МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №6 ПО ДИСЦИПЛИНЕ «РАСПРЕДЕЛЕННЫЕ ЗАДАЧИ И АЛГОРИТМЫ»**

Отчет подготовил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. Е. Павликов

(подпись, дата)

Факультет компьютерных технологий и прикладной математики,

4 курс, 45 группа

Отчет приняла\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.А. Приходько

(подпись, дата)

Краснодар

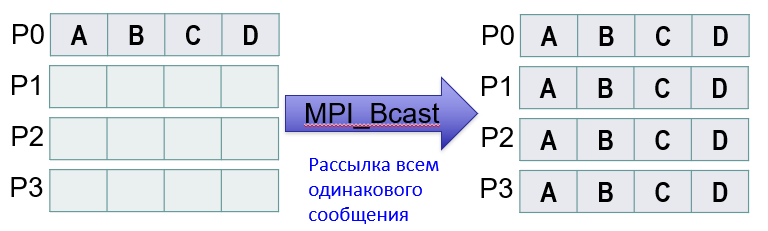
2019

**Цель работы:**

Решить задачу о нахождении скалярного произведения векторов А и В с учетом знания принципов коллективных обменов. В качестве таких принципов были выбраны: Broadcast и Scatter.

**Ход работы:**

1. Широковещательная рассылка Broadcast.



**MPI.COMM\_WORLD.Bcast (buf,offset,count,datatype,root)** – рассылка всем процессам сообщения buf  
Если номер процесса совпадает с root, то он отправитель, иначе – приемник

Нулевой процесс рассылает всем оба вектора А и В, после чего остальные процессы принимают данные и, исходя из своего номера, работают с определенными данными. Финальный результат формируется в нулевом процессе при помощи функции Reduce.

Код программы:

import mpi.MPI;

import java.util.Arrays;

public class BroadCast {

public static void main(String[] args) {

MPI.Init(args);

int rank = MPI.COMM\_WORLD.Rank();

int size = MPI.COMM\_WORLD.Size();

int root = 0;

int N = 1000;

int threadsCount = size - 1;

int k = N / threadsCount + (N % threadsCount != 0 ? 1 : 0);

int[] a = Utils.getRandomArray(N);

int[] b = Utils.getRandomArray(N);

long startTime = System.currentTimeMillis();

MPI.COMM\_WORLD.Bcast(a, 0, a.length, MPI.INT, 0);

MPI.COMM\_WORLD.Bcast(b, 0, a.length, MPI.INT, 0);

int start = getStartIndex(rank, k);

int step = getStep(start, k, N);

int count = start + step;

int[] recvA = Arrays.copyOfRange(a, start, count);

int[] recvB = Arrays.copyOfRange(b, start, count);

int arrSize = recvA.length; // doesn't matter recvA or recvB (they have same size)

int[] result = {0};

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

if (rank != root) {

result[0] += recvA[i] \* recvB[i];

}

}

int[] globalResult = new int[1];

MPI.COMM\_WORLD.Reduce(result, 0, globalResult, 0, 1, MPI.INT, MPI.SUM, 0);

if (rank == 0) {

long endTime = System.currentTimeMillis();

System.out.println("Work time: " + (endTime - startTime));

System.out.println("Result: " + (globalResult[0] == Utils.vectorsMultiplication(a, b)));

}

MPI.Finalize();

}

private static int getStartIndex(int rank, int k) {

if (rank != 0) {

return (rank - 1) \* k;

}

return 0;

}

private static int getStep(int start, int k, int N) {

int end = start + k;

if (N < end) {

end = N;

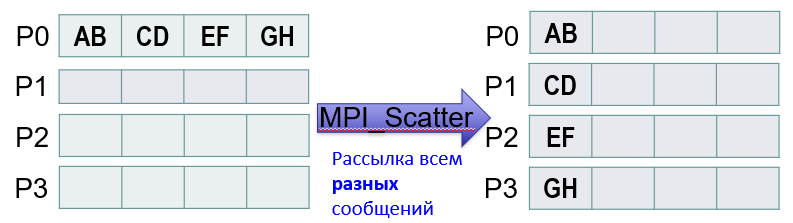
}

return end - start;

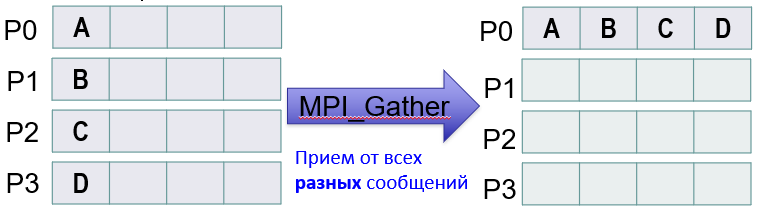
}

}

1. Рассылка частей данных Scatter.



**MPI.COMM\_WORLD.Scatter(sendbuff,sendoffset,sendcount,recvbuff,recvoffset,recvcount,datatype,root).** Сбор данных и формирование результата происходит при помощи вызова подпрограммы Gather.



Код программы:

import mpi.MPI;

public class Scatter {

public static void main(String[] args) {

MPI.Init(args);

int rank = MPI.COMM\_WORLD.Rank();

int size = MPI.COMM\_WORLD.Size();

int root = 0;

int N = 1000;

int k = N / size + (N % size != 0 ? 1 : 0);

int[] a = Utils.getRandomArray(N);

int[] b = Utils.getRandomArray(N);

int start = getStartIndex(rank, k);

int step = getStep(start, k, N);

int[] recvA = new int[step];

int[] recvB = new int[step];

long startTime = System.currentTimeMillis();

MPI.COMM\_WORLD.Scatter(a, 0, step, MPI.INT, recvA, 0, step, MPI.INT, root);

MPI.COMM\_WORLD.Scatter(b, 0, step, MPI.INT, recvB, 0, step, MPI.INT, root);

int[] result = {0};

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

result[0] += recvA[i] \* recvB[i];

}

int[] gather = new int[size];

MPI.COMM\_WORLD.Gather(result, 0, result.length, MPI.INT, gather, 0, 1, MPI.INT, root);

if (rank == root) {

int[] globalResult = new int[1];

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

globalResult[0] += gather[i];

}

System.out.println("Result: " + (globalResult[0] == Utils.vectorsMultiplication(a, b)));

long endTime = System.currentTimeMillis();

System.out.println("Work time: " + (endTime - startTime));

}

}

private static int getStartIndex(int rank, int k) {

return rank \* k;

}

private static int getStep(int start, int k, int N) {

int end = start + k;

if (N < end) {

end = N;

}

return end - start;

}

}

**Графики:**

Сравнение кривых времени выполнения программы при помощи Broadcast и Scatter.

Из графика видно, что Scatter значительнее проигрывает на больших потоках, особенно заметно это 50 потоке. Данная закономерность дает возможность предположить, что на большем количестве потоков Broadcast может обойти Scatter по скорости выполнения программы.

**Результаты и выводы:**

Выполнив лабораторную работу, я познакомился с такими методами рассылок, как Broadcast и Scatter, реализовал выполнение программы с их помощью, а также проанализировал скорость выполнения программы обоими методами на различных количествах потоков.