МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

Институт информатики и кибернетики

Кафедра программных систем

Дисциплина «Суперкомпьютеры и их применение»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

по теме

«Изучение влияния параметров коммуникационной среды на производительность распределенных параллельных программ»

Вариант №3

Выполнил:

обучающийся группы 6401-020302D Д.О. Колбанов

Проверил:

доцент кафедры

программных систем,

к.т.н. Р.М. Хабибуллин

Самара 2025

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc194877247)

[2 Результаты вычислительных экспериментов с объяснением полученных зависимостей 4](#_Toc194877248)

[2.1 Зависимость латентности от размера сообщения 4](#_Toc194877249)

[2.2 Зависимость пропускной способности от размера сообщения 5](#_Toc194877250)

[2.3 Сравнение локального копирования и MPI-передачи 6](#_Toc194877251)

[2.4 Широковещательная рассылка данных (broadcast) 7](#_Toc194877252)

[2.5 Рассылка данных процессам (scatter) 8](#_Toc194877253)

[2.6 Сбор данных от процессов (gather) 9](#_Toc194877254)

[ВЫВОДЫ ПО РАБОТЕ 11](#_Toc194877255)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 12](#_Toc194877256)

[ПРИЛОЖЕНИЕ А Код программы 13](#_Toc194877257)

1. Постановка задачи

Цель работы: изучение характеристик коммуникационной среды и их влияния на производительность распределенных программ.

Задание на лабораторную работу:

1. Разработать программу для измерения временных коммуникационной среды: латентности и пропускной способности. Программа должна использовать библиотеку MPI для передачи сообщений по коммуникационной среде. Размер сообщений должен варьироваться от 0 байт до нескольких мегабайт. Для повышения точности измерений программа должна усреднять значения, полученные в результате многочисленных (более 100000 раз) измерений.
2. Построить зависимость времени передачи сообщения между двумя процессами от размера сообщения. Объяснить наблюдаемую зависимость.
3. Произвести измерения, аналогичные пункту 1, для процессов, выполняющихся на одном вычислительном узле кластера. Сравнить время передачи сообщения со временем, необходимым для копирования данных такого же объема между двумя областями памяти.
4. Построить зависимости времени, необходимого для широковещательной рассылки данных (broadcast) от объема рассылаемых данных и количества процессов-получателей.
5. Построить зависимости времени, необходимого для рассылки данных процессам (scatter) от объема рассылаемых данных и количества процессов-получателей.
6. Построить зависимости времени, необходимого для сбора данных от процессов (gather) от объема данных и количества процессов.
7. Составить отчёт по результатам работы.

1. Результаты вычислительных экспериментов с объяснением полученных зависимостей

Для проведения вычислительных экспериментов была написана программа на языке Python, исходный код которой приведён в приложении А.

* 1. Зависимость латентности от размера сообщения

Латентность (или задержка) представляет собой временные затраты, которые возникают при подготовке программного обеспечения и сетевых устройств к передаче данных по каналу связи [1]. На рисунке 1 показан график, демонстрирующий зависимость латентности от объема передаваемого сообщения.

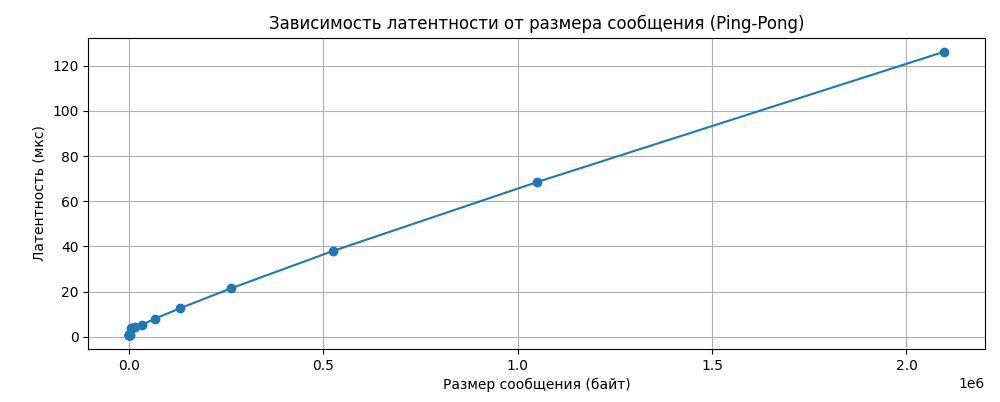


Рисунок 1 – Зависимость латентности от размера сообщения

При небольших объемах данных (до нескольких килобайт) латентность остается сравнительно низкой, составляя единицы или десятки микросекунд. По мере увеличения размера сообщения задержка начинает расти линейно.

Даже если размер сообщения равен нулю, некоторая задержка все равно присутствует. Это связано с выполнением вызовов MPI-функций и процессами синхронизации между узлами. После достижения минимального уровня латентности дальнейшее увеличение времени передачи обусловлено пропускной способностью канала: чем больше данных требуется передать, тем дольше это занимает, особенно если скорость передачи превышает возможности канала. При определенных размерах сообщений (например, 1 МБ или 2 МБ) могут происходить скачкообразные изменения латентности, поскольку MPI может переключаться на альтернативные механизмы передачи данных.

