**Федеральное государственное бюджетное образовательное учреждение высшего образования «Ивановский государственный энергетический университет имени В.И. Ленина»**

**Отчет по курсовой работе**

**по параллельному программированию**

**Выполнил: Тютюкин Е.С.**

**Проверила: Чернышева Л.П.**

**Иваново 2020**

**Задание №1**

**Текст задачи:** выполнить пересылку массива объединений, имеющих поля int, double при помощи производного типа данных.

**Используемые функции:**

**int** MPI\_Send (**void** \*buf, **int** count, MPI\_Datatype type, **int** dest, **int** tag, MPI\_Comm comm)

buf - адрес буфера памяти, в котором располагаются данные отправляемого сообщения,

count - количество элементов данных пересылаемого сообщения,

type - тип элементов данных пересылаемого сообщения,

dest - ранг процесса, которому отправляется сообщение,

tag - значение-тег, используемое для идентификации сообщения,

comm - коммуникатор, в рамках которого выполняется передача данных.

**int** MPI\_Recv (**void** \*buf, **int** count, MPI\_Datatype type, **int** source, **int** tag, MPI\_Comm comm, MPI\_Status \*status)

buf - адрес буфера памяти, в котором располагаются данные принимаемого сообщения,

count - количество элементов данных принимаемого сообщения;

type - тип элементов данных принимаемого сообщения;

source - ранг процесса, от которого должен быть выполнен прием сообщения;

tag - тег сообщения, которое должно быть принято для процесса;

comm - коммуникатор, в рамках которого выполняется передача данных;

status - указатель на структуру данных с информацией о результате выполнения.

**Текст программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include "mpi.h"

//mpiexec -n 2 ConsoleApplication1.exe

#define n 3

#define m 4

#define l 11 // Размер массива

#define k 3 // Количество полей

using namespace std;

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

// Шапка

int size, rank, msgtag = 3;

MPI\_Status status;

if (MPI\_Init(&argc, &argv) != MPI\_SUCCESS)

return 1;

if (MPI\_Comm\_size(MPI\_COMM\_WORLD, &size) != MPI\_SUCCESS) {

MPI\_Finalize();

return 2;

}

if (MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank) != MPI\_SUCCESS) {

MPI\_Finalize();

return 3;

}

union L {

double a;

int b;

} myunion[l], unionType[l];

// Создаем свой тип

MPI\_Datatype mytype;

MPI\_Datatype r[k] = { MPI\_DOUBLE, MPI\_INT }; // Массив с описанием типов полей

int block[k] = { 1, 1 }; // Массив с длинами полей

MPI\_Aint disp[k], tt; // Массив со смещением полей от начала структуры в байтах

MPI\_Get\_address(&myunion[0].a, &disp[0]);

MPI\_Get\_address(&myunion[0].b, &disp[1]);

tt = disp[0];

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

disp[i] -= tt;

}

MPI\_Type\_create\_struct(k, block, disp, r, &mytype);

MPI\_Type\_commit(&mytype);

double tNach = 0.0, tKon = 0.0; // Переменные для время

if (!rank) {

// Заполнение массива

for (int i = 0; i < l; i++)

{

myunion[i].a = i;

myunion[i].b = i;

}

tNach = MPI\_Wtime();

MPI\_Send(&tNach, 1, MPI\_DOUBLE, 1, msgtag, MPI\_COMM\_WORLD);

MPI\_Send(myunion, l, mytype, 1, msgtag, MPI\_COMM\_WORLD);

tNach = MPI\_Wtime();

MPI\_Send(&tNach, 1, MPI\_DOUBLE, 1, msgtag, MPI\_COMM\_WORLD);

}

else {

int pos = 0;

double tProm;

tKon = MPI\_Wtime();

MPI\_Recv(&tProm, 1, MPI\_DOUBLE, 0, msgtag, MPI\_COMM\_WORLD, &status);

MPI\_Recv(unionType, l, mytype, 0, msgtag, MPI\_COMM\_WORLD, &status);

tKon = MPI\_Wtime();

