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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Параллельные вычисления»

Тема: «Численные методы»

Студент гр. 1307 _____ Голубев М.А.

Преподаватель _____ Манжиков Л.П.

Санкт-Петербург
2025

Цель работы.

Приобрести навыки в распараллеливании программы.

Задание 1 (по вариантам).

Задание 1 (по вариантам). Представить последовательный и параллельный вариант программы, реализующей:

8) Метод нахождения поздних сроков выполнения операций алгоритма;

Описание последовательного алгоритма:

1. **Инициализация данных**:
    - Задаются параметры: количество операций (OPERATIONS) и плотность графа (DENSITY).
    - Инициализируются массивы для хранения графа, длительностей операций и флагов финальных операций.
2. **Генерация случайного ацикличного графа**:
    - Для каждой пары операций (i, j), где i < j, с вероятностью DENSITY создаётся ребро с случайным весом.
    - Для каждой операции задаётся случайная длительность.
    - Определяются финальные операции (те, у которых нет исходящих рёбер).
3. **Вычисление ранних сроков завершения (EFT)**:
    - Для каждой операции вычисляется максимальное время завершения всех её предшественников.
    - Ранний срок завершения операции равен сумме максимального времени предшественников и её длительности.
    - Процесс повторяется, пока все ранние сроки не стабилизируются.
4. **Вычисление поздних сроков завершения (LFT)**:
    - Для каждой операции (начиная с последней) вычисляется минимальное время начала всех её последователей.
    - Поздний срок завершения операции равен минимальному времени начала последователей минус длительность операции.

- Процесс повторяется, пока все поздние сроки не стабилизируются.

5. **Вычисление поздних сроков начала (LST)**:
   - Для каждой операции поздний срок начала равен разности её позднего срока завершения и длительности.
6. **Вывод результатов**:
   - Выводятся поздние сроки начала для всех операций.
   - Выводится время выполнения алгоритма.

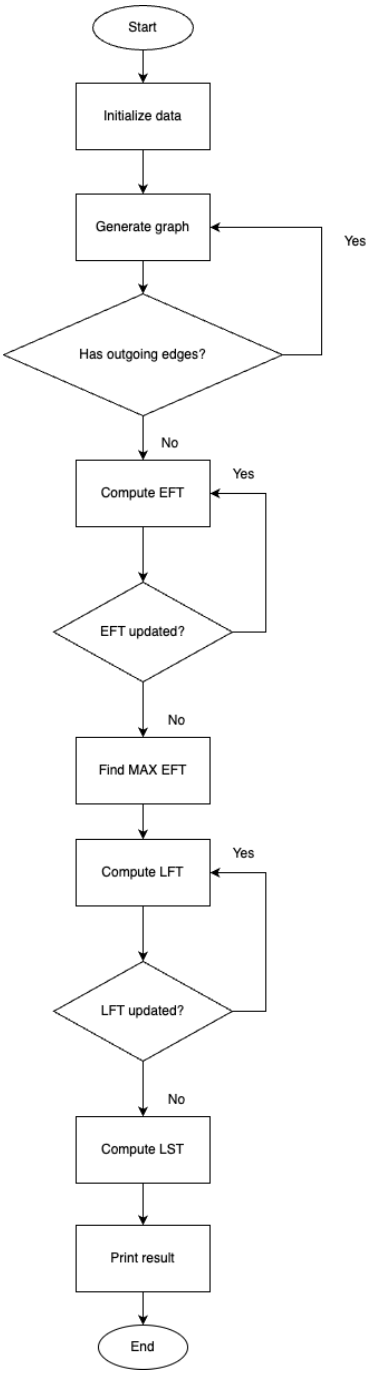

Рисунок 1. Блок-схема последовательного алгоритма

Описание распараллеленного алгоритма:

1. **Инициализация MPI**:
    - Запуск MPI и получение ранга процесса и общего числа процессов.
2. **Генерация графа** (только на процессе с рангом 0):
    - Создается случайный ацикличный граф с заданным количеством задач (numTasks) и плотностью (density).
    - Заполняются массивы:
        - taskTimes — время выполнения каждой задачи.
        - isTerminal — флаги, указывающие, является ли задача финальной (без исходящих ребер).
        - graphData — матрица смежности графа в виде одномерного массива.
3. **Распространение данных**:
    - Массивы taskTimes и isTerminal рассылаются всем процессам с помощью MPI_Bcast.
    - Граф (graphData) распределяется между процессами с помощью MPI_Scatterv.
4. **Вычисление ранних сроков завершения (EFT)**:
    - Каждый процесс вычисляет EFT для своих задач на основе зависимостей в графе.
    - Процессы обмениваются данными через MPI_Allgatherv для обновления глобальных значений EFT.
    - Процесс повторяется до тех пор, пока все EFT не стабилизируются.
5. **Инициализация поздних сроков завершения (LFT)**:
    - Для финальных задач LFT устанавливается равным максимальному значению EFT.
6. **Вычисление поздних сроков завершения (LFT)**:
    - Каждый процесс вычисляет LFT для своих задач, двигаясь от потомков к предкам.

