МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №7

по дисциплине «Системное программирование»

**Основы MPI. Коллективные обмены в MPI.**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: | студент гр. ФИБ-3301-51-01 |  | / В. Р. Кочкин / |
| Проверил: | к.ф.-м.н. доцент каф. ПМиИ |  | / В. А. Бызов / |

Киров 2022

Цель работы

Получить навыки разработки параллельных программ с использованием коллективных обменов MPI.

Задания

**Задание 1**

Написать параллельную программу, вычисляющую ln 2 по формуле.

Реализовать

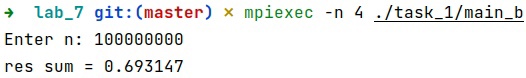
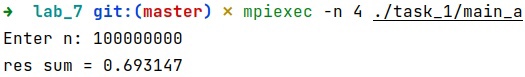
а) с использованием парных операций,

б) с использованием коллективных операций.

Замерить среднее время выполнения программы для n = 108, 5\*108, 109 (на каждое – не менее 3 запусков) для 1, 2, 4 и 8 процессов. Вычислить среднее ускорение для 2, 4 и 8 процессов. Рассчитать эффективность параллельного алгоритма. Построить графики ускорения и эффективности.

Сравнить результаты использования парных и коллективных операций.

**Полученные результаты**



Листинг программы приведен в [приложении А задание 1](#_Задание_1.).

**Задание 2**

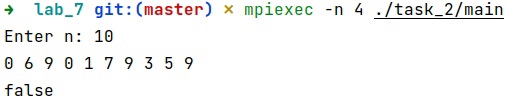
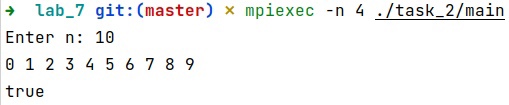
Написать параллельную программу, проверяющую, является ли заданный массив размера N (N ≤ 5\*108) упорядоченным.

Замерить среднее время выполнения программы для N = 108, 5\*108 (на каждое – не менее 3 запусков) для 1, 2, 4 и 8 процессов. Вычислить среднее ускорение для 2, 4 и 8 процессов. Рассчитать эффективность параллельного алгоритма.

Построить графики ускорения и эффективности.

**Полученные результаты**

*Проверка работы алгоритма.*



*Таблица 1 – Время выполнения программы, с*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность | Число процессов | | | |
| 1 | 2 | 4 | 8 |
| 108 | 1,95321 | 1,89774 | 1,86585 | 1,85251 |
| 5 \*108 | 9,97806 | 9,87481 | 9,17881 | 9,03427 |

Из таблицы видно, что программа быстрее всего исполнялась на 8 процессах.

*Таблица 2 – Ускорение выполнения программы, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Число процессов | | |
| 2 | 4 | 8 |
| 108 | 1,0292 | 1,0468 | 1,0543 |
| 5 \* 108 | 1,0104 | 1,0870 | 1,1045 |

Из таблицы видно, что максимальное ускорение достигается на 8 процессах, однако ускорение не достигает 8.

*График 1 – Ускорение выполнения программы, с*

Листинг программы приведен в [приложении А задание 2](#_Задание_2.).

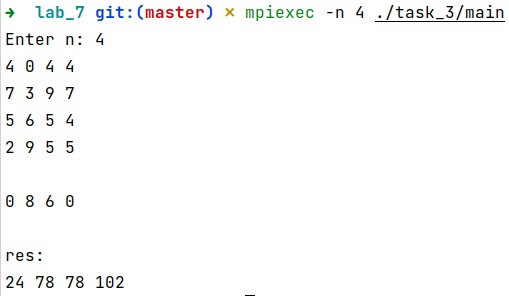
**Задание 3**

Написать параллельную программу, вычисляющую произведение квадратной матрицы размера N × N на вектор размера N (N ≤ 10000).

