Work 10 Кириленко Константин 351



***Подготовка:***

1) Подключим заголовочный файл mpi.h

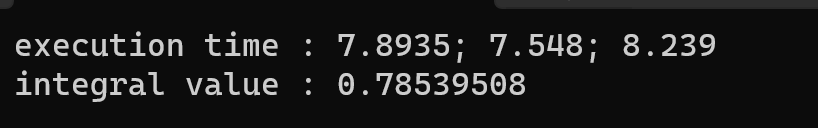
#include "mpi.h"

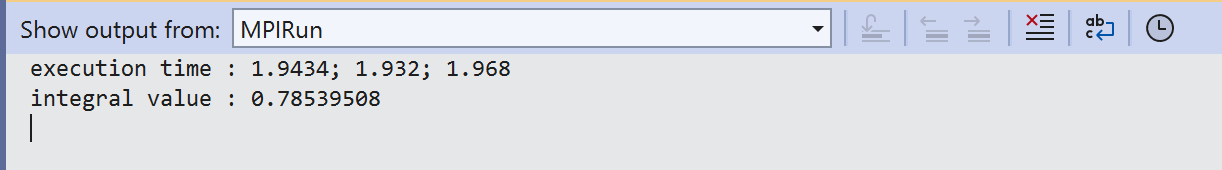
2) Инициализируем MPI-программу

MPI\_Init(NULL, NULL);

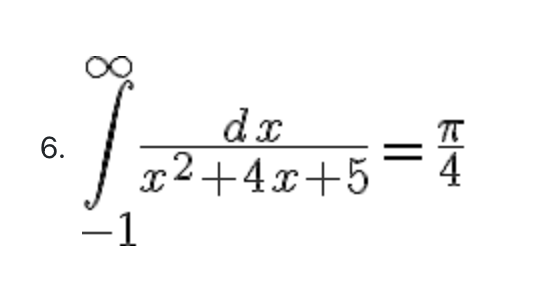
3) Последнем вызовом обязательно является функция

MPI\_Finalize();





Формула Симпсона Вариант 6



//MPI\_Simpliest\_c\_Bind.cpp

#include "mpi.h"

#include <iostream>

using namespace std;

// Число pi

#define PI 3.1415926535897932384626433832795

double f(double x) {

return 1 / (x \* x + 4 \* x + 5);

//return 1 / sqrt(x - x \* x);

}

void simp(const double a, const double b,

double h, double\* res)

{

long long int i, n;

double sum1, sum2, sum11, sum22; // локальная переменная для подсчета интеграла

double x; // координата точки сетки

n = (int)((b - a) / h); // количество точек сетки интегрирования

sum1 = 0.0;

sum2 = 0.0;

h = (b - a) / (2 \* n);

int NProc, ProcId;

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

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

long long n1 = n / NProc;

long long n2 = (ProcId + 1) \* n1;

if (NProc == ProcId + 1) {

n2 = n;

}

long long st = ProcId \* n1;

if (ProcId == 0) st++;

for (i = st; i <= n2; i++)

{

sum1 += f(a + (2 \* i - 1) \* h);

}

MPI\_Reduce(&sum1, &sum11, 1, MPI\_LONG\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

for (i = st; i < n2; i++)

{

sum2 += f(a + 2 \* i \* h);

}

MPI\_Reduce(&sum2, &sum22, 1, MPI\_LONG\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

\*res = h / 3 \* (f(a) + f(b) + 4 \* sum11 + 2 \* sum22);

}

void integral(const double a, const double b,

const double h, double\* res)

{

long long i, n;

double sum; // локальная переменная для подсчета интеграла

double x; // координата точки сетки

n = (int)((b - a) / h); // количество точек сетки интегрирования

int NProc, ProcId;

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

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

long long n1 = n / NProc;

sum = 0.0;

long long n2= (ProcId + 1) \* n1;

if (NProc == ProcId + 1) {

n2 = n;

}

for (i = ProcId \* n1; i < n2 ; i++)

{

x = a + i \* h + h / 2.0;

sum += f(x) \* h;

}

MPI\_Reduce(&sum, res, 1, MPI\_LONG\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

}

double experiment(double\* res)

{

double stime, ftime; // время начала и конца расчета

long double a = -1.0; // левая граница интегрирования

long double b = 1000000.0; // правая граница интегрирования

double h = 0.1; // шаг интегрирования

stime = clock();

simp(a, b, h, res); // вызов функции интегрирования

ftime = clock();

return (ftime - stime) / CLOCKS\_PER\_SEC;

}

int main()

{

MPI\_Init(NULL, NULL);

int i; // переменная цикла

double time; // время проведенного эксперимента

double res; // значение вычисленного интеграла

double min\_time; // минимальное время работы

// реализации алгоритма

double max\_time; // максимальное время работы

// реализации алгоритма

double avg\_time; // среднее время работы

// реализации алгоритма

int numbExp = 10; // количество запусков программы

// первый запуск

min\_time = max\_time = avg\_time = experiment(&res);

// оставшиеся запуски

for (i = 0; i < numbExp - 1; i++)

{

time = experiment(&res);

avg\_time += time;

if (max\_time < time) max\_time = time;

if (min\_time > time) min\_time = time;

}

int ProcId;

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

if (ProcId == 0) {

// вывод результатов эксперимента

cout << "execution time : " << avg\_time / numbExp << "; " <<

min\_time << "; " << max\_time << endl;

cout.precision(8);

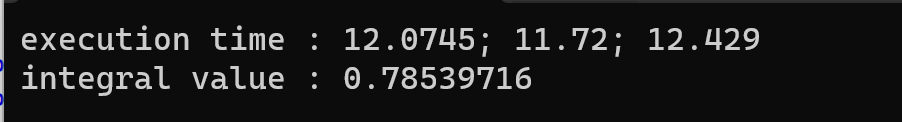
cout << "integral value : " << res << endl;

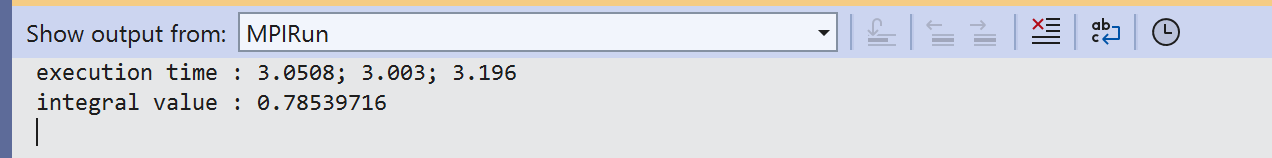
}

MPI\_Finalize();

return 0;

}





Результаты работы программ совпадают, но распараллеленная выигрывает у последовательной.