

Граф алгоритма и параллелизм: как структура зависимостей определяет скорость вычислений

Мурадян Денис Степанович

Санкт-Петербургский государственный университет
Математико-механический факультет
Искусственный интеллект и наука о данных
Бакалавриат, 3 курс

2025



- В параллельном программировании важна не только архитектура компьютера и число ядер, но и **структура самого алгоритма**.
- Будем смотреть на алгоритм как на **граф операций и зависимостей** между ними.
- Один и тот же объём вычислений можно организовать по-разному: с большим количеством независимых шагов или с длинной цепочкой зависимостей.

Цели и структура доклада

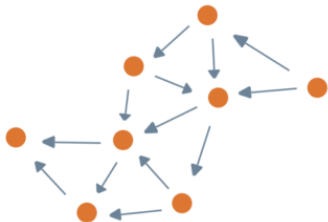
- Структура алгоритма как граф операций и зависимостей
- Базовые понятия теории графов, необходимые для анализа параллелизма
- Граф алгоритма, параллельные формы, ярусы, высота и ширина
- Пример вычисления произведения n чисел и принцип сдваивания
- Концепция неограниченного параллелизма, критический путь и оценка времени выполнения
- Связь графов алгоритмов с реальными системами параллельных вычислений

Алгоритм как ориентированный граф зависимостей

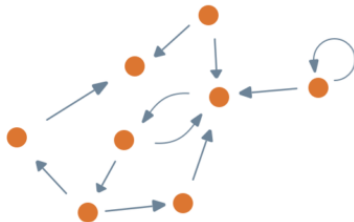
- В графовом представлении вершины соответствуют отдельным операциям или шагам алгоритма.
- Ориентированное ребро показывает отношение предшествования: результат одной операции используется в другой.
- Таким образом, граф фиксирует **частичный порядок** выполнения операций, порождённый зависимостями.

Ацикличность и ориентированные ациклические графы (DAG - directed acyclic graph)

- Для корректного порядка вычислений граф зависимостей должен быть **ациклическим**.
- Ориентированный ациклический граф (DAG) естественно моделирует причинно-временной порядок операций.
- Наличие ориентированного цикла означало бы логическое противоречие: операция косвенно зависит сама от себя.



DAG



Not a DAG

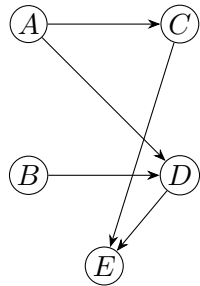
Топологический порядок и выполнение алгоритма

- Любое корректное выполнение алгоритма должно быть согласовано с ориентацией рёбер графа зависимостей. (DAG)
- Задача топологической сортировки: по DAG построить линейный порядок вершин, в котором каждое ребро направлено от более ранней вершины к более поздней.
- Такая сортировка позволяет перейти от частичного порядка зависимостей к конкретному плану выполнения операций и используется при планировании вычислений и распараллеливании.

Граф алгоритма: общее представление

Идея

- Алгоритм представляем в виде ориентированного ациклического графа.
- Вершины обозначают отдельные операции, вычислительные шаги.
- Рёбра фиксируют зависимости «что должно быть выполнено раньше».
- Такой граф задаёт частичный порядок вычислений.



Зависимости в графе алгоритма

Основные виды зависимостей

- **По данным:** результат одной операции используется в другой.
- **По управлению:** выбор ветви или продолжение цикла зависит от предыдущих результатов.
- При формировании графа алгоритма учитываются оба типа, они и задают частичный порядок.

Иллюстрация зависимости по данным

$$t_1 = a + b, \quad t_2 = t_1 \cdot c$$

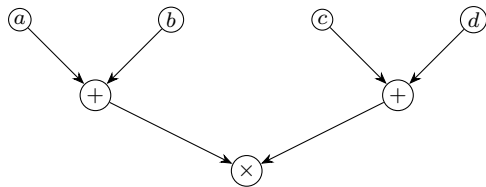
Операция умножения не может начаться, пока не вычислена сумма: в графе есть ребро от вершины t_1 к вершине t_2 .