Замерить среднее время выполнения программы для N = 3000, 5000 и 10000 (на каждое – не менее 3 запусков) для 1, 4 и 16 процессов. Вычислить среднее ускорение для 4 и 16 процессов. Рассчитать эффективность параллельного алгоритма. Построить графики ускорения и эффективности.

**Полученные результаты**

*Проверка работы алгоритма.*



*Таблица 3 – Время выполнения программы, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Число процессов | | |
| 1 | 4 | 16 |
| 3000 | 0,316259 | 0,280926 | 0,57212 |
| 5000 | 0,832159 | 0,774594 | 1,64845 |
| 10000 | 3,35082 | 3,06419 | 6,17594 |

Из таблицы видно, что программа быстрее всего исполнялась на 4 процессах.

*Таблица 4 – Ускорение выполнения программы, с*

|  |  |  |
| --- | --- | --- |
| Размерность | Число процессов | |
| 4 | 16 |
| 3000 | 1,1258 | 0,5528 |
| 5000 | 1,0743 | 0,5048 |
| 10000 | 1,0935 | 0,5426 |

Из таблицы видно, что максимальное ускорение при 4 процессах, а при 16 ускорение достигает примерно 0,5.

*График 2 – Ускорение выполнения программы, с*

Листинг программы приведен в [приложении А задание 3](#_Задание_3.).

**Задание 4**

Написать параллельную программу, в которой процессы с чётными номерами вычисляют минимальное значение в массиве из M элементов (M ≤ 5\*108), а процессы с нечётными номерами вычисляют скалярное произведение векторов размера N (N ≤ 5\*108). Числа M и N вводятся пользователем. Продемонстрировать корректность работы программы.

**Полученные результаты**



Листинг программы приведен в [приложении А задание 4](#_Задание_4.).

Вывод

В ходе лабораторной работы я получил навыки разработки параллельных программ с использованием коллективных обменов MPI.

# **Приложения**

## **Приложение А. Листинги программ**

### Задание 1.

main\_a.cpp

*//  
// Created by vladislav on 19.05.22.  
//*#include <iostream>  
#include <mpich/mpi.h>  
#include <cstdlib>  
#include <cmath>  
#include <vector>  
  
int main(int argc, char \*\*argv) {  
 MPI\_Init(&argc, &argv);  
  
 MPI\_Status status;  
  
 int proc\_num;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  
  
 int proc\_rank;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  
  
 if (proc\_rank == 0) {  
 int n = 0;  
  
 std::cout << "Enter n: ";  
 std::cin >> n;  
  
 if (n < 1 || n > 5 \* 1e9) {  
 printf("wrong n");  
 return -1;  
 }  
  
 int countSums = n / (proc\_num - 1);  
 int countSumsLast = countSums + n % (proc\_num - 1);  
 for (int i = 1; i < proc\_num; ++i) {  
 int k = (i - 1) \* countSums + 1;  
 MPI\_Send(i != proc\_num - 1 ? &countSums : &countSumsLast, 1, MPI\_INT, i, 0,MPI\_COMM\_WORLD);  
 MPI\_Send(&k, 1, MPI\_INT, i, 0,MPI\_COMM\_WORLD);  
 }  
  
 std::vector<double> sums;  
 for (int i = 1; i < proc\_num; ++i) {  
 double sum = 0;  
 MPI\_Recv(&sum, 1, MPI\_DOUBLE, i, MPI\_ANY\_TAG,MPI\_COMM\_WORLD, &status);  
 sums.push\_back(sum);  
 }  
  
 double resSum = 0;  
 for(double sum : sums) {  
 resSum += sum;  
 }  
 printf("res sum = %f\n", resSum);  
 } else {  
 int n;  
 MPI\_Recv(&n, 1, MPI\_INT, 0, 0,MPI\_COMM\_WORLD, &status);  
  
 int k;  
 MPI\_Recv(&k, 1, MPI\_INT, 0, 0,MPI\_COMM\_WORLD, &status);  
  
 double sum = 0;  
 for(int i = k; i < k + n; i++) {  
 sum += pow(-1, i - 1) / i;  
 }  
  