FILE\* z;

z = fopen("results1.dat ", "w");

fprintf(z, "TIME %f\n", tKon - tProm);

fprintf(z, "\n");

for (int i = 0; i < l; i++)

{

fprintf(z, "i=%d\n", i);

fprintf(z, "a=%f\n", unionType[i].a);

fprintf(z, "\n");

fprintf(z, "b=%d\n", unionType[i].b);

fprintf(z, "==============================\n");

}

fclose(z);

}

MPI\_Finalize();

return 0;

}

**Результаты:**

**results1.dat**

TIME 0.000469

i=0

a=0.000000

b=0

==============================

i=1

a=1.000000

b=1

==============================

i=2

a=2.000000

b=2

==============================

i=3

a=3.000000

b=3

==============================

i=4

a=4.000000

b=4

==============================

i=5

a=5.000000

b=5

==============================

i=6

a=6.000000

b=6

==============================

i=7

a=7.000000

b=7

==============================

i=8

a=8.000000

b=8

==============================

i=9

a=9.000000

b=9

==============================

i=10

a=10.000000

b=10

==============================

**Задание №2**

**Текст задачи:** выполнить пересылку массива объединений, имеющих поля int, double. Выполнить тоже самое, используя функции MPI\_Pack() и MPI\_Unpack()

**Используемые функции:**

**int** MPI\_Send (**void** \*buf, **int** count, MPI\_Datatype type, **int** dest, **int** tag, MPI\_Comm comm)

buf - адрес буфера памяти, в котором располагаются данные отправляемого сообщения,

count - количество элементов данных пересылаемого сообщения,

type - тип элементов данных пересылаемого сообщения,

dest - ранг процесса, которому отправляется сообщение,

tag - значение-тег, используемое для идентификации сообщения,

comm - коммуникатор, в рамках которого выполняется передача данных.

**int** MPI\_Recv (**void** \*buf, **int** count, MPI\_Datatype type, **int** source, **int** tag, MPI\_Comm comm, MPI\_Status \*status)

buf - адрес буфера памяти, в котором располагаются данные принимаемого сообщения,

count - количество элементов данных принимаемого сообщения;

type - тип элементов данных принимаемого сообщения;

source - ранг процесса, от которого должен быть выполнен прием сообщения;

tag - тег сообщения, которое должно быть принято для процесса;

comm - коммуникатор, в рамках которого выполняется передача данных;

status - указатель на структуру данных с информацией о результате выполнения.

**int** MPI\_Pack(**void** \*data, **int** count, MPI\_Datatype type, **void** \*buf, **int** bufsize, **int** \*bufpos, MPI\_Comm comm)

data - буфер памяти с элементами для упаковки;

count - количество элементов в буфере;

type - тип данных для упаковываемых элементов;

buf - буфер памяти для упаковки;

bufsize - размер буфера в байтах;

bufpos - позиция для начала записи в буфер (в байтах от начала буфера);

comm - коммуникатор для упакованного сообщения.

**int** MPI\_Unpack (**void** \*buf, **int** bufsize, **int** \*bufpos, **void** \*data, **int** count, MPI\_Datatype type, MPI\_Comm comm)

buf - буфер памяти с упакованными данными;

bufsize - размер буфера в байтах;

bufpos - позиция начала данных в буфере (в байтах от начала буфера);

data - буфер памяти для распаковываемых данных;

count - количество элементов в буфере;

type - тип распаковываемых данных;

comm - коммуникатор для упакованного сообщения.

**Текст программы:**

#include <mpi.h>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

//mpiexec -n 2 ConsoleApplication1.exe

#define comm MPI\_COMM\_WORLD

#define n 10

#define m 2

union

{

int a;

double b;

} un[n];

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

{

if (MPI\_Init(&argc, &argv) != MPI\_SUCCESS)

{

return 1;

}

int size, rank;

if (MPI\_Comm\_size(comm, &size) != MPI\_SUCCESS)

{

MPI\_Finalize();

return 2;

}

if (MPI\_Comm\_rank(comm, &rank) != MPI\_SUCCESS)

{

MPI\_Finalize();

return 3;

}

int msgtag = 12;