Примеры графов для простых фрагментов

Арифметическое выражение

$$(a + b) \cdot (c + d)$$

- Сложения $(a + b)$ и $(c + d)$ независимы.
- Оба результата должны быть готовы до умножения.



Цикл без и с зависимостью между итерациями

- Независимый цикл: $x_i = f(i)$ — итерации не зависят друг от друга.
- Рекуррентный цикл: $x_i = x_{i-1} + g(i)$ — образуется цепочка зависимостей.



Параллельные формы графа

Ярусы и параллельная форма

- Рассмотрим ориентированный ациклический граф $G = (V, E)$.
- **Ярус** (уровень) — подмножество вершин, которые можно выполнить одновременно после завершения всех их предшественников.
- **Параллельная форма** — разбиение множества V на упорядоченные ярусы

$$V = L_1 \cup L_2 \cup \dots \cup L_k,$$

такое, что все рёбра идут из более ранних ярусов в более поздние.

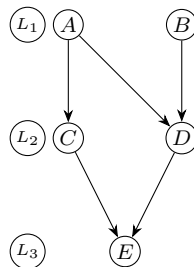
Высота и ширина графа

Характеристики параллельной формы

- **Высота** $h(G)$ — число ярусов в минимальной параллельной форме графа.
- В модели неограниченного числа процессоров $h(G)$ задаёт минимальное число шагов времени.
- **Ширина** $w(G)$ — максимальное число вершин в одном ярусе:

$$w(G) = \max_j |L_j|.$$

- Ширина оценивает потенциальное максимальное число независимых операций на одном шаге.



$$h(G) = 3, \quad w(G) = 2$$

Пример: произведение n чисел — последовательный алгоритм

Схема вычислений для $n = 8$

Данные: $a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8$

Ярус 1: $a_1 a_2$

Ярус 2: $(a_1 a_2) a_3$

Ярус 3: $(a_1 a_2 a_3) a_4$

Ярус 4: $(a_1 a_2 a_3 a_4) a_5$

Ярус 5: $(a_1 a_2 a_3 a_4 a_5) a_6$

Ярус 6: $(a_1 a_2 a_3 a_4 a_5 a_6) a_7$

Ярус 7: $(a_1 a_2 a_3 a_4 a_5 a_6 a_7) a_8$

Характеристики параллельной формы

- Высота: 7 (каждое умножение — отдельный ярус).
- Ширина: 1 (на каждом шаге выполняется одна операция умножения).

Пример: произведение n чисел — процесс сдваивания

Параллельная схема для $n = 8$

Данные: $a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8$

Ярус 1: $(a_1 a_2) \quad (a_3 a_4) \quad (a_5 a_6) \quad (a_7 a_8)$

Ярус 2: $(a_1 a_2 a_3 a_4) \quad (a_5 a_6 a_7 a_8)$

Ярус 3: $(a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8)$

Характеристики параллельной формы

- Высота: 3.
- Ширина на первом ярусе: 4 (четыре независимых произведения пар).
- В общем случае при сдваивании высота порядка $\lceil \log_2 n \rceil$, ширина — порядка $n/2$ на первом ярусе.

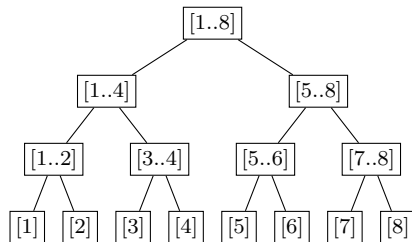
Сортировка слиянием: дерево рекурсий

Структура алгоритма

- Массив рекурсивно делится на две части до подмассивов длины 1.
- На каждом уровне получаем несколько независимых подзадач сортировки.
- Затем результаты попарно сливаются при движении обратно вверх.

Глубина и уровни

- Глубина дерева рекурсий пропорциональна логарифму длины массива.
- На каждом уровне количество подзадач удваивается по мере деления.



