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

Отчёт о лабораторной работе № 1 по дисциплине «Параллельное программирование»

Студент гр. 431-3

\_\_\_\_\_\_\_ Д.П. Андреев

«\_\_» \_\_\_\_\_\_\_\_\_ 2024

Проверил

Доцент каф. АСУ, к.т.н

\_\_\_\_\_\_\_ С.М. Алфёров

«\_\_» \_\_\_\_\_\_\_\_\_ 2024

Томск 2024

**1 Цель лабораторной работы**

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

## 2 Задание

Задание на лабораторную работу (вариант 9): разобраться с основами MPI. Вычислить интеграл по формуле на рисунке 2.1.

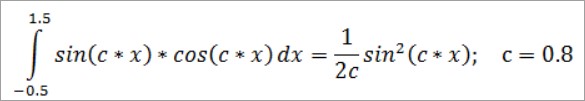


Рисунок 2.1 — формула для вариант 9

Запустить программу на различном количестве процессов, в индивидуальном и коллективном режиме. Сравнить результаты.

### 3 Использованные MPI функции

В этом коде используется несколько функций MPI, которые управляют процессами и обеспечивают распределённое вычисление интеграла. Разберём каждую из них:

MPI\_Init

Описание: Эта функция инициализирует среду MPI и должна быть вызвана в самом начале программы. Она подготавливает библиотеку MPI для работы, разбирает аргументы командной строки (если переданы) и инициализирует необходимые внутренние структуры для взаимодействия между процессами.

Аргументы: argc, argv — стандартные параметры командной строки.

Возвращаемое значение: нет.

Примечание: после вызова этой функции каждый процесс начинает выполнять одну и ту же программу.

MPI\_Comm\_size

Описание: Эта функция возвращает количество процессов, запущенных в коммуникаторе (группе процессов), указанном как первый аргумент (в данном случае, MPI\_COMM\_WORLD). В вашем примере все процессы принадлежат одному глобальному коммуникатору.

Аргументы:

MPI\_COMM\_WORLD — коммуникатор, представляющий всех запущенных процессов.

&numprocs — переменная, в которую записывается количество процессов.

Возвращаемое значение: нет.

Примечание: используется для определения общего числа процессов, участвующих в расчёте.

MPI\_Comm\_rank

Описание: возвращает идентификатор текущего процесса (его «ранг») в указанном коммуникаторе. Это число (индекс) указывает, каким по порядку является данный процесс среди всех запущенных.

Аргументы:

MPI\_COMM\_WORLD — коммуникатор.

&myid — переменная, в которую записывается идентификатор процесса.

Возвращаемое значение: нет.

Примечание: Значение myid будет уникально для каждого процесса и используется для разделения задач между процессами.

MPI\_Get\_processor\_name

Описание: Эта функция возвращает имя процессора, на котором запущен процесс, и длину этого имени.

Аргументы:

processor\_name — массив, в который записывается имя процессора.

&namelen — переменная, в которую записывается длина имени.

Возвращаемое значение: нет.

Примечание: Используется для отладки или вывода информации о распределении процессов по разным машинам в кластере.

MPI\_Send

Описание: Эта функция отправляет сообщение от одного процесса к другому. В данном случае, процесс с рангом myid == 0 отправляет количество интервалов n другим процессам.

Аргументы:

&n — указатель на буфер, содержащий данные для отправки (здесь это переменная n).

1 — количество элементов, которые нужно отправить (здесь один элемент типа int).

MPI\_INT — тип данных (в данном случае целое число int). i — ранг процесса, которому отправляется сообщение.

1 — метка сообщения (используется для различения сообщений одного типа).

MPI\_COMM\_WORLD — коммуникатор.

Возвращаемое значение: нет.

Примечание: Функция блокирующая, то есть выполнение программы приостанавливается до завершения отправки сообщения.