* 1. Зависимость пропускной способности от размера сообщения

Пропускная способность определяется как объем информации, который может быть передан между узлами сети за единицу времени (в байтах в секунду). Однако реальная пропускная способность часто оказывается ниже из-за дополнительных накладных расходов, связанных с обработкой служебной информации на стороне программного обеспечения [1]. График, демонстрирующий зависимость пропускной способности от размера сообщения, представлен на рисунке 2.

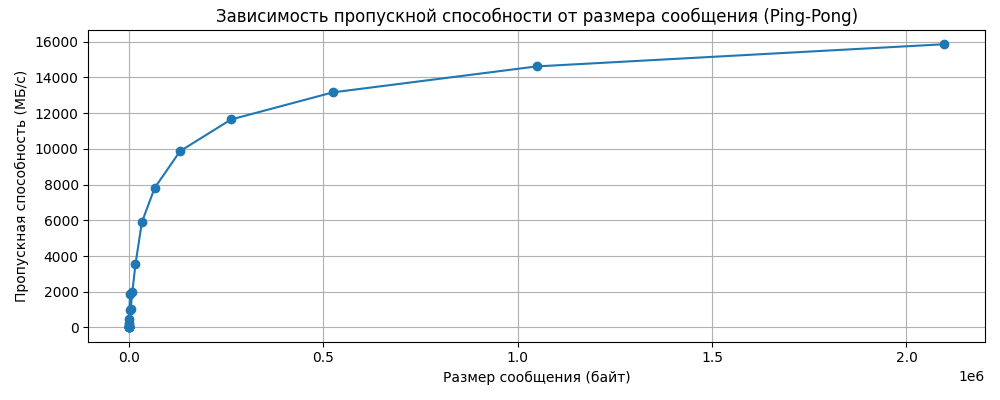


Рисунок 2 – Зависимость пропускной способности от размера сообщения

При передаче очень коротких сообщений значительная часть времени уходит не на саму передачу данных, а на выполнение системных вызовов, операции буферизации и синхронизацию между процессами. Это приводит к снижению эффективности использования канала связи.

Наиболее эффективная работа системы наблюдается при передаче сообщений среднего размера — примерно от сотен килобайт до нескольких мегабайт. В этом диапазоне объем данных уже достаточен для компенсации накладных расходов, а передача осуществляется непрерывным потоком, максимально загружая канал.

При работе с очень большими сообщениями производительность начинает падать. Это связано с ограничениями размеров буферов и кэшей, необходимостью разбиения данных на части, а также повышенной нагрузкой на память системы.

* 1. Сравнение локального копирования и MPI-передачи

На рисунке 3 приведён график зависимости времени передачи от размера сообщения для локального копирования и MPI-передачи.

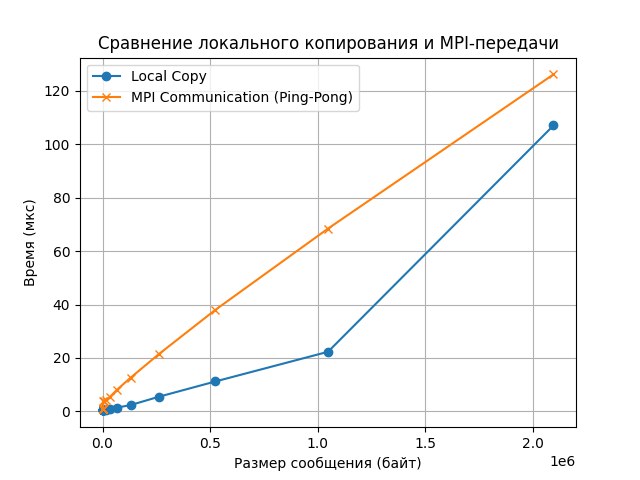


Рисунок 3 – Сравнение локального копирования и MPI-передачи

Локальное копирование работает быстрее, поскольку операции выполняются в пределах одной памяти, минуя необходимость синхронизации между процессами. Задержки в этом случае минимальны и обусловлены только пропускной способностью шины и особенностями работы кэша.

В свою очередь, передача данных через MPI занимает больше времени из-за дополнительных факторов, таких как синхронизация между процессами, буферизация, копирование данных и их обработка внутри MPI-стека.

Для небольших сообщений разница во времени между двумя подходами не так заметна, поскольку основное влияние оказывают накладные расходы, общие для обоих методов.

* 1. Широковещательная рассылка данных (broadcast)

Широковещательная рассылка (broadcast) — это метод передачи данных в компьютерных сетях, при котором поток данных (или каждый пакет в случае пакетной передачи) направляется всем участникам сети [2]. На рисунке 4 показан график зависимости времени выполнения широковещательной рассылки от размера сообщения для 4 процессов, а на рисунке 5 — для 8 процессов.

