Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ОСНОВНЫЕ ФУНКЦИИ MPI

Отчет по лабораторной работе №1

По дисциплине

«Параллельное программирование»

Студент гр. 431-3

\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Бекиш

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Руководитель:

Доцент кафедры АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_ С.М. Алфёров

(подпись)

Томск 2024

**Оглавление**

[Введение 4](#_Toc178339856)

[Ход работы 5](#_Toc178339857)

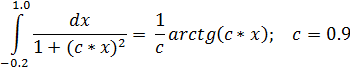
[Заключение 9](#_Toc178339858)

[Приложение А 10](#_Toc178339859)

# Введение

Цель работы: освоить применение основных функций MPI на примере параллельной программы численного интегрирования.

Индивидуальное задание по варианту №6-n:



# Ход работы

1. Создать в домашнем каталоге папку для лабораторной работы №1. Скопировать в него программу integn.c. Проверить работоспособность программы в режиме последовательного и параллельного выполнения на разном числе процессов. Какой метод численного интегрирования используется в программе? Разобраться в MPI функциях, используемых в программе. Определить назначение и типы параметров функций. Какие дополнительные функции, кроме шести основных используются в программе?

Посредством подключения через Visual Code Remote было произведено удалённое подключение к кластеру cluster.asu.tusur.ru. Далее файл integn.cpp был скопирован в каталог лабораторной работы и скомпилирован вызовом в терминале команды: *mpicxx -o lab1 integn.cpp*. Программа вычисляет интеграл от функции cos(x) в интервале от -0.5 до 0.8 методом численного интегрирования, а именно методом прямоугольника. Последовательный вызов программы, выполняемый командой *mpirun -np <n> lab1* на разном числе процессов (1, 2, 4, 8) и количестве интервалов (10000, 100000, 1000000, 10000000) привел к результату, который приведён в таблице 1.1.

Таблица 1.1 — Результаты работы программы integn.cpp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество процессов | Количество интервалов|Time estimating | | | |
|  | 1.00E+04 | 1.00E+05 | 1.00E+06 | 1.00E+07 |
| Последовательный | 6.050E-04 | 5.389E-03 | 4.787E-02 | 4.598E-01 |
| 2 процесса | 3.530E-04 | 2.298E-03 | 2.270E-02 | 2.267E-01 |
| 4 процесса | 3.300E-04 | 1.205E-03 | 1.143E-02 | 1.155E-01 |
| 8 процессов | 1.738E-03 | 1.414E-02 | 1.885E-02 | 5.815E-02 |
| AVERAGE INTEGRAL | 1.197E+00 | 1.197E+00 | 1.197E+00 | 1.197E+00 |
| AVERAGE ERROR | 8.427E-10 | 8.423E-12 | 7.928E-14 | 6.850E-15 |

Как видно из таблицы, с ростом числа разбиений увеличивается время вычисления и уменьшается погрешность. Однако стоит заметить, что при количестве разбиений равное 1000000, увеличение количества потоков уменьшает общее время на вычисление интегральной суммы.

В программе используется несколько основных функций. Помимо них, присутствует вызов следующих функций:

* MPI\_Send(void \*buf, int count, MPI\_Datatype type, int dest, int tag, MPI\_Comm comm) — функция передачи сообщения от процесса отправителя, где buf — адрес буфера памяти отправляемого сообщения, count — количество элементов данных в сообщении, type — тип элементов данных сообщения, dest — ранг процесса-получателя, tag — значение-тег для идентификации сообщения, comm — коммуникатор, в рамках которого выполняется передача данных.
* MPI\_Recv(void \*buf, int count, MPI\_Datatype type, int source, int tag, MPI\_Comm comm, MPI\_Status \*status) — buf, count, type, comm — аналогично MPI\_Send, только для приёма сообщения, source — ранг процесса-получателя, tag — тег сообщения, которое должно быть принято для процесса, status — указатель на структуру данных с информацией о результате выполнения операции приёма данных.

1. Скопировать программу в новый файл. Произвести замену подынтегральной и первообразной функции в соответствие с вариантом. В программе задать значения переменных для пределов интегрирования и параметра математической функции. Выполнить программу на различном числе процессов. Результаты программы направлять переназначением стандартного вывода в файлы результатов.

Для реализации данного пункта задачи постребовалось заменить уже существующие функции f() и fi() на соответствующие варианту подынтегральную и первообразную функции.

Таким образом подынтегральная функция: f(a) = 1/(1 + pow(0.9 \* a, 2)), а соответствующая ей первообразная: fi(a) = (10/(double)9)\*atan(0.9 \* a).

Результаты выполнения программы на различном числе процессов представлены в таблице 1.2.