- Процессы обмениваются данными через MPI_Allgatherv для обновления глобальных значений LFT.
- Процесс повторяется до тех пор, пока все LFT не стабилизируются.

7. **Сборка результатов**:
   - Все процессы собирают финальные значения LFT с помощью MPI_Allgatherv.
8. **Вычисление поздних сроков начала (LST)**:
   - Для каждой задачи LST вычисляется как LST = LFT - taskTime.
9. **Вывод результатов** (только на процессе с рангом 0):
   - Выводятся значения LST для всех задач.
   - Замеряется и выводится время выполнения программы.
10. **Завершение MPI**:
    - Освобождение ресурсов и завершение работы MPI.

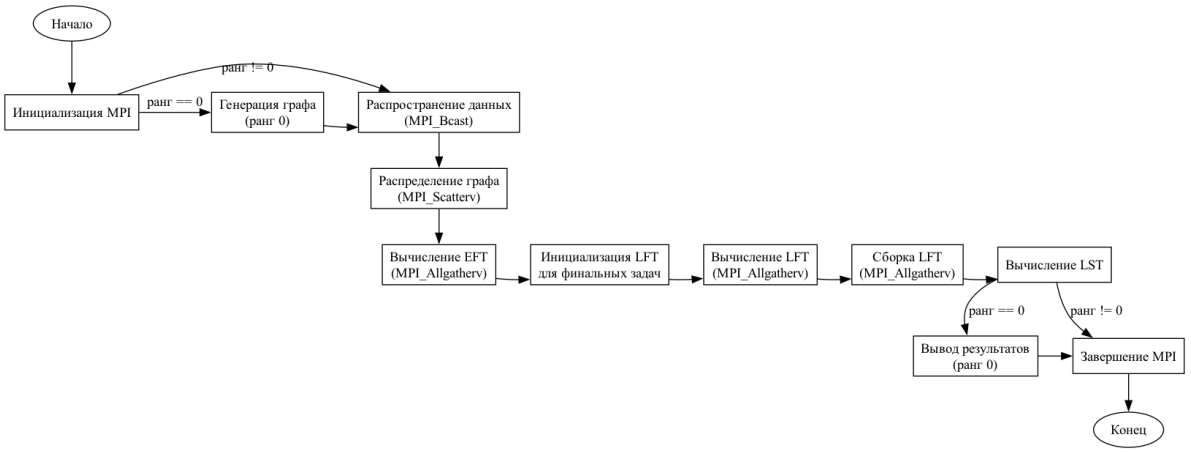

Рисунок 2. Блок-схема параллельного алгоритма.

Количество операций	N=1	N=5	N=10
100	0.16	0.93	1.03
200	1.08	2.99	2.73
300	1.80	8.84	7.3
400	2.44	10.15	9.09

500	5.5	12.44	11.2
600	6.16	13	12.17
700	6.96	13.8	12.08
800	8.07	14.3	13
900	16.87	14.87	13.9
1000	24.21	17.39	15.05

