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 №5**

**Дисциплина: Распределенные программные системы**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р. Р. Посевин

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

Краснодар

2021

**Постановка задачи**

Два вектора a и b размерности N представлены двумя одномерными массивами, содержащими каждый по N элементов. Напишите параллельную MPI-программу вычисления скалярного произведения этих векторов используя блокирующий двухточечный обмен сообщениями. Программа должна быть организована по схеме master-slave, причем master-процесс должен пересылать подчиненным процессам одинаковые по количеству элементов фрагменты векторов.

1. Измерьте и проанализируйте затраченное время во всех случаях.
2. Проведите исследование зависимости ускорения параллельной программы от размера сообщения (графики).
3. Сделайте то же самое для других вариантов блокирующих обменов (буферизированным, синхронизированным, по готовности).
4. Проанализируйте вариант использования неблокирующих функций и реализуйте его.
5. Реализуйте вариант с отложенными обменами.

# **Ход работы**

# Кол-во потоков выбиралось следующим: 5, 20, 50, 100. Размерность векторов – 10000 элементов. Мастер-процесс разделял вектора на равные части, и отправлял другим одну из частей. В оставшихся потоках производились вычисления и результат обратно возвращался мастеру. Затем полученные значения суммировались. Итоговая сумма – скалярное произведение векторов. Было проведено исследование.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 1 – Результаты для блокирующих обменов

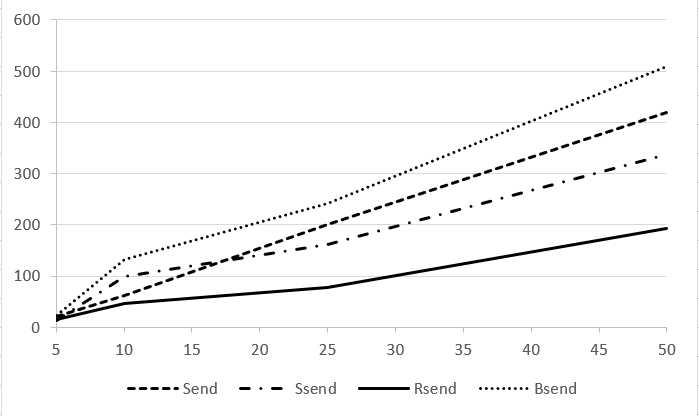


Рисунок 2 – График для блокирующих обменов

Из блокирующих самым медленным оказался буферизированный обмен. Дело в том, что выполняются дополнительные операции пересылки, выделение памяти для обмена информации буфером. Также сами операции выделения буферной памяти занимают некоторое вычислительное время.

Самым быстрым оказался обмен по готовности, ведь во время выполнения операций rsend, не выполняются никакие дополнительные проверки и пересылки, кроме одной единственной - отправки.

Для неблокирующих результаты получились следующими:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3 - Результаты для неблокирующих обменов

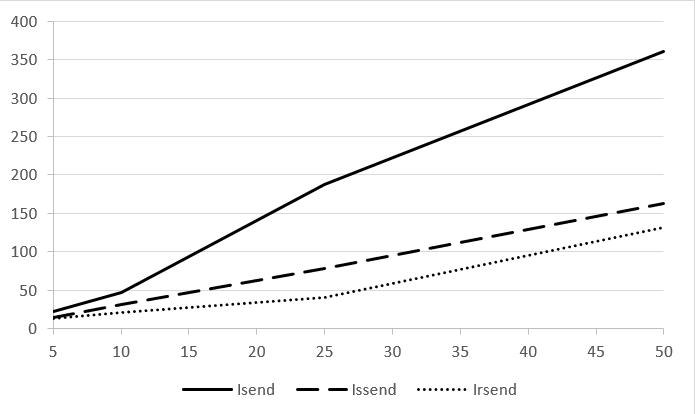


Рисунок 4 – График для неблокирующих обменов

Для неблокирующих видим колоссальную разницу, rsend снова показал себя лучше всех, но и общее время на фоне блокирующих значительно уменьшилось.

Код программы:

import os  
import logging  
import random  
import time  
  
import numpy  
from mpi4py import MPI  
  
from mpi\_utils import MPI\_Isend, MPI\_Ssend, MPI\_Issend, MPI\_Bsend, MPI\_Ibsend, MPI\_Rsend, MPI\_Irsend, MPI\_Send  
  
# Run command  
# mpiexec -np 2 py lab\_5.py  
  
SEND\_MODE = 'is'  
VECTORS\_MODE = 'RANDOM'  
TAG = 2  
VECTOR\_A = [1, 2, 3, 4, 5, 1]  
VECTOR\_B = [1, 1, 1, 1, 2, 1]  
  
