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)

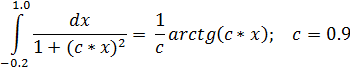
[Заключение 9](#_Toc178339858)

[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

#include <iostream>

#include <limits>

#include <iomanip>

#include <math.h>

#include <fstream>

#include <sstream>

#include "mpi.h"

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);

}

// запись полученной информации в файл

static void write\_file(std::string text, std::ostream& file) {

file << text;

}

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

int choose = 1;

int count\_process, // кол-во процессов

process\_ID, // номер процесса

name\_length; // длина имени процесса

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME]; // имя процесса

int intervals = 1000000; // изначальное кол-во интервалов

double xl = -0.2, // нижняя граница

xh = 1.0, // верхняя граница

c = 0.9, // параметр С

sum = 0.0, // сумма для подсчета интеграла

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

double integral, starttime, endtime;

// открытие файла для записи

std::ofstream file;

file.open("tmp.txt", std::ios\_base::out);

// форматирование строки

std::stringstream text;

MPI\_Init(&argc, &argv); // инициализация

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &process\_ID); // получение процесса

MPI\_Comm\_size(MPI\_COMM\_WORLD, &count\_process); // получение кол-во процессов

MPI\_Get\_processor\_name(processor\_name, &name\_length); // получение имени процесса

MPI\_Status status;

// подготовка информации для записи ее в файл

text << "Name process: " << processor\_name <<

"\n\nProcess\_ID: " << process\_ID <<

"\tCount process: " << count\_process <<

"\n\nКол-во интервалов: " << intervals <<

"\nНижняя граница: " << xl <<

"\nВерхняя граница: " << xh <<

"\nПараметр с: " << c <<

std::endl;

starttime = MPI\_Wtime();

if (choose == 1) {

if (process\_ID == 0) {

for (int \_process\_ID = 1; \_process\_ID < count\_process; \_process\_ID++) {

MPI\_Send(&intervals, 1, MPI\_INT, \_process\_ID, 1, MPI\_COMM\_WORLD);

}

}

else {

MPI\_Recv(&intervals, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &status);

}

text << "\n\nРаспараллеливание выполнено с помощью Send & Recv" << std::endl;

}

else if (choose == 2) {

if (process\_ID == 0) {

/\*\*

\* @brief Широковещательная рассылка

\* - Коллективная операция

\* - передача данных от одного процесса всем процессам программы

\* - должен быть осуществлен всеми процессами указываемого коммуникатора

\*

\* Метод осуществляет рассылку данных из буфера buffer,

\* содержащего count\_data элементов типа type с процесса,

\* имеющего номер root, всем процессам, входящим в коммуникатор comm

\*

\* Указываемый буфер памяти имеет различное назначение в разных процессах:

\* - Для процесса с рангом root,

\* с которого осуществляется рассылка данных,

\* в этом буфере должно находиться рассылаемое сообщение

\* - Для всех остальных процессов указываемый буфер предназначен

\* для приема передаваемых данных.

\*

\* @param buffer буфер памяти с отправляемым

сообщением (для процесса с рангом 0), и для

приема сообщений для всех остальных процессов

\* @param type тип данных пересылаемого/принимаемого сообщения

\* @param count\_data количество элементов памяти типа type

\* @param root ранг процесса, выполняющего рассылку данных

\*/

MPI\_Bcast(&xl, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&xh, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&intervals, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

}

text << "\n\nРаспараллеливание выполнено с помощью Bcast" << std::endl;

}

else if (choose == 3) {

char pack\_buf[100];

int position = 0;

if (process\_ID == 0) {

/\*\*

\* @brief Запаковка сообщения из буфера data, длиной count\_data типа данных type

в буферное пространство, описанное аргументами buf и count\_buf (в байтах).

Параметр position определяет номер ячейки в выходном буфере, с которого будет заполняться buf.

После заполнения значение position инкрементируется в количестве заполненных байтов.

\*

\* @param data буфер памяти с сообщением для запаковки

\* @param count\_data количество значений этого буфера типа type

\* @param buf буфер памяти для запакованных значений

\* @param count\_buf общее количество байтов этого буфера

\* @param type MPI\_Datatype (тип) данных пакуемого сообщения

\* @param bufpos позиция в выходном буфере, с которого необходимо начать заполнение

\*/

MPI\_Pack(&xl, 1, MPI\_DOUBLE, &pack\_buf, 100, &position, MPI\_COMM\_WORLD);

MPI\_Pack(&xh, 1, MPI\_DOUBLE, &pack\_buf, 100, &position, MPI\_COMM\_WORLD);

MPI\_Pack(&intervals, 1, MPI\_INT, &pack\_buf, 100, &position, MPI\_COMM\_WORLD);

}

MPI\_Bcast(&pack\_buf, 100, MPI\_PACKED, 0, MPI\_COMM\_WORLD);

if (process\_ID != 0) {

/\*\*

\* @brief Распаковка сообщений

\* Метод распаковывает сообщение в приемный буфер, описанный аргументами outbuf, outcount, type

\* из буферного пространства, описанного аргументами inbuf и insize.