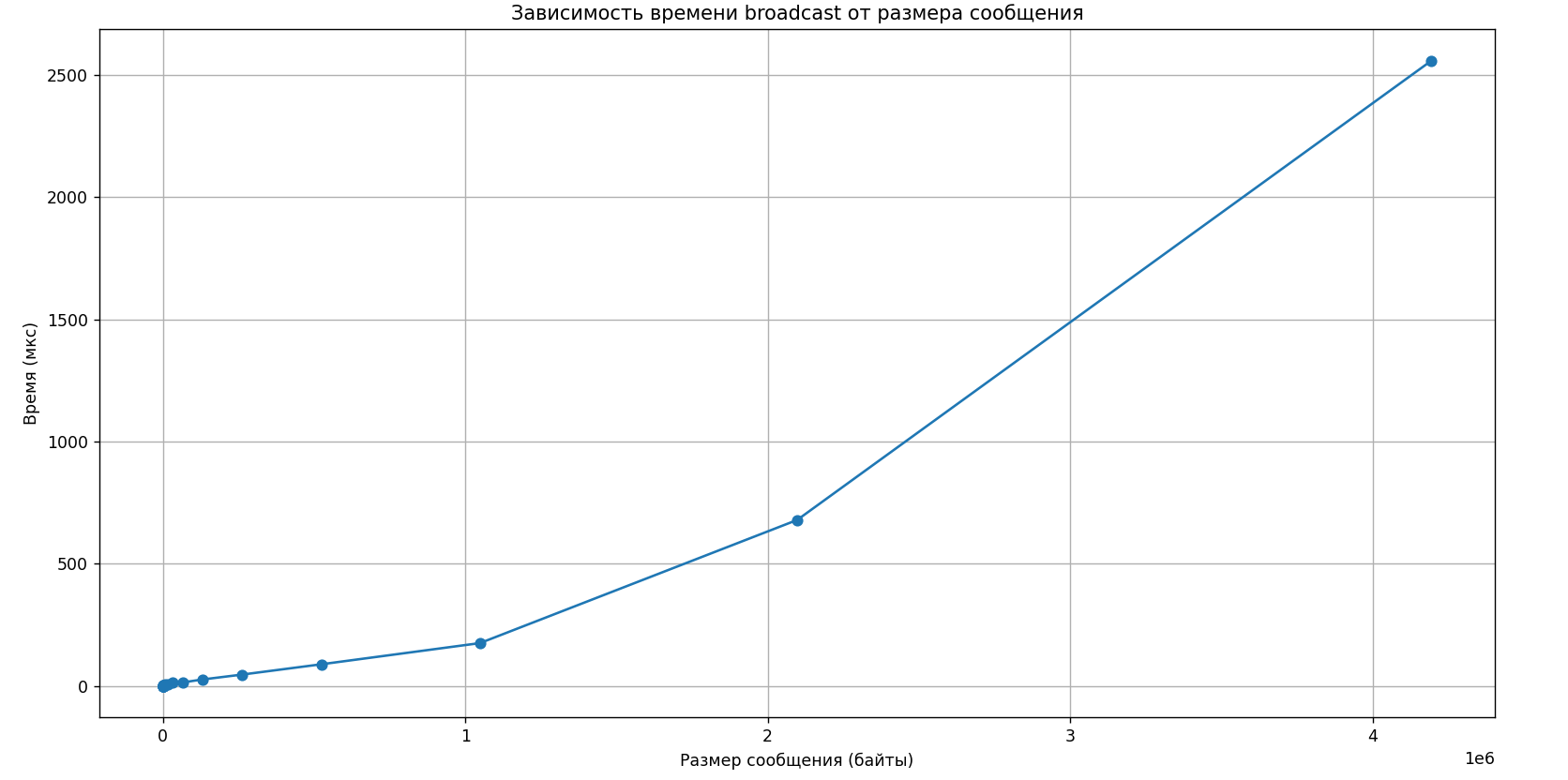


Рисунок 4 – Зависимость времени broadcast от размера сообщения (4 процесса)

При широковещательной рассылке данные отправляются всем процессам одновременно. Увеличение количества процессов приводит к росту числа передаваемых копий и связей между узлами, что, в свою очередь, увеличивает время передачи. Реализация broadcast в MPI может зависеть от структуры дерева передачи. Например, при 8 процессах дерево передачи может иметь больше уровней, чем при 4, что дополнительно замедляет процесс. Также с ростом числа процессов возрастают накладные расходы на синхронизацию, особенно заметные при передаче малых сообщений.

