Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Распределенные задачи и алгоритмы**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Нагалевский

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.И. Шиян

**Тема работы:** Вычисление скалярного произведения векторов, a и b – рассылаются всем процессам равными частями.

**Постановка задачи:**

Два вектора a и b размерности N представлены двумя одномерными массивами, содержащими каждый по N элементов. Решите задачу о нахождении скалярного произведения векторов A и B с учетом знания принципов коллективных обменов:

• с помощью функции Broadcast/Reduse;

Распределим данные из корневого процесса на все процессы в коммуникаторе с помощью Scatter:

int localN = N / size;

int[] localA = new int[localN];

int[] localB = new int[localN];

MPI.COMM\_WORLD.Scatter(A, 0, localN, MPI.INT, localA, 0, localN, MPI.INT, 0);

MPI.COMM\_WORLD.Scatter(B, 0, localN, MPI.INT, localB, 0, localN, MPI.INT, 0);

Далее вычисляем локальный скалярный произведение векторов для каждого процесса:

int[] localResult = new int[1];

for (int i = 0; i < 1; i++)

localResult[0] = calcLocalScalar(localA, localB, localN);

Производит сумирование локальных результатов с использованием операции MPI.SUM:

int[] globalResult = new int[1];

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

Далее на 0 ранге производим вывод результатов и подсчет итогового времени.

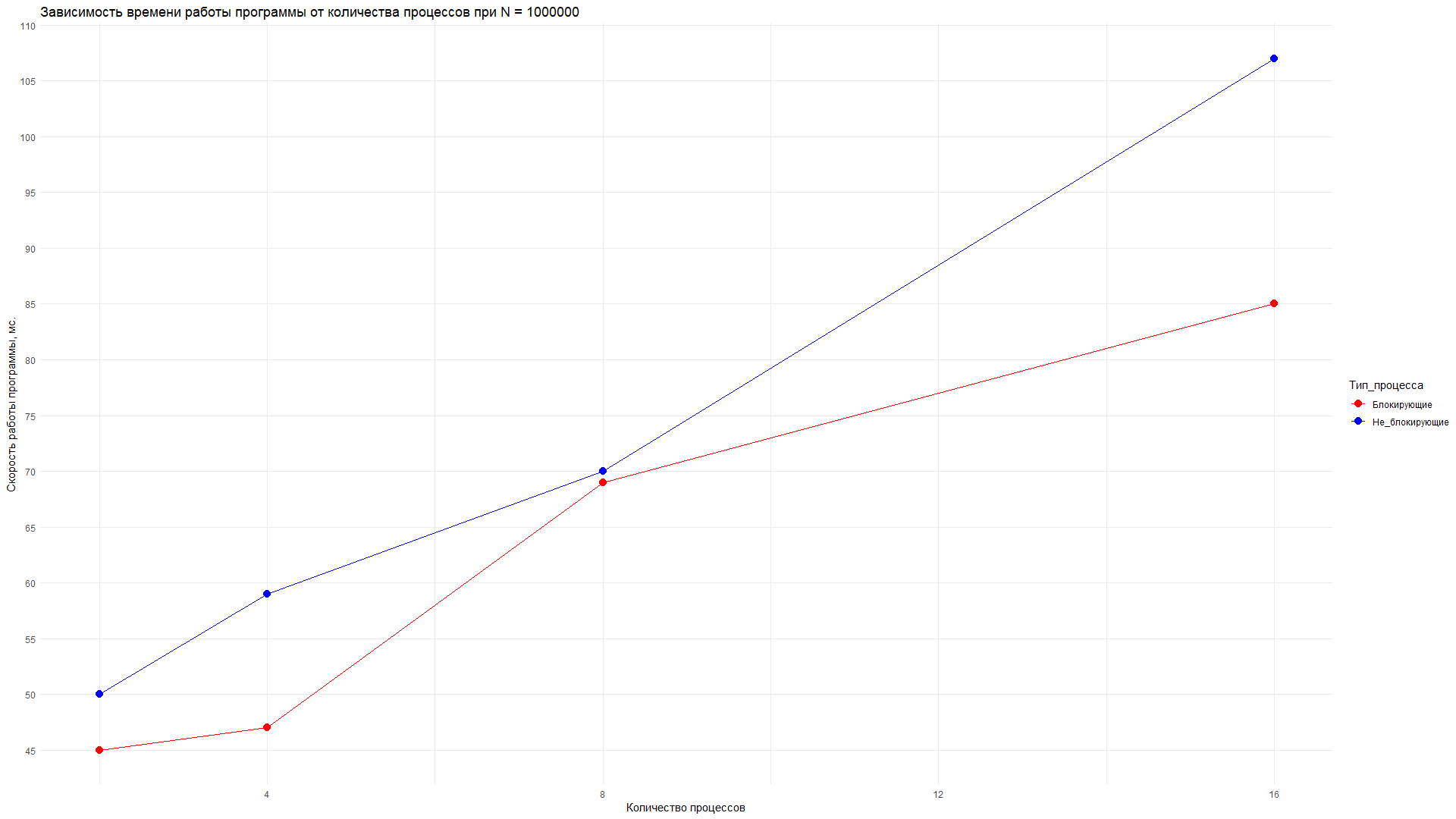
Также было проведено сравнение данного кода и решение задачи с помощью асинхронных методов решения. На рисунке 1 приведен график зависимости времени работы программы от количества процессов при N=100000.   


Рисунок 1 – График зависимости.

Таким образом, программа решает задачу вычисления скалярного произведения двух векторов с использованием параллельных вычислений на нескольких процессах. Каждый процесс выполняет часть вычислений, после чего результаты суммируются для получения общего результата.