 MPI\_Send(&sum, 1, MPI\_DOUBLE, 0, proc\_rank,MPI\_COMM\_WORLD);  
 }  
  
 MPI\_Finalize();  
 return **EXIT\_SUCCESS**;  
}

main\_b.cpp

*//  
// Created by vladislav on 19.05.22.  
//*#include <iostream>  
#include <mpich/mpi.h>  
#include <cstdlib>  
#include <cmath>  
  
int main(int argc, char \*\*argv) {  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  
  
 int proc\_rank;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  
  
 int n = 0;  
 if (proc\_rank == 0) {  
 std::cout << "Enter n: ";  
 std::cin >> n;  
  
 if (n < 1 || n > 5 \* 1e9) {  
 printf("wrong n");  
 return -1;  
 }  
 }  
  
 MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 int countSums = proc\_rank != proc\_num - 1 ? n / (proc\_num - 1) : n / (proc\_num - 1) + n % (proc\_num - 1);  
 int k = proc\_rank == 0 ? 1 : proc\_rank \* countSums + 1;  
  
 double sum = 0;  
 for (int i = k; i < k + countSums; ++i) {  
 sum += pow(-1, i - 1) / i;  
 }  
  
 double resSum = 0;  
 MPI\_Reduce(&sum,&resSum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);  
  
 if (proc\_rank == 0) {  
 printf("res sum = %f\n", resSum);  
 }  
  
 MPI\_Finalize();  
 return **EXIT\_SUCCESS**;  
}

### Задание 2.

*//  
// Created by vladislav on 19.05.22.  
//*#include <iostream>  
#include <mpich/mpi.h>  
#include <cstdlib>  
#include <vector>  
  
std::vector<int> create\_vector(int);  
void print\_vector(const std::vector<int>&);  
  
int main(int argc, char \*\*argv) {  
 srand(time(nullptr));  
  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  
  
 int proc\_rank;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  
  
 int n = 0;  
 std::vector<int> counts;  
 std::vector<int> offsets;  
 std::vector<int> vector;  
  
 if (proc\_rank == 0) {  
 std::cout << "Enter n: ";  
 std::cin >> n;  
  
 if (n < 1 || n > 5 \* 1e8) {  
 printf("wrong n");  
 return -1;  
 }  
  
 vector = create\_vector(n);  
 print\_vector(vector);  
  
 for (int i = 1; i < proc\_num - 1; ++i) {  
 int offset = i \* n / proc\_num;  
 if (vector[offset - 1] > vector[offset]) {  
 printf("vector not sorted\n");  
 MPI\_Finalize();  
 return **EXIT\_SUCCESS**;  
 }  
 }  
 }  
  
 MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 for (int i = 0; i < proc\_num; ++i) {  
 int partSize = i != proc\_num - 1 ? n / proc\_num : n / proc\_num + n % proc\_num;  
 int offset = i \* (n / proc\_num);  
 counts.push\_back(partSize);  
 offsets.push\_back(offset);  
 }  
  
 std::vector<int> part(counts[proc\_rank]);  
 MPI\_Scatterv(vector.data(), counts.data(), offsets.data(), MPI\_INT,  
 part.data(), counts[proc\_rank], MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 print\_vector(part);  
  
 bool isSorted = true;  
 for (int i = 0; i < part.size() - 1; ++i) {  
 if (part[i] > part[i + 1]) {  
 isSorted = false;  
 break;  
 }  
 }  
  
 bool resSorted;  
 MPI\_Reduce(&isSorted,&resSorted, 1, MPI\_C\_BOOL, MPI\_LAND, 0, MPI\_COMM\_WORLD);  
  
 if (proc\_rank == 0) {  
 if (resSorted) {  
 printf("true\n");  
 } else {  
 printf("false\n");  
 }  
 }  
  