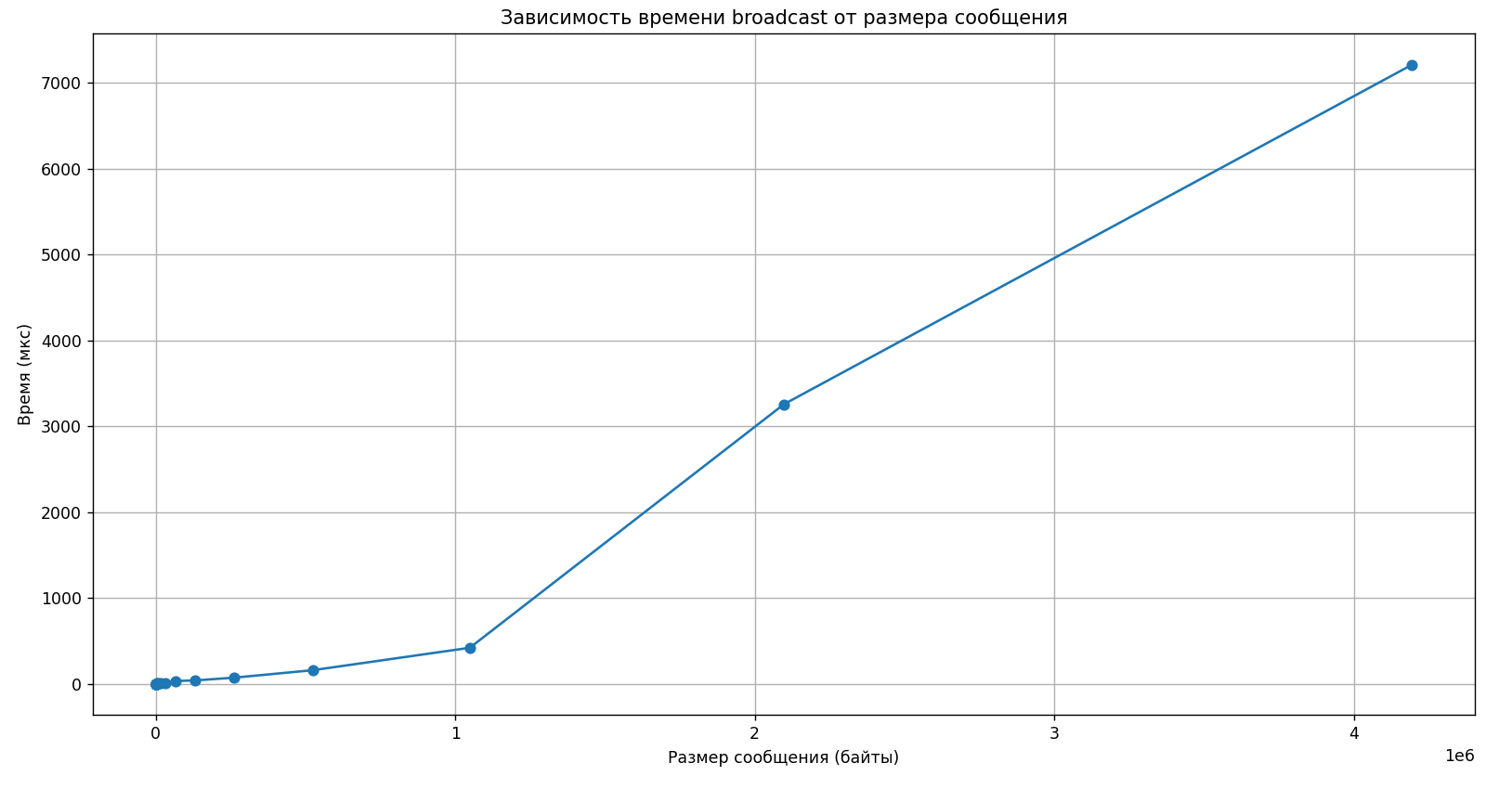


Рисунок 5 – Зависимость времени broadcast от размера сообщения (8 процессов)

* 1. Рассылка данных процессам (scatter)

Функция MPI\_Scatter действует так, как если бы корневой процесс отправлял сообщение с помощью функции MPI\_Send. Это сообщение делится на n равных сегментов, по одному для каждого члена группы. I-й сегмент отправляется в i-й процесс в группе [3]. Графики зависимости времени рассылки данных процессам (scatter) от размера сообщения для 4 и 8 процессов приведены соответственно на рисунках 6 и 7.

В случае 4 процессов – одна и та же порция данных делится на 4. В случае 8 процессов – каждый получает меньше, но число отправок увеличивается в 2 раза. При большом размере сообщения это приводит к накладным расходам на множество отправок и увеличение количества шагов дерева рассылки. При больших размерах сообщения и большом числе процессов растут и локальные расходы на разделение буфера, и глобальные – на передачу.

