузки между потоками.**

Но почему на практике самый лучший результат показало **guided** распределение

Я предполагаю это потому что :

**guided** распределение снижает накладные расходы за счет первоначальной обработки более крупных фрагментов. Тем самым помогает сбалансировать рабочую нагрузку между потоками.

**!НО не будем забывать, что эффективность того или иного распределения зависит от конкретно решаемой задачи.**