 MPI\_Finalize();  
 return **EXIT\_SUCCESS**;  
}  
  
std::vector<int> create\_vector(int n) {  
 std::vector<int> vector;  
  
 for (int i = 0; i < n; ++i) {  
 *//vector.push\_back(i);* vector.push\_back(rand() % 10);  
 }  
  
 return vector;  
}  
  
void print\_vector(const std::vector<int>& vec) {  
 for(int el : vec) {  
 std::cout << el << " ";  
 }  
 std::cout << std::endl;  
}

### Задание 3.

*//  
// Created by vladislav on 02.06.22.  
//*#include <iostream>  
#include <mpich/mpi.h>  
#include <cstdlib>  
#include <vector>  
  
std::vector<int> create\_matrix(int);  
std::vector<int> create\_vector(int);  
void print\_matrix(const std::vector<int>&, int n);  
void print\_vector(const std::vector<int>&);  
  
int main(int argc, char \*\*argv) {  
 srand(time(nullptr));  
  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  
  
 int proc\_rank;  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  
  
 int n = 0;  
 if (proc\_rank == 0) {  
 std::cout << "Enter n: ";  
 std::cin >> n;  
  
 if (n < 1 || n > 10000) {  
 printf("wrong n");  
 return -1;  
 }  
 }  
  
 MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 std::vector<int> matrix;  
 std::vector<int> vector(n);  
  
 if (proc\_rank == 0) {  
 matrix = create\_matrix(n);  
 print\_matrix(matrix, n);  
 printf("\n");  
  
 vector = create\_vector(n);  
 print\_vector(vector);  
 printf("\n");  
 }  
  
 MPI\_Bcast(vector.data(), n, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 int countRows = n / proc\_num;  
  
 std::vector<int> rows(countRows \* n);  
 MPI\_Scatter(matrix.data(), countRows \* n, MPI\_INT,  
 rows.data(), countRows \* n, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 std::vector<int> result(countRows);  
 for(int i = 0; i < countRows; i++) {  
 int sum = 0;  
 for (int j = 0; j < n; j++) {  
 sum += rows[j + i \* n] \* vector[j];  
 }  
 result[i] = sum;  
 }  
  
 print\_vector(result);  
  
 std::vector<int> allResult(n);  
 MPI\_Gather(result.data(), countRows, MPI\_INT,  
 allResult.data(), countRows, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 if (proc\_rank == 0) {  
 printf("res:\n");  
 print\_vector(allResult);  
 }  
  
 MPI\_Finalize();  
 return **EXIT\_SUCCESS**;  
}  
  
std::vector<int> create\_matrix(int n) {  
 std::vector<int> matrix;  
  
 for (int i = 0; i < n; ++i) {  
 auto row = create\_vector(n);  
 for (int j = 0; j < n; ++j) {  
 matrix.push\_back(row[j]);  
 }  
 }  
  
 return matrix;  
}  
  
std::vector<int> create\_vector(int n) {  
 std::vector<int> vector;  
  
 for (int i = 0; i < n; ++i) {  
 vector.push\_back(rand() % 10);  
 }  
  
 return vector;  
}  
  
void print\_matrix(const std::vector<int>& matrix, int n) {  
 for (int i = 0; i < matrix.size(); ++i) {  
 if (i != 0 && i % n == 0) {  
 printf("\n");  
 }  
 printf("%d ", matrix[i]);  
 }  
  
 printf("\n");  
}  
  
void print\_vector(const std::vector<int>& vec) {  
 for(int el : vec) {  
 std::cout << el << " ";  
 }  
 std::cout << std::endl;  
}