6. MPI\_Recv

Описание: Эта функция принимает сообщение. В коде она используется для получения количества интервалов n от процесса с рангом 0 другими процессами.

Аргументы:

&n — указатель на буфер для принятия данных.

1 — количество принимаемых элементов (один элемент типа int).

MPI\_INT — тип принимаемых данных.

— ранг процесса, от которого ожидается сообщение (здесь от процесса с рангом 0).

— метка сообщения (соответствующая отправке с меткой 1).

MPI\_COMM\_WORLD — коммуникатор.

&stats — структура, содержащая информацию о статусе передачи (например, источник сообщения, его длину и т.д.).

Возвращаемое значение: нет.

Примечание: Функция блокирующая, процесс будет ожидать, пока не получит сообщение.

MPI\_Wtime

Описание: Эта функция возвращает текущее время в виде вещественного числа с плавающей точкой (в секундах). В вашем коде она используется для измерения времени выполнения вычислений.

Аргументы: нет.

Возвращаемое значение: текущее время (точное время в секундах).

Примечание: Используется для замера времени выполнения программы или её частей. Разница между двумя вызовами MPI\_Wtime показывает затраченное время.

MPI\_Finalize

Описание: Эта функция завершает работу MPI и освобождает все ресурсы, связанные с библиотекой MPI. Она должна быть вызвана в конце программы.

Аргументы: нет.

Возвращаемое значение: нет.

Примечание: После вызова этой функции все MPI функции больше не доступны для использования. Программа завершает работу.

**4 Листинг программы**

Main.cpp:

#include "mpi.h"

#include <math.h>

#include <stdio.h>

#include <iostream>

static constexpr double f(double a) {

double c = 0.8;

return sin(c \* a) \* cos(c \* a);

}

static constexpr double fi(double a) { double c = 0.8;

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

}

int main(int argc, char \*argv[]) { int done = 0, n, myid, numprocs, i; double myfunk, funk, h, sum, x; double xl = -0.5, // low border

xh = 0.8; // high border double startwtime, endwtime;

int namelen;

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

MPI\_Status stats;

MPI\_Init(&argc, &argv);

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

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

MPI\_Get\_processor\_name(processor\_name, &namelen); fprintf(stderr, "Process %d on %s\n", myid, processor\_name); fflush(stderr); n = 100000000; while (!done) { if (myid == 0) {

std::cout << "Using interval count: " << n << std::endl; startwtime = MPI\_Wtime();

/\* Sending the number of intervals to other nodes \*/

for (i = 1; i < numprocs; i++) { MPI\_Send(&n, /\* buffer \*/

1, /\* one data \*/ MPI\_INT, /\* type \*/ i, /\* to which node \*/ 1, /\* tag of message \*/

MPI\_COMM\_WORLD); /\* common communicator \*/

}

} else {

MPI\_Recv(&n, /\* buffer \*/

1, /\* one data \*/

MPI\_INT, /\* type \*/

0, /\* from which node \*/

1, /\* tag of message \*/

MPI\_COMM\_WORLD, /\* common communicator \*/

&stats); /\* status of errors \*/

}

if (n == 0) done = 1; else {

/\* Calculate of integral \*/ h = (xh - xl) / (double)n; sum = 0.0;

for (i = myid + 1; i <= n; i += numprocs) { x = xl + h \* ((double)i - 0.5); sum += f(x);

}

myfunk = h \* sum;

printf("Process %d SUMM %.16f\n", myid, myfunk);

/\* Sending the local sum to node 0 \*/ if (myid != 0) {

MPI\_Send(&myfunk, /\* buffer \*/

1, /\* one data \*/

MPI\_DOUBLE, /\* type \*/

0, /\* to which node \*/

1, /\* tag of message \*/

MPI\_COMM\_WORLD); /\* common communicator \*/

}

if (myid == 0) { funk = myfunk;

for (i = 1; i < numprocs; i++) {

MPI\_Recv(&myfunk, 1, MPI\_DOUBLE, i, 1, MPI\_COMM\_WORLD, &stats); funk += myfunk;

}

printf("Integral is approximately %.16f, Error %.16f\n", funk, funk - fi(xh) + fi(xl));

endwtime = MPI\_Wtime();

printf("Time of calculation = %f\n", endwtime - startwtime);

}

}

}

MPI\_Finalize();

}

### 5 Примеры работы программы

Разберём работу программы. Пример работы изображён на рисунке 5.1.