Таблица 1.2 — Результаты работы программы в соответствие с вариантом

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество процессов | Количество интервалов|Time estimating | | | |
|  | 1.00E+04 | 1.00E+05 | 1.00E+06 | 1.00E+07 |
| Последовательный | 1.125E-03 | 1.071E-02 | 9.425E-02 | 9.164E-01 |
| 2 процесса | 5.700E-04 | 4.626E-03 | 4.580E-02 | 4.575E-01 |
| 4 процесса | 4.960E-04 | 2.755E-03 | 2.318E-02 | 2.314E-01 |
| 8 процессов | 5.800E-04 | 1.232E-03 | 1.242E-02 | 1.189E-01 |
| AVERAGE INTEGRAL | 1.012E+00 | 1.012E+00 | 1.012E+00 | 1.012E+00 |
| AVERAGE ERROR | 4.791E-10 | 1.215E-05 | 1.215E-06 | 1.215E-07 |

В данном случае явно прослеживается польза параллельного выполнения программы, т. к. вне зависимости от используемого количества разбиений при увеличении количества процессов уменьшается общее время вычисления интеграла.

1. Скопировать программу с индивидуальной функцией в новый файл. Заменить индивидуальные функции передачи и приема на коллективные функции MPI\_Bcast и MPI\_Reduce. Заменить в программе назначения пределов интегрирования и параметра функции на ввод их с терминала. Ввод провести в нулевом процессе. Упаковать введенные с терминала пределы интегрирования и параметр функции в буфер, разослать буфер всем процессам и распаковать (функции MPI\_Pack и MPI\_Unpack).

В нулевом процессе осуществляется ввод данных с консоли в программу. Далее с помощью метода multiple\_pack данные запаковываются в буфер и рассылаются посредством метода broadcast на все процессы. Получив буфер, процессы распаковывают в своих экземплярах класса необходимые члены класса. Наконец, после вычисления, все процессы посредством метода reduce посылают root процессу свой результат, в свою очередь root процесс принимает данные, применяет оператор MPI\_SUM и выводит результат вычислений. Результаты выполнения параллельной программы на различном числе процессов представлены в таблице 1.3.

Таблица 1.3 — Результаты работы программы с использованием запаковки данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество процессов | Количество интервалов|Time estimating | | | |
|  | 1.00E+04 | 1.00E+05 | 1.00E+06 | 1.00E+07 |
| Последовательный | 1.208E-03 | 1.068E-02 | 9.358E-02 | 9.184E-01 |
| 2 процесса | 7.153E-04 | 4.616E-03 | 4.596E-02 | 4.586E-01 |
| 4 процесса | 7.148E-04 | 2.785E-03 | 2.346E-02 | 2.315E-01 |
| 8 процессов | 6.501E-04 | 1.255E-03 | 1.190E-02 | 1.174E-01 |
| AVERAGE INTEGRAL | 1.012E+00 | 1.012E+00 | 1.012E+00 | 1.012E+00 |
| AVERAGE ERROR | 4.791E-10 | 1.215E-05 | 1.215E-06 | 1.215E-07 |

Как видно из таблицы, с ростом количества разбиений программа работает медленнее, однако, в перспективе конкретного количества разбиений, увеличение количества процессов в результате приводит к меньшим потерям по времени. Листинг программы представлен приложении А.1.

1. Провести анализ времен выполнения всех трех программ. Какие выводы и рекомендации можно сделать из этого анализа?

В результате анализа всех трёх программ можно сделать вывод, что с увеличением количества процессов повышается эффективность выполнения вычисления, что заметнее всего на большем количестве разбиений интервала определённого интеграла. Также стоит заметить, что при использовании запаковки и распаковки программа в некоторых случаях работает медленнее, так как для данных операций нужны временные ресурсы.

# Заключение

В результате выполнения лабораторной работы я освоил применение основных функций MPI на примере параллельной программы численного интегрирования.

# Приложение А

(обязательное)

**Листинг программы**

Листинг А.1

#pragma once

#include "mpi.h"

class Init\_Process {

protected:

MPI\_Comm \_comm;

int numprocs;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int namelen;

protected:

Init\_Process(int argc, char\* argv[], MPI\_Comm comm = MPI\_COMM\_WORLD)

{

MPI\_Init(&argc, &argv);

\_comm = comm;

MPI\_Comm\_size(\_comm, &numprocs);

MPI\_Get\_processor\_name(processor\_name, &namelen);

}

~Init\_Process() {

MPI\_Finalize();

}

public:

virtual void printInfo(std::string accompanying\_message = "", std::ostream& out = std::cout)

{

out << "\nИмя процесса: " << processor\_name

<< "\nНомер процесса: " << numprocs

<< "\n" << accompanying\_message;

fflush(NULL);

}

};

#pragma once

#include "mpi.h"

#include "init\_process.h"

class Process : public Init\_Process

{

protected:

int process\_ID;

enum PIDs

{

INIT

};

public:

Process(int argc, char\* argv[], MPI\_Comm comm = MPI\_COMM\_WORLD) : Init\_Process(argc, argv, comm) {