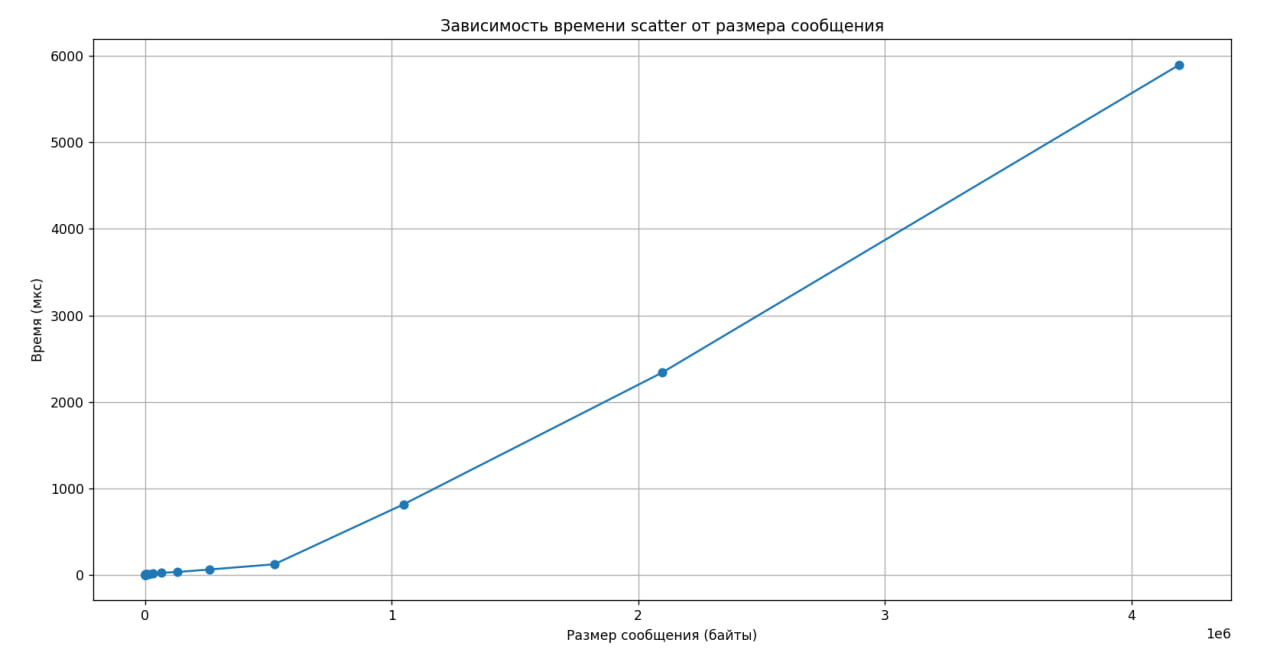


Рисунок 6 – Зависимость времени scatter от размера сообщения (4 процесса)

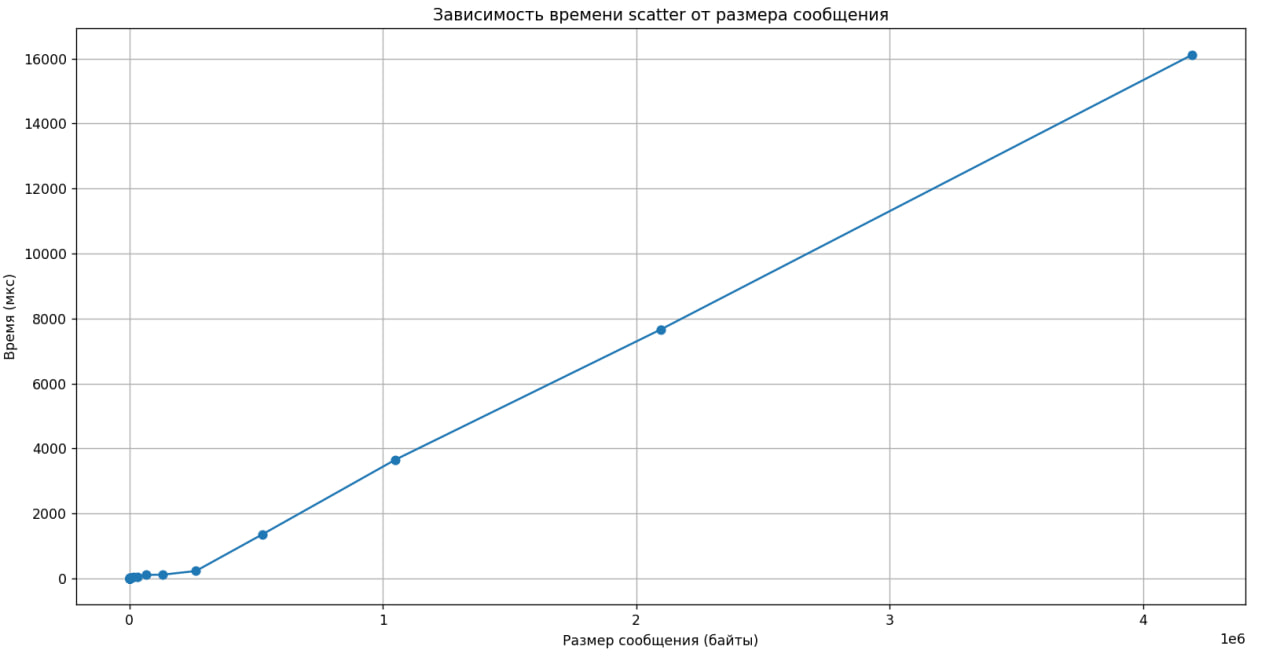


Рисунок 7 – Зависимость времени scatter от размера сообщения (8 процессов)

* 1. Сбор данных от процессов (gather)

MPI\_Gather собирает данные от всех участников группы к одному участнику [4]. Графики зависимости, необходимого для сбора данных от процессов (gather) от объёма данных для 4 и 8 процессов приведены на рисунках 8 и 9 соответственно.