\* Выходным буфером может быть любой коммуникационный буфер, разрешенный в MPI\_RECV.

\* Входной буфер есть смежная область памяти, содержащая insize байтов, начиная с адреса inbuf.

\* Входное значение position есть первая ячейка во входном буфере, занятом упакованным сообщением.

\* рosition инкрементируется размером упакованного сообщения,

\* так что выходное значение рosition есть первая ячейка во входном буфере после ячеек,

\* занятых сообщением, которое было упаковано.

\* сomm есть коммуникатор для приема упакованного сообщения.

\*

\* @param inbuf буфер, из которого будет распаковываться сообщение

\* @param insize размер этого буфера в байтах

\* @param outbuf буфер, куда будет распаковываться сообщение

\* @param bufpos позиция во входном буфере, указывающая откуда распаковывать сообщение

\* @param type тип данных распаковываемого сообщения

\* @param outcount число единиц распаковываемого сообщения типа type

\*/

MPI\_Unpack(pack\_buf, 100, &position, &xl, 1, MPI\_DOUBLE, MPI\_COMM\_WORLD);

MPI\_Unpack(pack\_buf, 100, &position, &xh, 1, MPI\_DOUBLE, MPI\_COMM\_WORLD);

MPI\_Unpack(pack\_buf, 100, &position, &intervals, 1, MPI\_INT, MPI\_COMM\_WORLD);

}

text << "\n\nРаспараллеливание выполнено с помощью Pack & Unpack" << std::endl;

}

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

double x = xl + h \* ((double)i - 0.5); // вычисление равномерной сетки

sum += f(x, c);

}

sum \*= h;

if (choose == 1) {

if (process\_ID != 0) {

/\*MPI\_Send(buf, count, datatype, dest, tag, comm)

@brief Метод передачи сообщения: в стандартном режиме (блокирующий)

Метод выполняет посылку count элементов типа type сообщения buffer

с идентификатором tag процессу \_process\_ID в области связи коммуникатора comm.

Переменная buf - это, как правило, массив или скалярная переменная.

В последнем случае значение count = 1.

@param buf начальный адрес буфера посылки сообщения(альтернатива)

@param count число элементов в буфере посылки(неотрицательное целое)

@param datatype тип данных каждого элемента в буфере посылки(дескриптор)

@param dest номер процесса - получателя(целое)

@param tag тэг сообщения(целое)

@param comm коммуникатор(дескриптор)\*/

MPI\_Send(&sum, 1, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD);

}

if (process\_ID == 0) {

integral = sum;

for (int \_process\_ID = 1; \_process\_ID < count\_process; \_process\_ID++) {

/\*

\* @brief Метод приёма сообщения: блокирующий

\*

\* Метод выполняет прием count\_data элементов типа type сообщения buffer

\* с идентификатором tag от процесса from\_PID в области связи коммуникатора comm.

\*

\* @param buffer адрес заполняемого буфера памяти

\* @param \_process\_ID ранг процесса, от которого ожидать сообщение

\* @param type тип данных принимаемого сообщений

\* @param count\_data \_максимальное\_ количество элементов памяти типа type

\* @param tag идентификатор принимаемого сообщения: целое число от 0 до 32767

\* - определяет смысл принятого сообщения.

\* - Сообщения, пришедшие в неизвестном порядке,

\* могут извлекаться из общего входного потока в нужном алгоритму порядке.

\* @return status MPI\_Status получения сообщения

\*/

MPI\_Recv(&sum, 1, MPI\_DOUBLE, \_process\_ID, 1, MPI\_COMM\_WORLD, &status);

integral += sum;

}

endtime = MPI\_Wtime();

text <<

"\nСумма интеграла: " << std::scientific << sum <<

"\nАпроксимация интеграла: " << std::scientific << integral <<

"\nОшибка: " << std::scientific << integral - fi(xh, c) + fi(xl, c) <<

"\nВремя подсчета: " << std::scientific << endtime - starttime <<

std::endl;

}

}

else if (choose == 2) {

integral = sum;

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

endtime = MPI\_Wtime();

text <<

"\nСумма интеграла: " << std::scientific << sum <<

"\nАпроксимация интеграла: " << std::scientific << integral <<

"\nОшибка: " << std::scientific << integral - fi(xh, c) + fi(xl, c) <<

"\nВремя подсчета: " << std::scientific << endtime - starttime <<

std::endl;

}

write\_file(text.str(), file);

MPI\_Finalize();

return 0;

}

//mpiexex -n ... file