МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные системы»

Тема: «Численные методы»

Студентка гр. 1307 _____ Грунская Н.Д.

Преподаватель _____ Манжиков Л.П.

Санкт-Петербург

2025

Цель: приобрести навыки в распараллеливании программы.

Задание 1 Представить последовательный и параллельный вариант программы, реализующей:

Вариант 3

Метод приведения разреженной матрицы к блочной диагональной форме;

Команды:

1) int MPI_Bcast(void *buf, int count, MPI_Datatype type, int root, MPI_Comm comm) – рассылка от одного всем целиком;
2) int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm) – сбор данных со всех к одному и применение к данным заданной операции;
3) int MPI_Scatter(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm) - – рассылка от одного всем частями;
4) int MPI_Gather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm) - сбор частей от всех к одному в целое
5) int MPI_Alltoall(void *sbuf,int scount,MPI_Datatype stype, void *rbuf,int rcount,MPI_Datatype rtype,MPI_Comm comm) – передача от всех всем.

Описание последовательного алгоритма

Основная идея

Алгоритм преобразует матрицу смежности графа к блочно-диагональному виду путем перестановки строк и столбцов. Сначала находится компоненты связности графа с помощью DFS. Затем вершины сортируются по принадлежности к компонентам связности. Наконец, матрица перестраивается в соответствии с полученным порядком вершин, образуя блочно-диагональную форму.

Шаги алгоритма

1. **Ввод данных:**
    - Пользователь вводит размер квадратной матрицы N.
    - Пользователь построчно вводит элементы матрицы, представляющие собой матрицу смежности графа.
2. **Построение списка смежности:**
    - Создается список смежности adjacencyList, представляющий граф. Для каждой вершины i в списке adjacencyList[i] хранятся номера вершин, с которыми iсвязана (есть ненулевой элемент в матрице смежности).
    - Происходит итерация по верхней треугольной части матрицы (чтобы избежать дублирования ребер) и, если элементы matrix[i][j] или matrix[j][i]ненулевые, в список adjacencyList добавляются соответствующие ребра.
3. **Поиск компонент связности (DFS):**
    - Создаются два вектора: visited (логический вектор, отмечающий посещенные вершины) и component (вектор, хранящий идентификатор компоненты связности для каждой вершины).
    - Алгоритм обходит граф с помощью поиска в глубину (DFS), чтобы определить компоненты связности.

- Начиная с каждой непосещенной вершины i, вызывается рекурсивная функция dfs:
  - Функция dfs помечает вершину v как посещенную (visited[v] = true) и присваивает ей идентификатор текущей компоненты compId (component[v] = compId).
  - Для каждой смежной вершины w вершины v (т.е., вершины, соединенной с v), если w еще не посещена, рекурсивно вызывается dfs для w с тем же идентификатором компоненты compId.
- После завершения обхода каждой компоненты связности, идентификатор compId инкрементируется, чтобы начать поиск новой компоненты.

4. **Определение порядка вершин:**
   - Создается вектор permutation, содержащий исходный порядок вершин (от 0 до N-1).
   - Вектор permutation сортируется на основе значений в векторе component. Вершины сортируются так, чтобы вершины, принадлежащие одной и той же компоненте связности, располагались рядом друг с другом. Используется лямбда-функция для сравнения вершин на основе их принадлежности к компонентам связности.
5. **Преобразование матрицы к блочно-диагональной форме:**
   - Создается новая матрица outputMatrix того же размера, что и исходная матрица.
   - Элементы outputMatrix заполняются на основе исходной матрицы, но с использованием перестановки вершин, хранящейся в векторе permutation. То есть, строка i и столбец j матрицы outputMatrix содержат элемент, который находился в строке permutation[i] и столбце permutation[j] исходной матрицы matrix.

6. **Вывод результатов:**
    - Выводится исходная матрица.
    - Выводится информация о компонентах связности для каждой вершины.
    - Выводится порядок вершин, определенный для получения блочно-диагональной формы.
    - Выводится преобразованная матрица в блочно-диагональной форме.
    - Выводится время выполнения алгоритма.

Сложность алгоритма:

- **Построение списка смежности:** O(N^2), где N - размер матрицы.
- **Поиск компонент связности (DFS):** O(N + E), где N - количество вершин, E - количество ребер. В худшем случае E может быть O(N^2), поэтому сложность O(N^2).
- **Сортировка вершин:** O(N log N).
- **Преобразование матрицы:** O(N^2).

Итоговая сложность: O(N^2) , поскольку это самая высокая сложность среди всех этапов.