MPI\_Status status;

double tn0 = 0.0, tk0 = 0.0, tn1 = 0.0, tk1 = 0.0;

int mass = (n \* sizeof(double)), pos; //

int mass2 = (n \* sizeof(int)), pos2; //

if (!rank)

{

void\* buf = malloc(mass);

void\* buf2 = malloc(mass2);

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

{

un[i].b = rand() % 9;

}

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

{

un[i].a = rand() % 9;

}

FILE\* f;

fopen\_s(&f, "results1.dat", "w");

fprintf(f, "Печать исходного объединения\n");

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

{

fprintf(f, "un[%d].a = %i\n", i, un[i].a);

fprintf(f, "un[%d].b = %f\n", i, un[i].b);

}

fclose(f);

pos = 0;

pos2= 0;

tn0 = MPI\_Wtime();

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

{

MPI\_Pack(&un[i].a, 1, MPI\_INT, buf2, mass2, &pos2, comm); //

MPI\_Pack(&un[i].b, 1, MPI\_DOUBLE, buf, mass, &pos, comm); //

}

MPI\_Send(&pos, 1, MPI\_INT, 1, msgtag, comm);

MPI\_Send(&pos2, 1, MPI\_INT, 1, msgtag, comm);

MPI\_Send(buf, mass, MPI\_PACKED, 1, msgtag, comm);

MPI\_Send(buf2, mass2, MPI\_PACKED, 1, msgtag, comm);

tk0 = MPI\_Wtime();

}

else

{

pos = 0;

pos2 = 0;

int p = 0;

int p2 = 0;

tn1 = MPI\_Wtime();

MPI\_Recv(&pos, 1, MPI\_INT, 0, msgtag, comm, &status);

MPI\_Recv(&pos2, 1, MPI\_INT, 0, msgtag, comm, &status);

void\* mybuf2 = malloc(mass2);

void\* mybuf = malloc(mass);

MPI\_Recv(mybuf, pos, MPI\_PACKED, 0, msgtag, comm, &status);

MPI\_Recv(mybuf2, pos2, MPI\_PACKED, 0, msgtag, comm, &status);

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

{

MPI\_Unpack(mybuf2, pos2, &p2, &un[i].a, 1, MPI\_INT, comm);

MPI\_Unpack(mybuf, pos, &p, &un[i].b, 1, MPI\_DOUBLE, comm);

}

tk1 = MPI\_Wtime();

FILE\* z;

fopen\_s(&z, "results.dat", "w");

fprintf(z, "Печать распакованного объединения\n");

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

{

fprintf(z, "un[%d].a = %i\n", i, un[i].a);

fprintf(z, "un[%d].b = %f\n", i, un[i].b);

}

fclose(z);

}

if (!rank)

{

double tn = tn0 + tn1;

double tk = tk0 + tk1;

FILE\* o;

fopen\_s(&o, "time.dat", "w");

fprintf(o, "\ntime = %f\n", tk - tn);

fclose(o);

}

MPI\_Finalize();

return 0;

}

**Результаты:**

**results1.dat**

Печать исходного объединения

un[0].a = 8

un[0].b = 5.000000

un[1].a = 2

un[1].b = 8.000000

un[2].a = 7

un[2].b = 7.000000

un[3].a = 6

un[3].b = 4.000000

un[4].a = 7

un[4].b = 8.000000

un[5].a = 5

un[5].b = 1.000000

un[6].a = 7

un[6].b = 3.000000

un[7].a = 8

un[7].b = 0.000000

un[8].a = 3

un[8].b = 7.000000

un[9].a = 0

un[9].b = 2.000000

**results.dat**

Печать распакованного объединения

un[0].a = 8

un[0].b = 5.000000

un[1].a = 2

un[1].b = 8.000000

un[2].a = 7

un[2].b = 7.000000

un[3].a = 6

un[3].b = 4.000000

un[4].a = 7

un[4].b = 8.000000

un[5].a = 5

un[5].b = 1.000000

un[6].a = 7

un[6].b = 3.000000

un[7].a = 8

un[7].b = 0.000000

un[8].a = 3

un[8].b = 7.000000

un[9].a = 0

un[9].b = 2.000000

**time.dat**

time = 0.000346