ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра АСУ

Отчет

по лабораторной работе №2

по предмету «Параллельные вычисления»

на тему «Блокирующий и неблокирующий обмен. Дедлоки.»

Выполнил:

студент группы ИС-18б

Терещенко Вадим

Проверил:

Новиков Д. Д.

Донецк – 2021

**Цель работы:** Получить навыки работы в MPI с блокирующими и неблокирующими обменами данных.

**Индивидуальное задание**

Составить программу с использованием блокирующих и неблокирующих операций согласно варианту. Обеспечить выполнение операций в нескольких процессах. Раздача исходных данных должна выполняться с использованием неблокирующих операций, а сбор результатов – с помощью блокирующих.

|  |  |
| --- | --- |
| Вариант | Операции с векторами |
| 12 | A = B + C – D\*e |

Условные обозначения:

* A – вектор размерности N;
* e – экспонента;

**Листинг программы**

#include <stdio.h>

#include "mpi.h"

#include <iostream>

void initArray(int\* arr, int n) {

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

arr[i] = rand() % 10;

}

}

void printArray(int\* arr, int n) {

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

printf("%d ", arr[i]);

}

std::cout << std::endl;

}

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

{

MPI\_Request request;

MPI\_Status status;

int procRank, n, procNum, root = 0;

MPI\_Init(&argc, &argv);

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

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

int\* B = NULL;

int\* C = NULL;

int\* D = NULL;

int\* sizes = NULL;

float exp = 2.718;

int numOfParts = procNum - 1;

sizes = new int[numOfParts];

if (procRank == root) {

int step, residue;

std::cout << "Enter array size:";

std::cin >> n;

fflush(stdout);

B = new int[n];

C = new int[n];

D = new int[n];

initArray(B, n);

initArray(C, n);

initArray(D, n);

printf("Main arrays from process #%d:\n", procRank);

printf("Array B: ");

printArray(B, n);

printf("Array C: ");

printArray(C, n);

printf("Array D: ");

printArray(D, n);

step = n / numOfParts;

residue = n % numOfParts;

int currentProc = 1;

int temp = 0;

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

int subArraySize = n / numOfParts;

if(residue != 0) {

if (i == (numOfParts - 1)) {

subArraySize += residue;

}

}

int\* subB = new int[subArraySize];

int\* subC = new int[subArraySize];

int\* subD = new int[subArraySize];

for (int j = 0; j < subArraySize; j++) {

subB[j] = B[temp];

subC[j] = C[temp];

subD[j] = D[temp];

temp++;

}

sizes[currentProc - 1] = subArraySize;

MPI\_Isend(&subArraySize, 1, MPI\_INT, currentProc, 0, MPI\_COMM\_WORLD, &request);

MPI\_Isend(subB, subArraySize, MPI\_INT, currentProc, 0, MPI\_COMM\_WORLD, &request);

MPI\_Isend(subC, subArraySize, MPI\_INT, currentProc, 0, MPI\_COMM\_WORLD, &request);

MPI\_Isend(subD, subArraySize, MPI\_INT, currentProc, 0, MPI\_COMM\_WORLD, &request);

currentProc++;

}

int sizeOfPortion = NULL;

int iterator = 0;

int currentSize = NULL;

int\* A = NULL;

A = new int[n];

for (int i = 1; i < procNum; i++) {

int\* test = NULL;

currentSize = sizes[i - 1];

test = new int[currentSize];

MPI\_Recv(test, currentSize, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

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

A[iterator] = test[i];

iterator++;

}

sizeOfPortion = 0;

test = NULL;

}

printf("Processed array from process #%d:\n", procRank);

printArray(A, n);

} else {

int subArraySize;

int\* subA = NULL;

int\* subB = NULL;

int\* subC = NULL;

int\* subD = NULL;

MPI\_Irecv(&subArraySize, 1, MPI\_INT, root, 0, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status);

subA = new int[subArraySize];

subB = new int[subArraySize];

subC = new int[subArraySize];

subD = new int[subArraySize];

MPI\_Irecv(subB, subArraySize, MPI\_INT, root, 0, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status);

MPI\_Irecv(subC, subArraySize, MPI\_INT, root, 0, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status);

MPI\_Irecv(subD, subArraySize, MPI\_INT, root, 0, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status);

printf("Recieved part from Array B: ");

printArray(subB, subArraySize);

printf("Recieved part from Array C: ");

printArray(subC, subArraySize);

printf("Recieved part from Array D: ");

printArray(subD, subArraySize);

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

subA[i] = subB[i] + subC[i] - (subD[i] \* exp);

}

printf("Calculated part from Array A:");

printArray(subA, subArraySize);

MPI\_Send(subA, subArraySize, MPI\_INT, root, 0, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

return 0;

}

**Результаты работы программы**

> mpiexec -n 3 Laboratory\_work\_2.exe

Enter array size:10

Main arrays from process #0:

Array B: 1 7 4 0 9 4 8 8 2 4

Array C: 5 5 1 7 1 1 5 2 7 6

Array D: 1 4 2 3 2 2 1 6 8 5

Recieved part from Array B: 1 7 4 0 9

Recieved part from Array C: 5 5 1 7 1

Recieved part from Array D: 1 4 2 3 2

Calculated part from Array A:3 1 0 -1 4

Recieved part from Array B: 4 8 8 2 4

Recieved part from Array C: 1 5 2 7 6

Recieved part from Array D: 2 1 6 8 5

Calculated part from Array A:0 10 -6 -12 -3

Processed array from process #0:

3 1 0 -1 4 0 10 -6 -12 -3

> mpiexec -n 7 Laboratory\_work\_2.exe

Enter array size:10

Main arrays from process #0:

Array B: 1 7 4 0 9 4 8 8 2 4

Array C: 5 5 1 7 1 1 5 2 7 6

Array D: 1 4 2 3 2 2 1 6 8 5

Recieved part from Array B: 1

Recieved part from Array C: 5

Recieved part from Array D: 1

Calculated part from Array A:3

Recieved part from Array B: 7

Recieved part from Array C: 5

Recieved part from Array D: 4

Calculated part from Array A:1

Recieved part from Array B: 4

Recieved part from Array C: 1

Recieved part from Array D: 2

Calculated part from Array A:0

Recieved part from Array B: 0

Recieved part from Array C: 7

Recieved part from Array D: 3

Calculated part from Array A:-1

Recieved part from Array B: 9

Recieved part from Array C: 1

Recieved part from Array D: 2

Calculated part from Array A:4

Recieved part from Array B: 4 8 8 2 4

Recieved part from Array C: 1 5 2 7 6

Recieved part from Array D: 2 1 6 8 5

Calculated part from Array A:0 10 -6 -12 -3

Processed array from process #0:

3 1 0 -1 4 0 10 -6 -12 -3