**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.

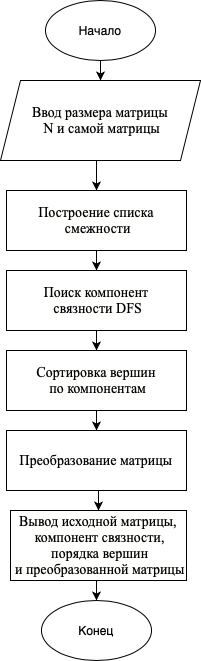


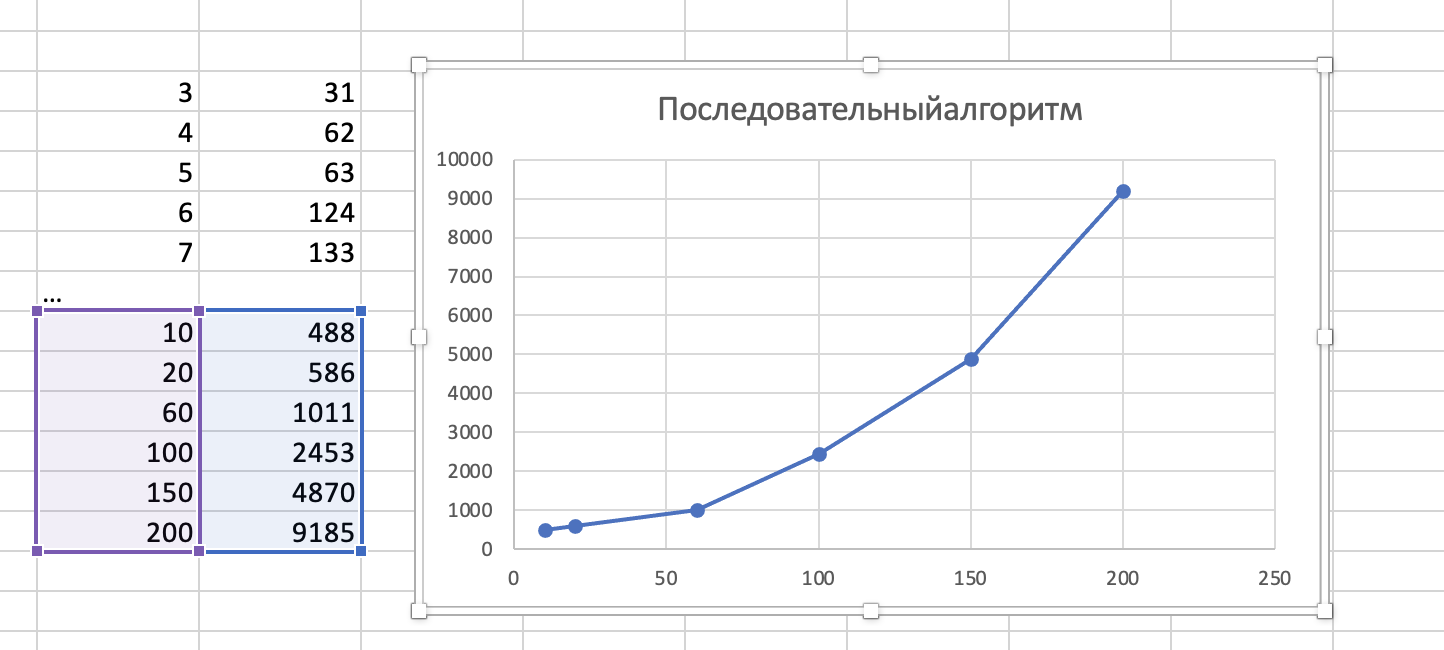
Рисунок. 1 Схема последовательного алгоритма



Рисунок. 2 Схема параллельного алгоритма

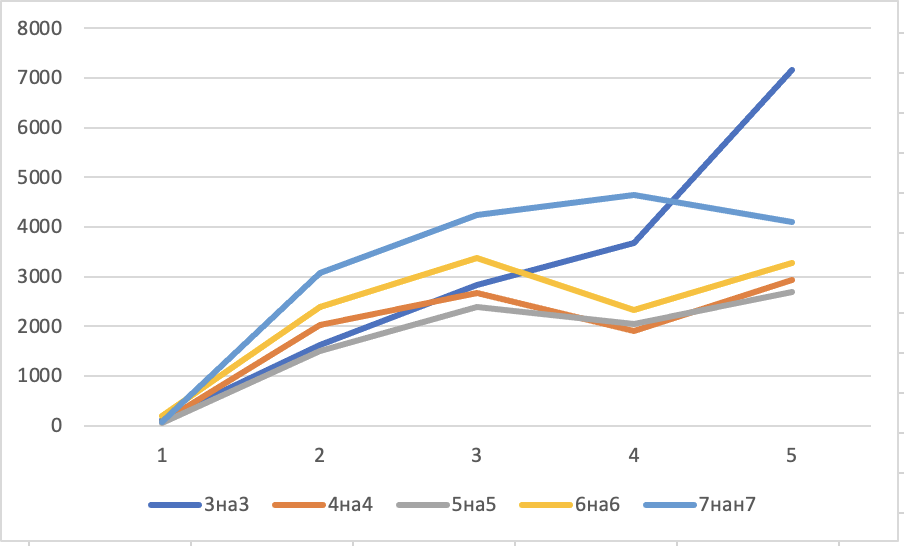
**Результаты выполнения:**

**Последовательный алгоритм:**

****

**Параллельный алгоритм**

****



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

****

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:**

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

Cложность 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.