### Задание 4.

*//  
// Created by vladislav on 02.06.22.  
//*#include <iostream>  
#include <mpich/mpi.h>  
#include <cstdlib>  
#include <vector>  
#include <random>  
  
std::vector<int> create\_vector(int);  
void print\_vector(const std::vector<int>&);  
  
int main(int argc, char\*\* argv)  
{  
 srand(time(nullptr));  
  
 int proc\_rank, proc\_num, n, m;  
 std::vector<int> vec1, vec\_recv1;  
 std::vector<int> vec2, vec\_recv2, vec3, vec\_recv3;  
  
 int localMin = 1e8, globalMin = 0;  
 int localRes = 0, globalRes;  
  
 int color, key;  
  
 MPI\_Init(&argc, &argv);  
  
 MPI\_Comm new\_comm;  
  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);  
  
 if (proc\_rank % 2 == 0)  
 {  
 key = proc\_rank;  
 color = 0;  
 }  
 else  
 {  
 key = proc\_num - proc\_rank;  
 color = 1;  
 }  
  
 MPI\_Comm\_split(MPI\_COMM\_WORLD, color, key, &new\_comm);  
 MPI\_Comm\_size(new\_comm, &proc\_num);  
  
 if (proc\_rank == 0)  
 {  
 std::cout << "Enter n: ";  
 std::cin >> n;  
  
 if (n < 1 || n > 1e8) {  
 printf("wrong n");  
 return -1;  
 }  
  
 std::cout << "Enter m: ";  
 std::cin >> m;  
  
 if (n < 1 || n > 1e8) {  
 printf("wrong m");  
 return -1;  
 }  
 }  
  
 MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
 MPI\_Bcast(&m, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
  
 vec1 = create\_vector(m);  
 vec2 = create\_vector(n);  
 vec3 = create\_vector(n);  
  
 if (proc\_rank == 0) {  
 printf("vec1:\n");  
 print\_vector(vec1);  
  
 printf("vec2:\n");  
 print\_vector(vec2);  
  
 printf("vec3:\n");  
 print\_vector(vec3);  
 }  
  
 if (proc\_rank % 2 == 0)  
 {  
 vec\_recv1.resize(m / proc\_num);  
 MPI\_Scatter(vec1.data(), m / proc\_num, MPI\_INT,  
 vec\_recv1.data(), m / proc\_num, MPI\_INT, 0, new\_comm);  
  
 for (int el : vec\_recv1)  
 {  
 if (el < localMin) {  
 localMin = el;  
 }  
 }  
  
 MPI\_Reduce(&localMin, &globalMin, 1, MPI\_INT, MPI\_MIN, 0, new\_comm);  
 if (proc\_rank == 0) {  
 printf("min = %d\n", globalMin);  
 }  
 }  
 else  
 {  
 vec\_recv2.resize(n / proc\_num);  
 MPI\_Scatter(vec2.data(), n / proc\_num, MPI\_INT,  
 vec\_recv2.data(), n / proc\_num, MPI\_INT, 0, new\_comm);  
  
 vec\_recv3.resize(n / proc\_num);  
 MPI\_Scatter(vec3.data(), n / proc\_num, MPI\_INT,  
 vec\_recv3.data(), n / proc\_num, MPI\_INT, 0, new\_comm);  
  
 for (size\_t i = 0; i < vec\_recv2.size(); i++)  
 {  
 localRes += vec\_recv2[i] \* vec\_recv3[i];  
 }  
  
 MPI\_Reduce(&localRes, &globalRes, 1, MPI\_INT, MPI\_SUM, 1, new\_comm);  
 if (proc\_rank == 1) {  
 printf("res = %d\n", globalRes);  
 }  
 }  
  
 MPI\_Finalize();  
 return **EXIT\_SUCCESS**;  
}  
  
std::vector<int> create\_vector(int n) {  
 std::vector<int> vector;  
  
 for (int i = 0; i < n; ++i) {  
 vector.push\_back(rand() % 10);  
 }  
  
 return vector;  
}  
  
void print\_vector(const std::vector<int>& vec) {  
 for(int el : vec) {  
 std::cout << el << " ";  
 }  
 std::cout << std::endl;  
}