MPI\_Comm\_rank(comm, &process\_ID);

}

int get\_process\_ID() const {

return process\_ID;

}

void printInfo(std::string accompanying\_message = "", std::ostream& out = std::cout) {

out << "\n"

<< process\_ID << ": " << accompanying\_message;

fflush(NULL);

}

void send(void\* buffer, int to\_process\_ID, MPI\_Datatype type = MPI\_INT, int count = 1, int tag = 1) {

MPI\_Send(buffer, count, type, to\_process\_ID, tag, \_comm);

}

MPI\_Status recv(void\* buffer, int from\_process\_ID, MPI\_Datatype type = MPI\_INT, int count = 1, int tag = 1) {

MPI\_Status status;

MPI\_Recv(buffer, count, type, from\_process\_ID, tag, \_comm, &status);

return status;

}

void bcast(void\* buffer, MPI\_Datatype type = MPI\_INT, int count = 1, int root = Process::INIT) {

MPI\_Bcast(buffer, count, type, root, \_comm);

}

void reduce(void\* send\_buffer, void\* recv\_buffer, MPI\_Datatype type = MPI\_INT, int root = Process::INIT, MPI\_Op op = MPI\_SUM, int count = 1) {

MPI\_Reduce(send\_buffer, recv\_buffer, count, type, op, root, \_comm);

}

void pack(void\* in\_buffer, int count, void\* out\_buffer, int count\_buffer, MPI\_Datatype type = MPI\_INT, int\* position = NULL) {

MPI\_Pack(in\_buffer, count, type, out\_buffer, count\_buffer, position, \_comm);

}

void unpack(void\* in\_buffer, int insize, void\* out\_buffer, int\* position, MPI\_Datatype type = MPI\_INT, int count = 1) {

MPI\_Unpack(in\_buffer, insize, position, out\_buffer, count, type, \_comm);

}

};

#pragma once

#include <math.h>

#include <fstream>

#include <sstream>

#include "process.h"

class Integral : public Process

{

private:

int intervals = 10;

double xl = -0.2, xh = 1.0, c = 0.9,

h, sum = 0.0, startwtime, endwtime;

std::ofstream fout;

public:

Integral(int argc, char\* argv[], MPI\_Comm comm = MPI\_COMM\_WORLD, std::string filename = "file.txt") : Process(argc, argv, comm)

{

fflush(NULL);

fout.open(filename, std::ios::out);

fflush(NULL);

}

~Integral() {

fout.close();

}

int get\_intervals() {

if (process\_ID == Process::INIT) {

std::cout << "Введите кол-во интервалов: ";

std::cin >> intervals;

}

return intervals;

}

double calculate\_integral() {

h = (xh - xl) / static\_cast<double>(intervals);

std::cout << h;

for (int i = process\_ID + 1; i <= intervals; i += numprocs) {

double x = xl + h \* ((double)i - 0.5);

sum += f(x, c);

}

sum \*= h;

std::stringstream str;

str << std::scientific << "Сумма: " << sum;

Process::printInfo(str.str());

fflush(NULL);

str.clear();

str.str("");

return sum;

}

double sum\_results\_process() {

double integral = 0;

reduce(&sum, &integral, MPI\_DOUBLE);

endwtime = MPI\_Wtime();

if (process\_ID == Process::INIT) {

Init\_Process::printInfo("", fout);

printInfo("", fout);

std::stringstream str;

str << std::scientific << "Апроксимация интегралла: " << integral;

Process::printInfo(str.str(), fout);

fflush(NULL);

str.clear();

str.str("");

str << std::scientific << "Ошибка: " << integral - fi(xh, c) + fi(xl, c);

Process::printInfo(str.str(), fout);

fflush(NULL);

str.clear();

str.str("");

str << std::scientific << "Время подсчета: " << endwtime - startwtime;

Process::printInfo(str.str(), fout);

fflush(NULL);

str.clear();

str.str("");

return integral;

}

return sum;

}

static double f(double a, double c) {

return 1 / (1 + pow(c \* a, 2));

}

static double fi(double a, double c) {

return (1 / c) \* atan(c \* a);

}

friend std::ostream& operator<<(std::ostream& out, Integral& integral) {

return out << "\n\nКол-во интервалов: " << integral.intervals <<

"\nНижняя граница: " << integral.xl << "\nВерхняя граница: " << integral.xh <<

"\nПараметр с: " << integral.c << "\n\nСумма интеграла: " << integral.sum << std::endl;

}

};

#include <iostream>

#include "integral.h"

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

/\*int r, n;

MPI\_Init(&argc, &argv);

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

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

printf("Rank: %i\tProsses: %i\n", r, n);

MPI\_Finalize();\*/

Integral a(argc, argv);

a.calculate\_integral();

a.sum\_results\_process();

std::cout << a << std::endl;

return 0;

}

// mpiexex -n ... file