Дерево рекурсивных вызовов для $n = 8$

Параллельное выполнение сортировки слиянием

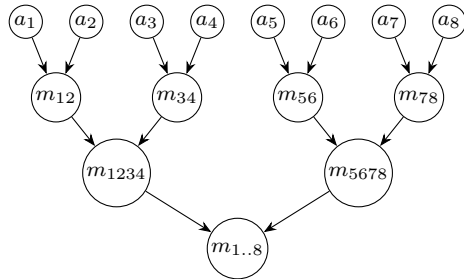
Граф операций слияния для $n = 8$

Уровень 0: $a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8$

Уровень 1: $m_{12} m_{34} m_{56} m_{78}$

Уровень 2: $m_{1234} m_{5678}$

Уровень 3: $m_{1..8}$



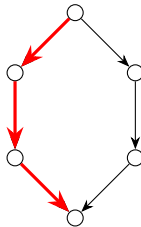
DAG операций слияния

Параллельная форма

- На каждом уровне слияния несколько независимых операций.
- Каждый уровень можно рассматривать как отдельный ярус графа.

Критический путь в графе алгоритма

- Путь в графе — цепочка операций, связанных зависимостями «раньше → позже».
- **Критический путь** — самый длинный путь от входа к выходу графа.
- Его длина совпадает с минимально возможной высотой параллельной формы.
- Операции на критическом пути остаются строго последовательными.



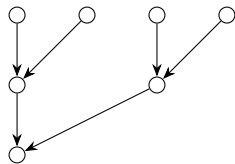
Критический путь выделен
красным

Концепция неограниченного параллелизма

- Идеализированная модель: процессоров сколь угодно много, операции выполняются синхронно.
- Передача данных считается мгновенной и без конфликтов.
- Время вычислений определяется только зависимостями в графе алгоритма.
- **Цель: построение алгоритмов минимальной высоты.**
- Высота графа в такой модели равна нижней границе времени выполнения.

Внутренний параллелизм

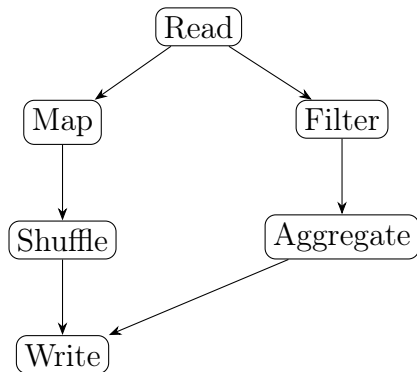
- Внутренний параллелизм задаётся структурой алгоритма — формой графа зависимостей.
- Он определяется длиной критического пути и шириной ярусов.
- Увеличивается не за счёт технологий, а за счёт изменения алгоритма.
- Пример: переход от последовательного перемножения к дереву сдваивания
 - последовательная схема: критический путь $O(n)$;
 - дерево: критический путь $O(\log n)$.



Дерево сдваивания

Графы алгоритмов в современных системах

- Современные вычислительные фреймворки используют DAG как основную модель выполнения.
- Spark: построение DAG при трансформациях, оптимизация планов и распределение задач по узлам.
- Dask: динамическое построение графа операций для задач разной гранулярности.
- TensorFlow/JAX: граф вычислений для оптимизации порядка операций и размещения на устройствах.



Пример DAG-плана в стиле Spark

- Граф зависимостей отражает реальную структуру алгоритма.
- Именно он определяет, какой параллелизм принципиально возможен.
- Ускорение достигается через изменение структуры вычислений.
- Поэтому современные системы используют DAG как естественную форму исполнения.

- **Бурова И. Г., Демьянович Ю. К.**
Алгоритмы параллельных вычислений и программирование.
СПб.: Изд-во СПбГУ, 2007.
- **Воеводин В. В., Воеводин Вл. В.**
Параллельные вычисления.
СПб.: БХВ-Петербург, 2004.
- **A. Grama, A. Gupta, G. Karypis, V. Kumar**
Introduction to Parallel Computing. Addison–Wesley, 2003.