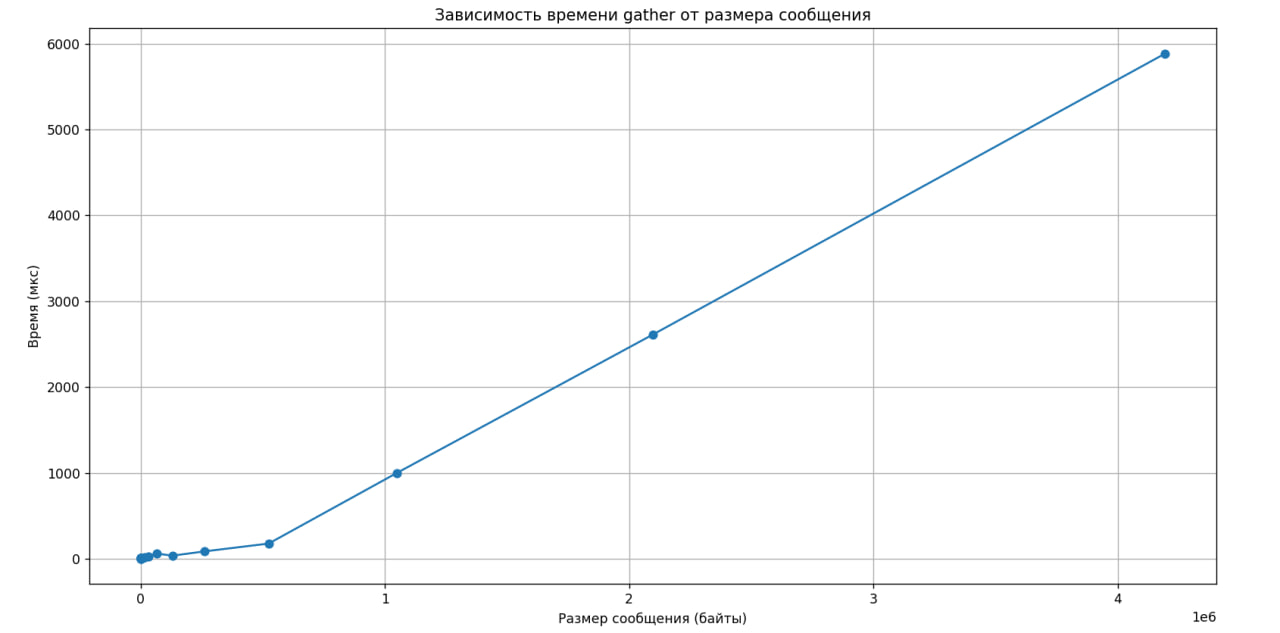


Рисунок 8 – Зависимость времени gather от размера сообщения (4 процесса)

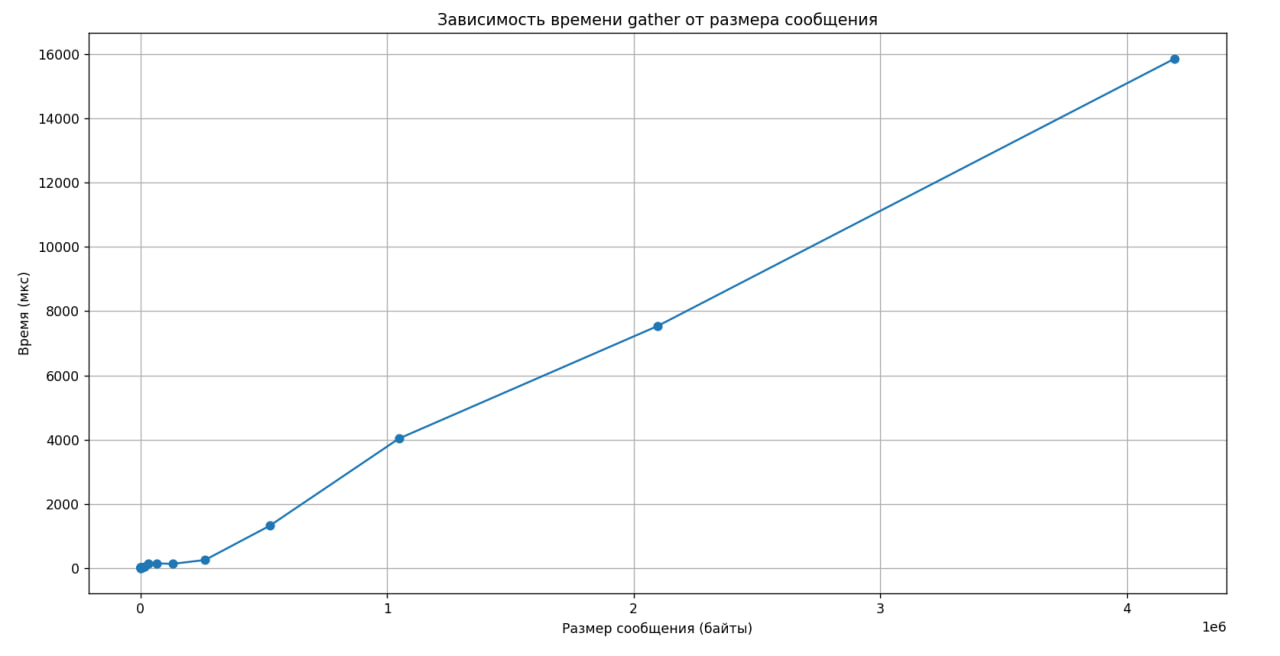


Рисунок 9 – Зависимость времени gather от размера сообщения (8 процессов)

При 4 процессах корневой процесс собирает данные от 3 других, при 8 – уже от 7 и каждый из них может передавать такой же по размеру кусок, поэтому общий объём данных на входе увеличивается почти в 2 раза, что ведёт к увеличению времени. Также корневой процесс может не успевать обрабатывать большое количество данных. Это особенно критично при увеличении числа процессов, так как все потоки данных сходятся в одну точку.

