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 №8**

**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**

# Разработать алгоритм определения того, является ли граф гиперкубом. Входные данные – матрица смежности.

# **Ход работы**

# В ходе работы был реализован параллельный алгоритм с использованием интерфейса MPI на языке Python.

# Параллельный алгоритм реализован с использованием методов коллективного для обмена (Scatter, Bcast, Reduce). Использование метода Gather в рамках данной задачи не является эффективным.

# Управляющий процесс в параллельном алгоритме в начале работы проверяет, является ли количество вершин в графе степенью 2. Если нет, то работы программы прекращается. Иначе, управляющий процесс отправляет каждому подчиненному процессу размерность предполагаемого гиперкуба, разбивает матрицу смежности на равные, если возможно, части и отправляет всем остальным процессам. Если матрицу смежности не удается разбить на равные части, то остаток строк будет отправлен последнему подчиненному процессу.

# Каждый подчиненный процесс получает размерность предполагаемого гиперкуба и фрагмент матрицы смежности. После чего он проверяет, что все вершины имеют соответствующую степень (размерность гиперкуба). Если это так, то в управляющий процесс с помощью метода Reduce направляется логическое значение True, иначе False. Управляющий процесс выполняет собирает результаты каждого процесса и с помощью метода Reduce выполняет конъюнкцию.

# Полученный результат определяет, является ли граф гиперкубом.

# Анализ эффективности алгоритма не выполнялся в связи со сложностью создания необходимых тестовых данных.

Код программы:

import os  
import logging  
import random  
import time  
from math import log  
  
from mpi4py import MPI  
  
# Run command  
# mpiexec -np 2 py lab\_8.py  
  
RANDOM\_MODE = False  
TAG = 2  
GRAPH = [  
 [0, 1, 0, 1],  
 [1, 0, 1, 0],  
 [0, 1, 0, 1],  
 [1, 0, 1, 0],  
]  
  
GRAPH\_SIZE = 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 init\_test\_data():  
 def generate\_matrix(matrix\_size):  
 return [[random.randint(0, 1) for \_ in range(matrix\_size)] for \_ in range(matrix\_size)]  
  
 def get\_random\_graph(graph\_size):  
 graph = generate\_matrix(graph\_size)  
 for i in range(graph\_size):  
 for j in range(graph\_size):  
 if i == j:  
 graph[i][j] = 0  
 continue  
  
 graph[i][j] = graph[j][i]  
  
 return graph  
  
 def get\_matrix\_width(matrix):  
 lines\_width = {len(line) for line in matrix}  
 if len(lines\_width) > 1:  
 raise Exception('Incorrect matrix!')  
  
 return list(lines\_width)[0]  
  
 if RANDOM\_MODE:  
 return get\_random\_graph(GRAPH\_SIZE)  
  
 matrix = GRAPH  
  
 width = get\_matrix\_width(matrix)  
 if width != len(matrix):  
 raise Exception('Not correct matrix!')  
  
 if size - 1 > width:  
 raise Exception('Too many processes!')  
  
 return matrix  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 if rank == 0:  
 graph = init\_test\_data()  
 start\_time = time.time()  
 # Вычисляем размерность гиберкуба  
 hibercube\_rank = log(len(graph), 2)  
 hibercube\_rank = int(hibercube\_rank) if hibercube\_rank.is\_integer() else None  
 # Если получить не удалось (вершин в графе не 2^N), то работа программы прекращается сразу  
 print(f'hibercube\_rank: {hibercube\_rank}')  
 # Передаем каждому процессу размерность гиберкуба или сигнал прекращения работы  
 comm.bcast(  
 obj=hibercube\_rank,  
 root=0,  
 )  
 result = False  
 if hibercube\_rank is not None:  
 matrix\_split\_len = len(graph) // (size - 1)  
 matrix\_for\_send = []  
 for proc\_rank in range(1, size):  
 if proc\_rank == size - 1:  
 matrix\_for\_send.append(graph[matrix\_split\_len \* (proc\_rank - 1):])  
 else:  
 matrix\_for\_send.append(graph[matrix\_split\_len \* (proc\_rank - 1): matrix\_split\_len \* proc\_rank])  
  
 scatter\_matrix = [None, \*matrix\_for\_send]  
 comm.scatter(  
 sendobj=scatter\_matrix,  
 root=0,  
 )  
  
 # Получение результатов от процессов  
 # LAND - Logical AND  
 result = comm.reduce(  
 sendobj=True,  
 root=0,  
 op=MPI.LAND,  
 )  
  
 result\_message = f'Result: {result}'  
 time\_message = 'Time: {} ms'.format(round((time.time() - start\_time) \* 1000))  
 print(result\_message)  
 print(time\_message)  
 else:  
 # Получение размерности гиперкуба от 0 процесса  
 # Если будет получен None, граф не может быть гиперкубом и работа программы на этом заканчивается  
 hibercube\_rank = comm.bcast(  
 obj=None,  
 root=0,  
 )  
  
 if hibercube\_rank is not None:  
 graph\_fragment = comm.scatter(  
 sendobj=None,  
 root=0,  
 )  
 if not RANDOM\_MODE:  
 print(f'process {rank} get: {graph\_fragment}')  
  
 # Проверка количества ребер для каждой вершины  
 answer = True  
 for line in graph\_fragment:  
 logger.info(line)  
 if sum(line) != hibercube\_rank:  
 answer = False  
 logger.info('break')  
 break  
  
 comm.reduce(  
 sendobj=answer,  
 root=0,  
 op=MPI.LAND,  
 )  
 if not RANDOM\_MODE:  
 print(f'process {rank} send: {answer}')

# **Вывод**

В ходе лабораторной работы 8 были исследованы методы коллективного обмена в MPI, а также реализован параллельный алгоритм, определяющий, является ли граф, гиперкубом.