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

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине «Параллельные алгоритмы и системы»

Тема: ЧИСЛЕННЫЕ МЕТОДЫ

Студент гр. 1307 _____ Николаев К.Д.

Преподаватель _____

Санкт-Петербург

2025

Введение

Тема работы: Численные методы.

Цель работы: Приобрести навыки в распараллеливании программы.

Задание:

8) Метод нахождения поздних сроков выполнения операций алгоритма.

```
Process 1 received vertices from 200 to 399
Process 0 received vertices from 0 to 199
Process 2 received vertices from 400 to 599
Sample Late Finish Times (LFT):
Operation 0: LFT = 1037
Operation 1: LFT = 1054
Operation 2: LFT = 1041
```

```
Operation 596: LFT = 3215
Operation 597: LFT = 3210
Operation 598: LFT = 3213
Operation 599: LFT = 3215
[RESULT] Project Duration: 3215
Total Computation Time: 1008 ms
```

Рисунок 1 - результат работы программы.

1. **Описание последовательного алгоритма;**

Генерация DAG и длительностей операций

- Заполняем верхний треугольник матрицы случайными рёбрами
- Генерируем длительность каждой операции
- Отмечаем финальные вершины (без исходящих рёбер)

Вычисление ранних сроков завершения (EFT)

- По всем вершинам считаем максимум от предшественников
- Итеративно обновляем значения до сходимости

- Инициализация LFT для финальных операций
- LFT финальных узлов = максимальный EFT в графе
- Остальные LFT инициализируются Integer.MAX_VALUE

Вычисление поздних сроков завершения (LFT)

- Идём по графу назад
- Минимизируем LFT через потомков (наследников)
- Итерации продолжаем до сходимости

Вычисление LST

- LST[i] = LFT[i] - durations[i] для каждой операции
- Вывод LST всех операций и времени выполнения

2. Блок-схема алгоритма;

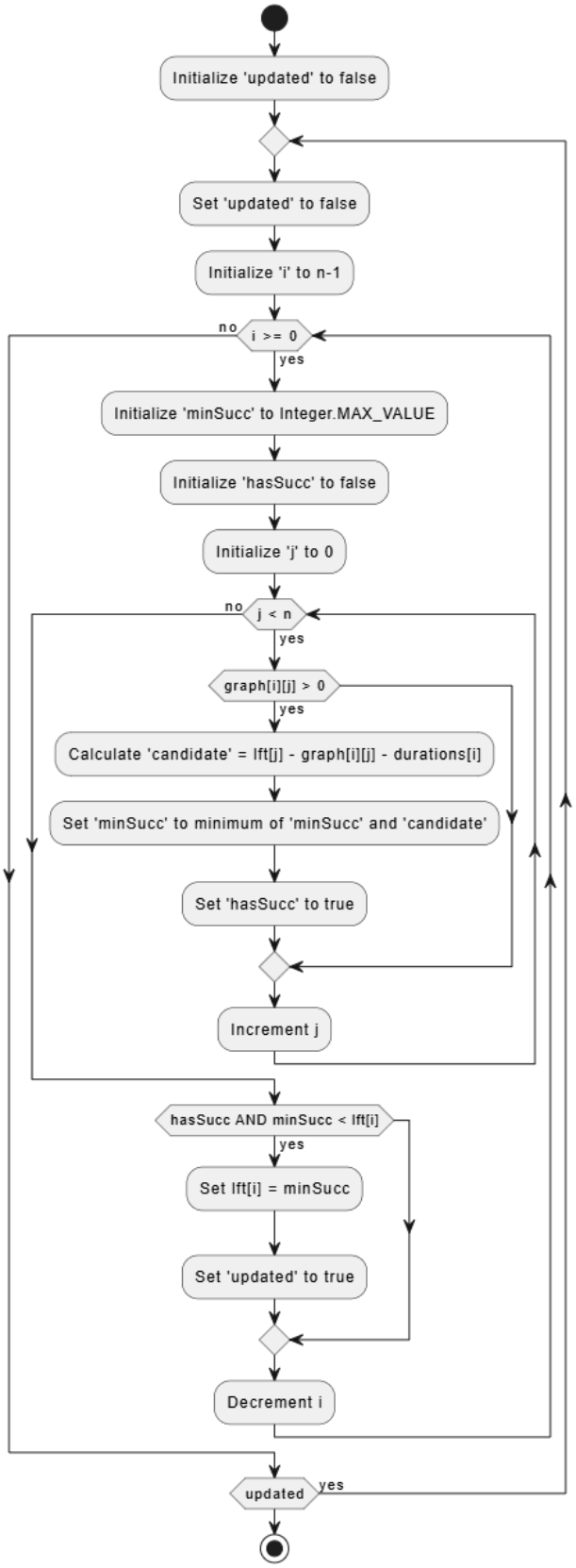

Рисунок 2 - блок-схема алгоритма подсчета позднего срока выполнения операций последовательного алгоритма.