ВЫВОДЫ ПО РАБОТЕ

В рамках выполнения лабораторной работы была разработана программа на языке Python с использованием библиотеки MPI. Целью работы было исследование влияния параметров коммуникационной среды на производительность распределённых параллельных программ. В ходе вычислительных экспериментов, проведённых для различных параметров задачи, были сделаны следующие выводы:

1. При малых объёмах сообщений время передачи практически не зависит от размера данных. Наиболее эффективное использование канала связи наблюдается при средних размерах сообщений. При больших объёмах данных производительность начинает снижаться из-за ограничений, связанных с буферизацией и кэшированием.
2. Локальное копирование демонстрирует более высокую скорость по сравнению с MPI-передачей на всех диапазонах размеров сообщений. Это объясняется тем, что взаимодействие через MPI влечёт за собой дополнительные накладные расходы, такие как синхронизация, буферизация и обработка данных в стеке MPI.
3. Увеличение числа процессов приводит к росту нагрузки на корневой процесс, что оказывает негативное влияние на общее время передачи данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Латентность и пропускная способность [Электронный ресурс]. URL: <https://parallel.ru/testmpi/transfer.html> (дата обращения: 06.04.2025).
2. Broadcast [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/broadcast](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D0%B5%D1%89%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%B0%D0%BD%D0%B0%D0%BB) (дата обращения: 06.04.2025).
3. Scatter [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/message-passing-interface/mpi-scatter-function> (дата обращения: 06.04.2025).
4. Gather [Электронный ресурс. URL: https: <https://learn.microsoft.com/ru-ru/message-passing-interface/mpi-gather-function> (дата обращения: 06.04.2025).

ПРИЛОЖЕНИЕ А  
Код программы

from mpi4py import MPI

import numpy as np

import matplotlib.pyplot as plt

import sys

import time

# Функция для измерения "ping-pong" обмена между двумя процессами.

def ping\_pong\_test(comm, iterations, message\_size):

rank = comm.Get\_rank()

# Партнер: если 0, то 1; если 1, то 0 (тест выполняется только между двумя процессами)

partner = 1 if rank == 0 else 0

# Создаем сообщение заданного размера (массив байт)

data = np.empty(message\_size, dtype=np.byte)

# "Разогрев": несколько обменов без замеров

for \_ in range(10):

if rank == 0:

comm.Send([data, MPI.BYTE], dest=partner, tag=100)

comm.Recv([data, MPI.BYTE], source=partner, tag=101)

else:

comm.Recv([data, MPI.BYTE], source=partner, tag=100)

comm.Send([data, MPI.BYTE], dest=partner, tag=101)

comm.Barrier()

start = time.time()

for \_ in range(iterations):

if rank == 0:

comm.Send([data, MPI.BYTE], dest=partner, tag=100)

comm.Recv([data, MPI.BYTE], source=partner, tag=101)

else:

comm.Recv([data, MPI.BYTE], source=partner, tag=100)

comm.Send([data, MPI.BYTE], dest=partner, tag=101)

end = time.time()

# Возвращаем общее время обмена (round-trip)

return end - start

# Функция для измерения времени broadcast операции.

def broadcast\_test(comm, iterations, message\_size):

data = np.empty(message\_size, dtype=np.byte)

# Разогрев

for \_ in range(10):

comm.Bcast(data, root=0)

comm.Barrier()

start = time.time()

for \_ in range(iterations):

comm.Bcast(data, root=0)

end = time.time()

return end - start

# Функция для измерения времени scatter операции.

def scatter\_test(comm, iterations, message\_size):

rank = comm.Get\_rank()

size = comm.Get\_size()

# На корневом процессе формируем общий буфер, разделённый на части

sendbuf = np.empty(message\_size \* size,

dtype=np.byte) if rank == 0 else None

recvbuf = np.empty(message\_size, dtype=np.byte)

# Разогрев

for \_ in range(10):

comm.Scatter(sendbuf, recvbuf, root=0)

comm.Barrier()

start = time.time()

for \_ in range(iterations):

comm.Scatter(sendbuf, recvbuf, root=0)

end = time.time()

return end - start

# Функция для измерения времени gather операции.

def gather\_test(comm, iterations, message\_size):

rank = comm.Get\_rank()

size = comm.Get\_size()

sendbuf = np.empty(message\_size, dtype=np.byte)

recvbuf = np.empty(message\_size \* size,

dtype=np.byte) if rank == 0 else None

# Разогрев

for \_ in range(10):

comm.Gather(sendbuf, recvbuf, root=0)

comm.Barrier()

start = time.time()

for \_ in range(iterations):

comm.Gather(sendbuf, recvbuf, root=0)

end = time.time()

return end - start

# Функция для измерения времени локального копирования данных

def local\_copy\_test(iterations, message\_size):

data = np.empty(message\_size, dtype=np.byte)

target = np.empty(message\_size, dtype=np.byte)

# Разогрев

for \_ in range(10):

np.copyto(target, data)

start = time.time()

for \_ in range(iterations):

np.copyto(target, data)

end = time.time()

return end - start

def main():

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

# Для ping-pong теста требуется как минимум 2 процесса

if size < 2:

if rank == 0:

print("Для выполнения теста необходимо минимум 2 процесса.")