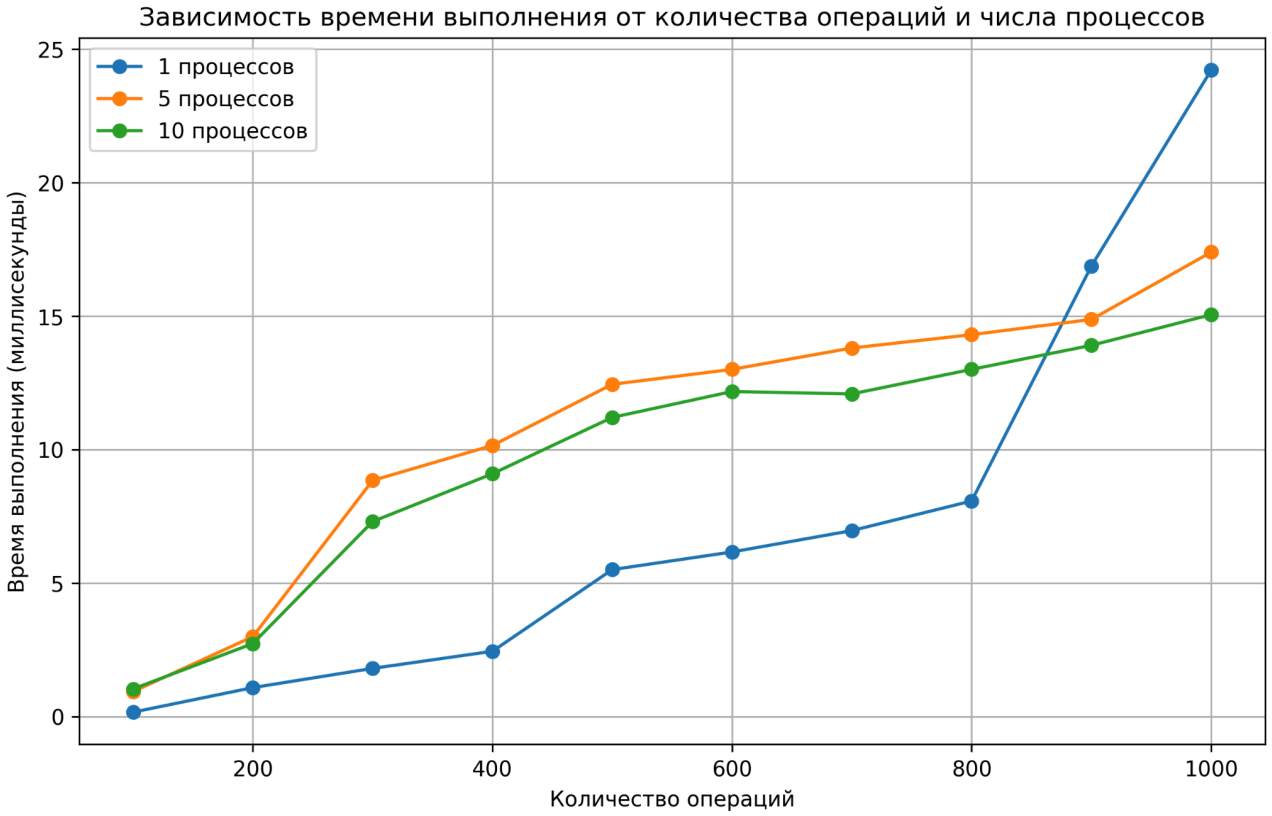

Рисунок 3. Графическое и табличное представление зависимости времени выполнения от количества операций и количества процессов

Доказательство оптимальности параллельного алгоритма

На графике видно, что на малом количестве операций параллельный алгоритм дает результаты даже хуже, чем последовательный. Это может быть связано с накладными расходами на синхронизацию и инициализацию процессов. Однако с ростом количества операций разница сокращается, пока на

900 операциях параллельность не начинает окончательно выигрывать у последовательного подхода.

Ускорение вычисляется как отношение времени выполнения на 1 процессе к времени выполнения на p процессах:

$$S(p) = \frac{T(1)}{T(p)}$$

Например, для 1000 операций:

- T(1) = 24.21 мс
- T(5) = 12.44 мс
- T(10) = 11.2 мс

Ускорение для 5 процессов: $S(5) = \frac{24.21}{12.44} = 1.95$

Ускорение для 10 процессов: $S(10) = \frac{24.21}{11.2} = 2.16$

Эффективность вычисляется как отношение ускорения к числу процессов:

$$E(p) = \frac{S(p)}{p}$$

Для 5 процессов: $E(5) = \frac{1.95}{5} = 0.39$ (39%)

Для 5 процессов: $E(10) = \frac{2.16}{10} = 0.216$ (21.6%)

В идеальном случае ускорение должно быть линейным (например, 5 процессов дают ускорение в 5 раз). Однако на практике ускорение меньше из-за накладных расходов на коммуникацию и синхронизацию.

Алгоритм близок к оптимальному при использовании 5 процессов, так как ускорение составляет около 2 раз, а эффективность — 39%.

При использовании 10 процессов эффективность падает до 21.6%, что указывает на неоптимальность алгоритма для такого числа процессов.

Таким образом, для оптимизированного решения предлагается выполнять несколько правил:

- Использовать параллельность с достаточно большим количеством процессов, чтобы избежать излишней оптимизации;

- С увеличением N лучше увеличивать количество процессов;

Выводы.

В ходе лабораторной работы были реализованы последовательный и параллельный методы подсчёта поздних сроков выполнения операций алгоритма. Последовательный алгоритм демонстрирует классический подход с временной сложностью $O(N^2)$, где N — количество операций. Параллельная реализация, основанная на использовании MPI, позволяет распределить вычисления между несколькими процессами, что значительно ускоряет выполнение задачи для больших объёмов данных.