3. **Рассчитанная временная сложность алгоритма;**

Генерация графа: **O(n²)**

Расчёт EFT и LFT в худшем случае — итеративный проход по графу: **O(n² * k)**, где k — количество итераций до сходимости.

Итог: в плотном графе сложность ближе к **O(n²)**.

4. **Схема распараллеленного алгоритма;**

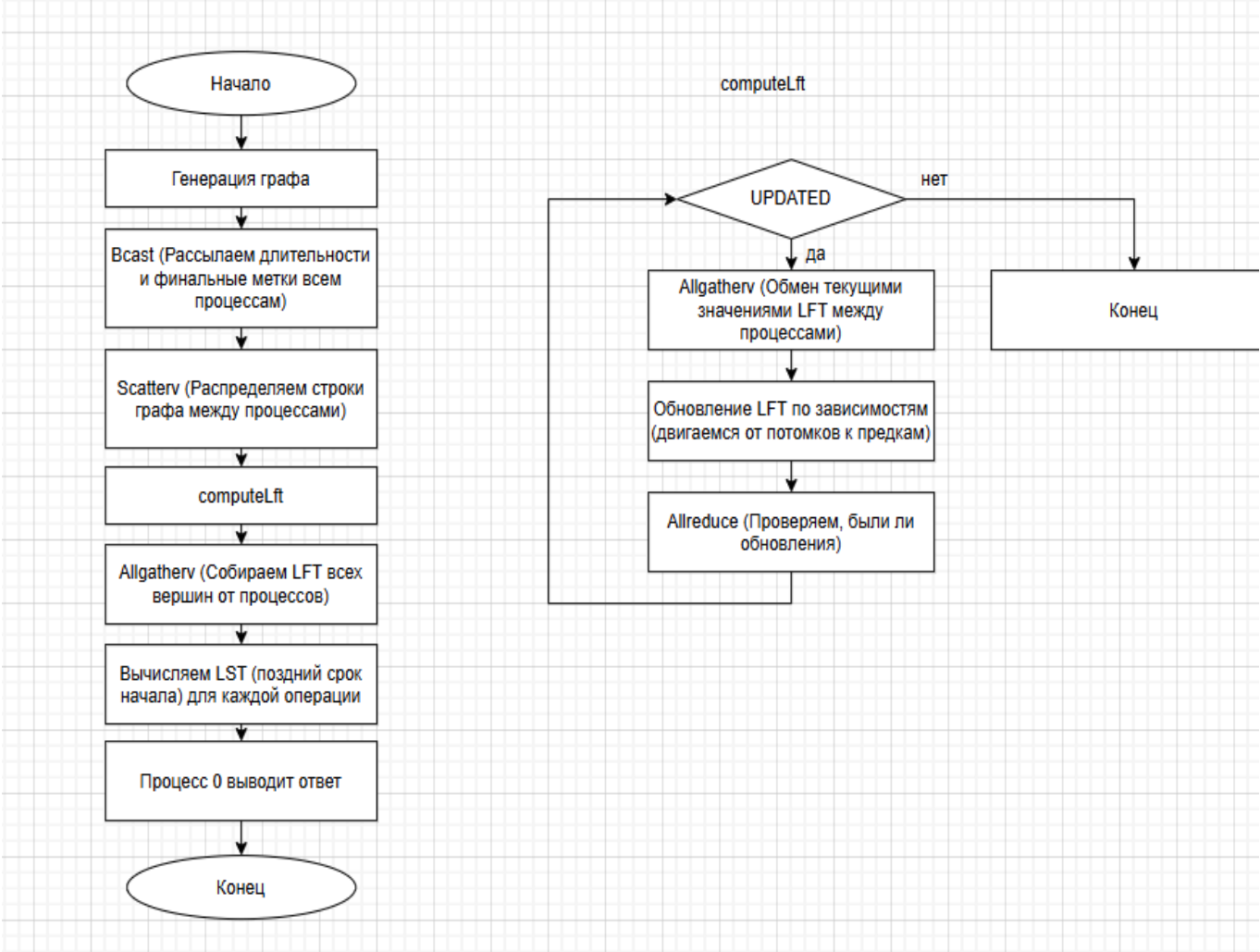

Рисунок 3 - блок-схема алгоритма подсчета позднего срока выполнения операций параллельного алгоритма.

5. **Результаты более 10 тестов для обоих алгоритмов. Результаты должны быть представлены таблично и графически + Оценка оптимальности.**

В ходе эксперимента измерялось время выполнения параллельного алгоритма вычисления поздних сроков выполнения операций для графов различного размера. Для расчётов использовалась реализация на MPJ Express с различным количеством процессов (1, 3, 5 и 7).

Табл.1 - Результаты работы программы.