**Листинг программ**

Файл Main.java

import mpi.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 MPI.*Init*(args);  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 long startTime, endTime;  
 startTime = System.*currentTimeMillis*();  
  
 int N = 1000000;  
  
 int[] A = new int[N];  
 int[] B = new int[N];  
  
 if (rank == 0) {  
 *initializeVectors*(A, B, N);  
 }  
  
 // Рассылка векторов A и B на все процессы  
 MPI.*COMM\_WORLD*.Bcast(A, 0, N, MPI.*INT*, 0);  
 MPI.*COMM\_WORLD*.Bcast(B, 0, N, MPI.*INT*, 0);  
  
 int localN = N / size;  
 int[] localA = new int[localN];  
 int[] localB = new int[localN];  
  
 //распределения данных из корневого процесса на все процессы в коммуникаторе  
 MPI.*COMM\_WORLD*.Scatter(A, 0, localN, MPI.*INT*, localA, 0, localN, MPI.*INT*, 0);  
 MPI.*COMM\_WORLD*.Scatter(B, 0, localN, MPI.*INT*, localB, 0, localN, MPI.*INT*, 0);  
  
 int[] localResult = new int[1];  
 for (int i = 0; i < 1; i++)  
 localResult[0] = *calcLocalScalar*(localA, localB, localN);  
  
 int[] globalResult = new int[1];  
  
 MPI.*COMM\_WORLD*.Reduce(localResult, 0, globalResult, 0, 1, MPI.*INT*, MPI.*SUM*, 0);  
  
 endTime = System.*currentTimeMillis*();  
  
 if (rank == 0) {  
 System.*out*.println("Result: " + globalResult[0]);  
 System.*out*.println("The program finished its work in " + (endTime - startTime) + " ms. and the number of processes " + size);  
 }  
  
 MPI.*Finalize*();  
 }  
  
 private static void initializeVectors(int[] A, int[] B, int N) {  
 for (int i = 0; i < N; i++) {  
 A[i] = (int) (Math.*random*() \* 100);  
 B[i] = (int) (Math.*random*() \* 100);  
 }  
 }  
  
 private static int calcLocalScalar(int[] A, int[] B, int N) {  
 int result = 0;  
 for (int i = 0; i < N; i++) {  
 result += A[i] \* B[i];  
 }  
 return result;  
 }  
}

Файл Asynch.java:

import mpi.\*;  
  
public class Asynch {  
 public static void main(String[] args) {  
 MPI.*Init*(args);  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 long startTime, endTime;  
 startTime = System.*currentTimeMillis*();  
  
 int N = 1000000;  
  
 int[] A = new int[N];  
 int[] B = new int[N];  
  
 if (rank == 0) {  
 *initializeVectors*(A, B, N);  
 }  
  
 // Асинхронная рассылка векторов A и B на все процессы  
 final Thread bcastThreadA = new Thread(() -> MPI.*COMM\_WORLD*.Bcast(A, 0, N, MPI.*INT*, 0));  
 final Thread bcastThreadB = new Thread(() -> MPI.*COMM\_WORLD*.Bcast(B, 0, N, MPI.*INT*, 0));  
  
 bcastThreadA.start();  
 bcastThreadB.start();  
  
 // Ждем завершения асинхронных рассылок  
 try {  
 bcastThreadA.join();  
 bcastThreadB.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 int localN = N / size;  
 int[] localA = new int[localN];  
 int[] localB = new int[localN];  
  
 // Асинхронное распределение данных из корневого процесса на все процессы в коммуникаторе  
 final Thread scatterThreadA = new Thread(() -> MPI.*COMM\_WORLD*.Scatter(A, 0, localN, MPI.*INT*, localA, 0, localN, MPI.*INT*, 0));  
 final Thread scatterThreadB = new Thread(() -> MPI.*COMM\_WORLD*.Scatter(B, 0, localN, MPI.*INT*, localB, 0, localN, MPI.*INT*, 0));  
  
 scatterThreadA.start();  
 scatterThreadB.start();  
  
 // Ждем завершения асинхронных разделений  
 try {  
 scatterThreadA.join();  
 scatterThreadB.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 int[] localResult = new int[1];  
 for (int i = 0; i < 1; i++)  
 localResult[0] = *calcLocalScalar*(localA, localB, localN);  
  
 int[] globalResult = new int[1];  
  
 // Асинхронная редукция  
 final Thread reduceThread = new Thread(() -> MPI.*COMM\_WORLD*.Reduce(localResult, 0, globalResult, 0, 1, MPI.*INT*, MPI.*SUM*, 0));  
 reduceThread.start();  
  
 // Ждем завершения асинхронной редукции  
 try {  
 reduceThread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 endTime = System.*currentTimeMillis*();  
  
 if (rank == 0) {  
 System.*out*.println("Result: " + globalResult[0]);  
 System.*out*.println("The program finished its work in " + (endTime - startTime) + " ms. and the number of processes " + size);  
 }  
  
 MPI.*Finalize*();  
 }  
  
 private static void initializeVectors(int[] A, int[] B, int N) {  
 for (int i = 0; i < N; i++) {  
 A[i] = (int) (Math.*random*() \* 100);  
 B[i] = (int) (Math.*random*() \* 100);  
 }  
 }  
  
 private static int calcLocalScalar(int[] A, int[] B, int N) {  
 int result = 0;  
 for (int i = 0; i < N; i++) {  
 result += A[i] \* B[i];  
 }  
 return result;  
 }  
}