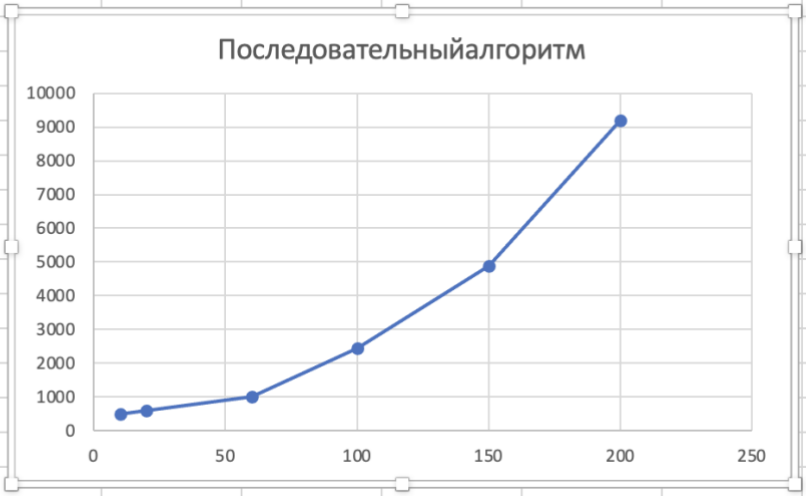
Рисунок. 1 Схема последовательного алгоритма

Рисунок. 2 Схема параллельного алгоритма

Результаты выполнения:

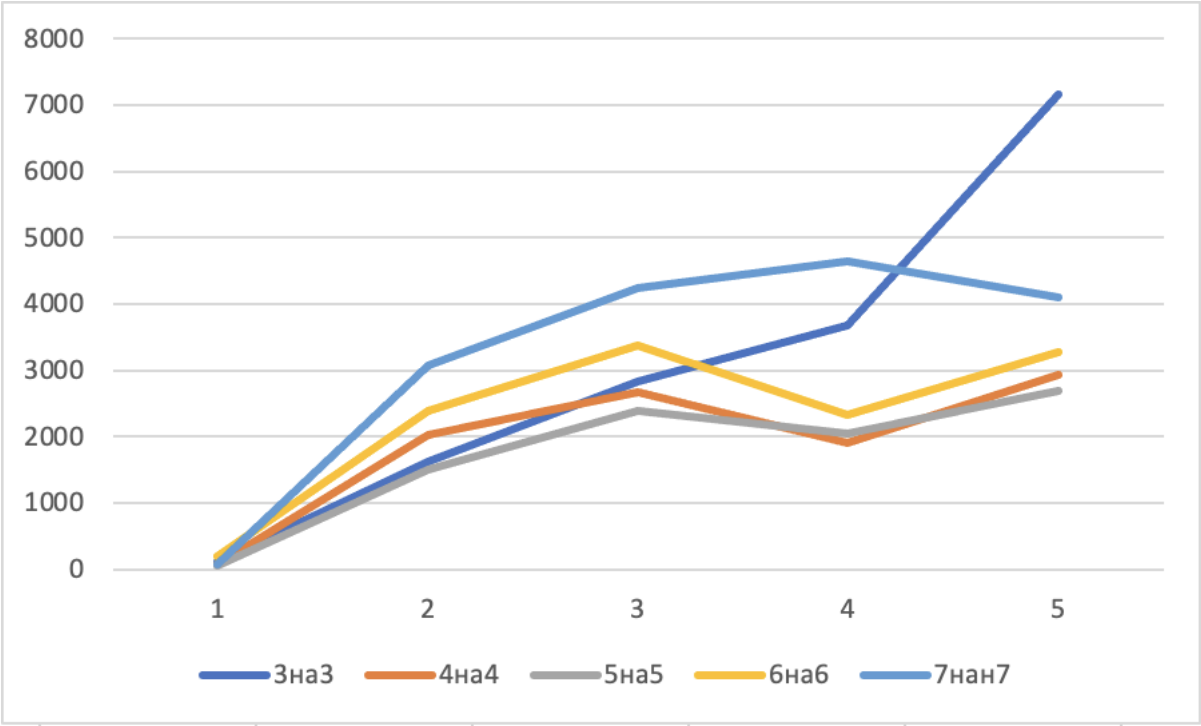
Последовательный алгоритм:

3	31
4	62
5	63
6	124
7	133
...	
10	488
20	586
60	1011
100	2453
150	4870
200	9185



Параллельный алгоритм

	Потоки				
Размер матрицы	1	2	3	4	5
3	104	1622	2838	3676	7156
4	64	2034	2660	1897	2933
5	58	1502	2379	2037	2684
6	192	2387	3381	2335	3282
7	68	3075	4246	4642	4105



На обычных для нас размерах матрицы мы видим, что время только увеличивается с увеличением потоков – распараллеливание съедает очень много ресурсов.

	Потоки				
Размер матрицы	1	2	3	4	5
10	2616	2066	2414	2244	5004
20	276	2224	-	-	-
60	426	450	-	-	-
100	736	693	660	-	-
150	1463	1239	1035	1025	
200	2496	2368	2225	2199	

Далее видим, что при больших размерах матриц (более 100 строк) параллельный алгоритм начинает улучшать свои показатели и становится лучше послежовательного. При последующем увеличении размера будет виден прирост по скорости при увеличении кол-ва потоков.

Доказательство оптимальности параллельного алгоритма:

Сложность последовательного алгоритма: *O(N^2)*.

Сложность параллельного алгоритма: *O(N^2)/P + накладные на коммуникацию между потоками*.
P – кол-во потоков, из-за расходов на коммуникацию работает быстрее при большем *N*.

Вывод:

В ходе лабораторной работы были реализованы последовательный и параллельный методы приведения разреженной матрицы к блочной диагональной форме.