Размер\Процессы	1	3	5	7
100	0,28	0,5	0,67	1,28
500	2,97	3,32	3,52	5,07
1000	11,02	9,48	8,66	9,59
2000	82,58	53,19	39,1	37,97
3000	152	134	120	107
4000	451	398	293	243
5000	1003	773	531	447
6000	1701	1281	976	801
7000	2312	1799	1523	1243

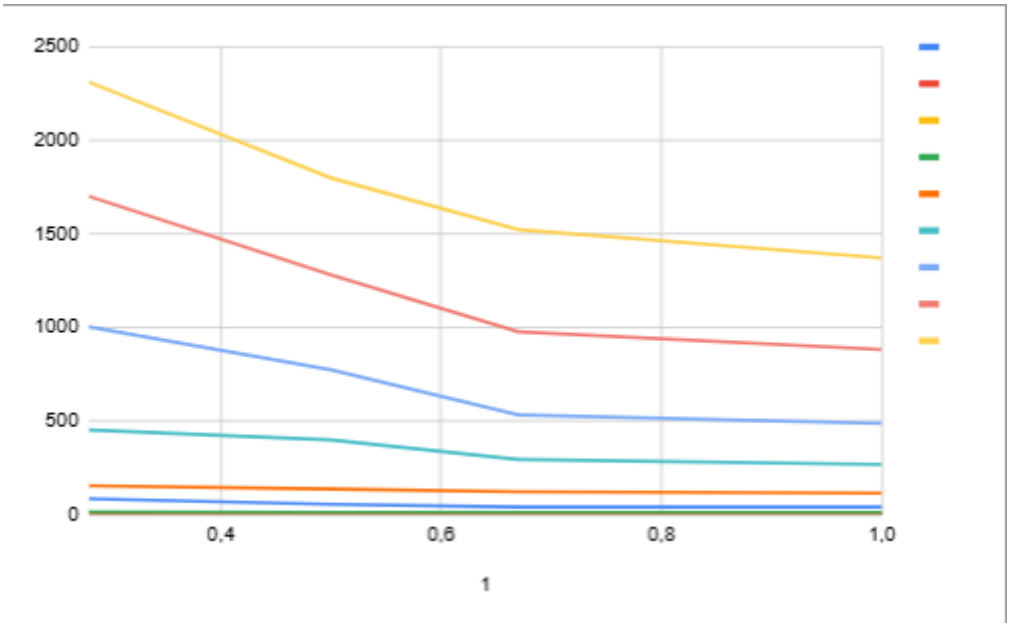

Рисунок 4 - графическое представление (размер графа и зависимость количества процессов и времени выполнения).

На малых размерах задач (100, 500 операций) параллельная реализация не даёт ускорения. Это связано с тем, что расходы на коммуникацию между процессами превышают выигрыш от параллелизма. Время выполнения при увеличении числа процессов даже растёт — это ожидаемое поведение для MPI-программ на малых объёмах данных.

При размере графа 1000 операций начинает наблюдаться выгода от параллельной обработки — время на 5 и 7 процессах сравнимо с последовательным, но с ростом размера уже становится лучше.

Начиная с 2000 операций и выше, параллелизм начинает давать стабильный прирост производительности. Видно, что увеличение числа процессов приводит к сокращению времени выполнения.

Для крупных графов (4000–7000 операций) ускорение от использования 5 и 7 процессов становится особенно заметным:

Например, при 5000 операций время выполнения сокращается с 1003 с на 1 процессе до 447 с на 7 процессах.

Ускорение не является линейным, что типично для MPI-программ из-за накладных расходов на обмен данными и синхронизацию между процессами.

Результаты демонстрируют, что выбранный параллельный алгоритм эффективно работает на средних и больших графах. С ростом размера графа параллельная версия начинает значительно опережать последовательную за счёт распределения вычислений между процессами.

Для маленьких графов (100–500 операций) параллельность неэффективна, эффективность ниже 30% — потери на коммуникации превышают пользу от распараллеливания.

Начиная с 2000 операций заметен рост ускорения — эффективность на 3 процессах доходит до 52%, на 7 процессах — до 31%.

Для больших графов (5000–7000) ускорение сохраняется стабильным:

Например, при 5000 операций ускорение на 7 процессах — 2.24x, эффективность — 32%.

Это хороший результат для распределённых вычислений с учётом накладных расходов на обмен данными.

Тенденция показывает, что эффективность растёт с увеличением размера задачи, как и положено для MPI-алгоритмов. Это хорошо видно по графикам, где мы видим убывающие функции, то есть с увеличением числа процессов (1000+ вершин) функция убывает.

Таким образом алгоритм эффективен для больших графов, когда размер 2000+ тогда видна выгода, и чем больше этот размер, тем эффективнее увеличивается количество необходимых процессов.

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.

Поэтому N должен быть достаточно большим, чтобы расходы на коммуникацию были гораздо меньше, чем расходы на расчет данных. И с увеличением N будет расти и количество процессов на которых эта задача будет выполняться эффективнее

Вывод

В ходе лабораторной работы были реализованы последовательный и параллельный методы подсчета поздних сроков выполнения операций алгоритма.