MPI.Finalize()

sys.exit(1)

# Задаем набор размеров сообщений (в байтах)

message\_sizes = [0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024,

2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,

524288, 1048576, 2097152]

ITER\_SMALL = 100000

ITER\_LARGE = 10000

# для хранения (размер, средняя латентность, пропускная способность)

pingpong\_results = []

copy\_results = [] # для локального копирования

# Выполняем тест "ping-pong" между процессами 0 и 1

if rank in [0, 1]:

for msize in message\_sizes:

iter\_count = ITER\_SMALL if msize <= 1024 else ITER\_LARGE

total\_time = ping\_pong\_test(comm, iter\_count, msize)

# Среднее время одного обмена (одна сторона) = (round-trip time) / (2 \* итераций)

avg\_time = total\_time / (2 \* iter\_count)

# Пропускная способность: размер/время (в байтах/сек), переводим в МБ/с

bandwidth = (msize / avg\_time) / \

(1024 \* 1024) if avg\_time > 0 else 0

if rank == 0:

print(f"[Ping-Pong] Размер: {msize:7d} байт, Латентность: {avg\_time\*1e6:7.2f} мкс, "

f"Пропускная способность: {bandwidth:6.2f} МБ/с")

pingpong\_results.append((msize, avg\_time, bandwidth))

# Тест локального копирования (только на процессе 0)

if rank == 0:

for msize in message\_sizes:

iter\_count = ITER\_SMALL if msize <= 1024 else ITER\_LARGE

t\_copy = local\_copy\_test(iter\_count, msize)

avg\_copy\_time = t\_copy / iter\_count

print(

f"[Local Copy] Размер: {msize:7d} байт, Время копирования: {avg\_copy\_time\*1e6:7.2f} мкс")

copy\_results.append((msize, avg\_copy\_time))

# Измерение collective операций (broadcast, scatter, gather)

ITER\_BCAST = 1000

ITER\_COLL = 1000

bcast\_results = []

for msize in message\_sizes:

iter\_count = ITER\_BCAST if msize > 1024 else 10000

t = broadcast\_test(comm, iter\_count, msize)

avg\_time = t / iter\_count

if rank == 0:

print(

f"[Broadcast] Размер: {msize:7d} байт, Время: {avg\_time\*1e6:7.2f} мкс")

bcast\_results.append((msize, avg\_time))

scatter\_results = []

for msize in message\_sizes:

iter\_count = ITER\_COLL if msize > 1024 else 10000

t = scatter\_test(comm, iter\_count, msize)

avg\_time = t / iter\_count

if rank == 0:

print(

f"[Scatter] Размер: {msize:7d} байт, Время: {avg\_time\*1e6:7.2f} мкс")

scatter\_results.append((msize, avg\_time))

gather\_results = []

for msize in message\_sizes:

iter\_count = ITER\_COLL if msize > 1024 else 10000

t = gather\_test(comm, iter\_count, msize)

avg\_time = t / iter\_count

if rank == 0:

print(

f"[Gather] Размер: {msize:7d} байт, Время: {avg\_time\*1e6:7.2f} мкс")

gather\_results.append((msize, avg\_time))

# Построение графиков (выполняется только процессом 0)

if rank == 0:

# Графики для ping-pong теста: латентность и пропускная способность

sizes = [r[0] for r in pingpong\_results]

# перевод в микросекунды

latencies = [r[1]\*1e6 for r in pingpong\_results]

bandwidths = [r[2] for r in pingpong\_results]

plt.figure(figsize=(10, 8))

plt.subplot(2, 1, 1)

plt.plot(sizes, latencies, marker='o')

plt.xlabel("Размер сообщения (байт)")

plt.ylabel("Латентность (мкс)")

plt.title("Зависимость латентности от размера сообщения (Ping-Pong)")

plt.grid(True)

plt.subplot(2, 1, 2)

plt.plot(sizes, bandwidths, marker='o')

plt.xlabel("Размер сообщения (байт)")

plt.ylabel("Пропускная способность (МБ/с)")

plt.title(

"Зависимость пропускной способности от размера сообщения (Ping-Pong)")

plt.grid(True)

plt.tight\_layout()

plt.savefig("pingpong\_results.png")

plt.show()

# График для сравнения локального копирования и MPI-передачи

copy\_sizes = [r[0] for r in copy\_results]

copy\_times = [r[1]\*1e6 for r in copy\_results]

# латентность как время передачи

mpi\_times = [r[1]\*1e6 for r in pingpong\_results]

plt.figure()

plt.plot(copy\_sizes, copy\_times, marker='o', label="Local Copy")

plt.plot(copy\_sizes, mpi\_times, marker='x',

label="MPI Communication (Ping-Pong)")

plt.xlabel("Размер сообщения (байт)")

plt.ylabel("Время (мкс)")

plt.title("Сравнение локального копирования и MPI-передачи")

plt.legend()

plt.grid(True)

plt.savefig("copy\_vs\_mpi.png")

plt.show()

# Графики для collective операций: Broadcast, Scatter, Gather

def plot\_coll(results, op\_name, filename):

sizes = [r[0] for r in results]

times = [r[1]\*1e6 for r in results]

plt.figure()

plt.plot(sizes, times, marker='o')

plt.xlabel("Размер сообщения (байт)")

plt.ylabel("Время (мкс)")

plt.title(f"{op\_name}: Зависимость времени от размера сообщения")

plt.grid(True)

plt.savefig(filename)

plt.show()

plot\_coll(bcast\_results, "Broadcast", "broadcast\_results.png")

plot\_coll(scatter\_results, "Scatter", "scatter\_results.png")

plot\_coll(gather\_results, "Gather", "gather\_results.png")

if \_\_name\_\_ == '\_\_main\_\_':

main()