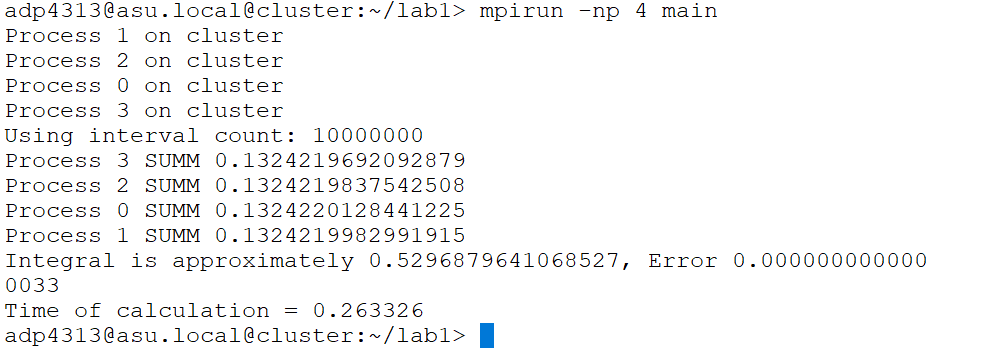


Рисунок 5.1 — Пример работы программы

На рисунке изображён процесс компиляции через MPI, и последующий запуск. На рисунке можно наблюдать значительное ускорение выполнения за счёт парализации работы суммирования.

Теперь заменим индивидуальные функции передачи и приема на коллективные функции MPI\_Bcast и MPI\_Reduce. Пример работы изменёной программы на рисунке 5.2.

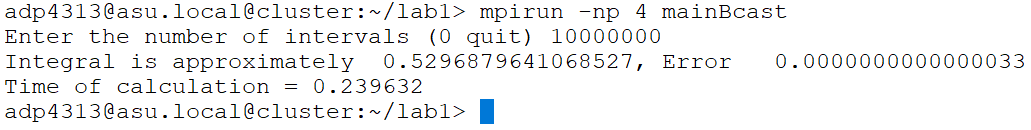


Рисунок 5.2 — Пример работы программы

Далее заменим в программе назначения пределов интегрирования и параметра функции на ввод их с терминала. Упакуем введенные с терминала пределы интегрирования и параметр функции в буфер и разошлём буфер всем процессам и распакуем, используя функции MPI\_Pack и MPI\_Unpack. Пример работы такой программы изображён на рисунке 5.3.

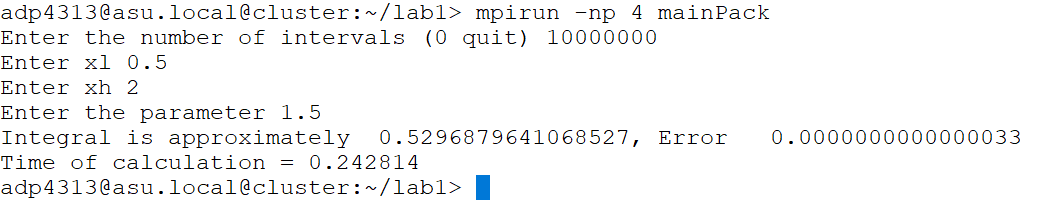


Рисунок 5.3 — Пример работы программы

### 6 Выводы

Таким образом, я изучил основы работы с MPI и узнал практическую пользу от парализации работы программы. Время решения задачи можно таким образом значительно уменьшить.