VECTORS\_LEN = 10000  
MIN\_NUMBER = -100  
MAX\_NUMBER = 100  
  
file\_name = os.path.basename(\_\_file\_\_)  
# logger = logging.getLogger(file\_name)  
# logging.basicConfig(  
# filename='logs/logs.log',  
# level=logging.INFO,  
# format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',  
# )  
comm = MPI.COMM\_WORLD  
rank = comm.Get\_rank()  
size = comm.Get\_size()  
  
  
def send\_message(comm: MPI.Intracomm, \*\*kwargs):  
 if SEND\_MODE == 'i':  
 return MPI\_Isend(comm=comm, \*\*kwargs)  
 if SEND\_MODE == 's':  
 return MPI\_Ssend(comm=comm, \*\*kwargs)  
 if SEND\_MODE == 'is':  
 return MPI\_Issend(comm=comm, \*\*kwargs)  
 if SEND\_MODE == 'b':  
 return MPI\_Bsend(comm=comm, \*\*kwargs)  
 if SEND\_MODE == 'ib':  
 return MPI\_Ibsend(comm=comm, \*\*kwargs)  
 if SEND\_MODE == 'r':  
 return MPI\_Rsend(comm=comm, \*\*kwargs)  
 if SEND\_MODE == 'ir':  
 return MPI\_Irsend(comm=comm, \*\*kwargs)  
  
 return MPI\_Send(comm=comm, \*\*kwargs)  
  
  
def recv(comm: MPI.Intracomm, \*\*kwargs):  
 if 'i' in SEND\_MODE:  
 return comm.irecv(\*\*kwargs)  
  
 return comm.recv(\*\*kwargs)  
  
  
if VECTORS\_MODE == 'RANDOM':  
 vectors\_len = VECTORS\_LEN  
 vector\_a = numpy.array([random.randint(MIN\_NUMBER, MAX\_NUMBER) for \_ in range(vectors\_len)])  
 vector\_b = numpy.array([random.randint(MIN\_NUMBER, MAX\_NUMBER) for \_ in range(vectors\_len)])  
else:  
 vectors\_len = len(VECTOR\_A)  
 vector\_a = numpy.array(VECTOR\_A)  
 vector\_b = numpy.array(VECTOR\_B)  
  
if len(VECTOR\_A) != len(VECTOR\_B):  
 raise Exception('Vectors has different lengths')  
  
if size - 1 > vectors\_len:  
 raise Exception('Too many processes')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 vector\_split\_len = vectors\_len // (size - 1)  
 if rank == 0:  
 start\_time = time.time()  
 print('Vector A: {}'.format(vector\_a))  
 print('Vector B: {}'.format(vector\_b))  
  
 # Разбиение векторов на части для процессов  
 for proc\_rank in range(1, size):  
 if proc\_rank == size - 1:  
 fragment\_from\_a = vector\_a[vector\_split\_len \* (proc\_rank - 1):]  
 fragment\_from\_b = vector\_b[vector\_split\_len \* (proc\_rank - 1):]  
 else:  
 fragment\_from\_a = vector\_a[vector\_split\_len \* (proc\_rank - 1): vector\_split\_len \* proc\_rank]  
 fragment\_from\_b = vector\_b[vector\_split\_len \* (proc\_rank - 1): vector\_split\_len \* proc\_rank]  
  
 # Сборка векторов в один и отправка  
 fragment\_for\_send = numpy.array(list(fragment\_from\_a) + list(fragment\_from\_b))  
 send\_request = send\_message(  
 comm=comm,  
 obj=fragment\_for\_send,  
 dest=proc\_rank,  
 tag=TAG,  
 )  
 if isinstance(send\_request, MPI.Request):  
 send\_request.wait()  
  
 # logger.info('Sent message to {}: {}'.format(proc\_rank, fragment\_for\_send))  
  
 # Получение результатов от процессов и вычисление скалярного произведения  
 result = 0  
 for proc\_rank in range(1, size):  
 buffer = numpy.arange(1)  
 answer = recv(comm=comm, source=proc\_rank, tag=TAG)  
 if isinstance(answer, MPI.Request):  
 answer = answer.wait()  
  
 # logger.info('Recv message from {}: {}'.format(proc\_rank, answer))  
 result += answer[0]  
  
 result\_message = 'Result: {}'.format(result)  
 time\_message = 'Time: {} ms'.format(round((time.time() - start\_time) \* 1000))  
 print(result\_message)  
 # logger.info(result\_message)  
 print(time\_message)  
 # logger.warning(time\_message)  
 else:  
 # Получение данных от 0 процесса  
 answer = recv(comm=comm, source=0, tag=TAG)  
 if isinstance(answer, MPI.Request):  
 answer = answer.wait()  
  
 answer\_middle\_index = len(answer) // 2  
 fragment\_from\_a = answer[:answer\_middle\_index]  
 fragment\_from\_b = answer[answer\_middle\_index:]  
  
 result = 0  
 for coord\_a, coord\_b in zip(fragment\_from\_a, fragment\_from\_b):  
 result += coord\_a \* coord\_b  
  
 number\_for\_send = numpy.array([result])  
 send\_request = send\_message(  
 comm=comm,  
 obj=number\_for\_send,  
 dest=0,  
 tag=TAG,  
 )  
 if isinstance(send\_request, MPI.Request):  
 send\_request.wait()

# **Вывод**

В ходе лабораторной работы 3 было проведено исследование зависимости времени выполнения многопоточной программы от количества потоков и размеров вектором. Исследование было проведено как на блокирующих функциях, так и на их неблокирующих аналогах.