Параллельные алгоритмы

Надуткин Федор

January 2023

Параллельные алгоритмы

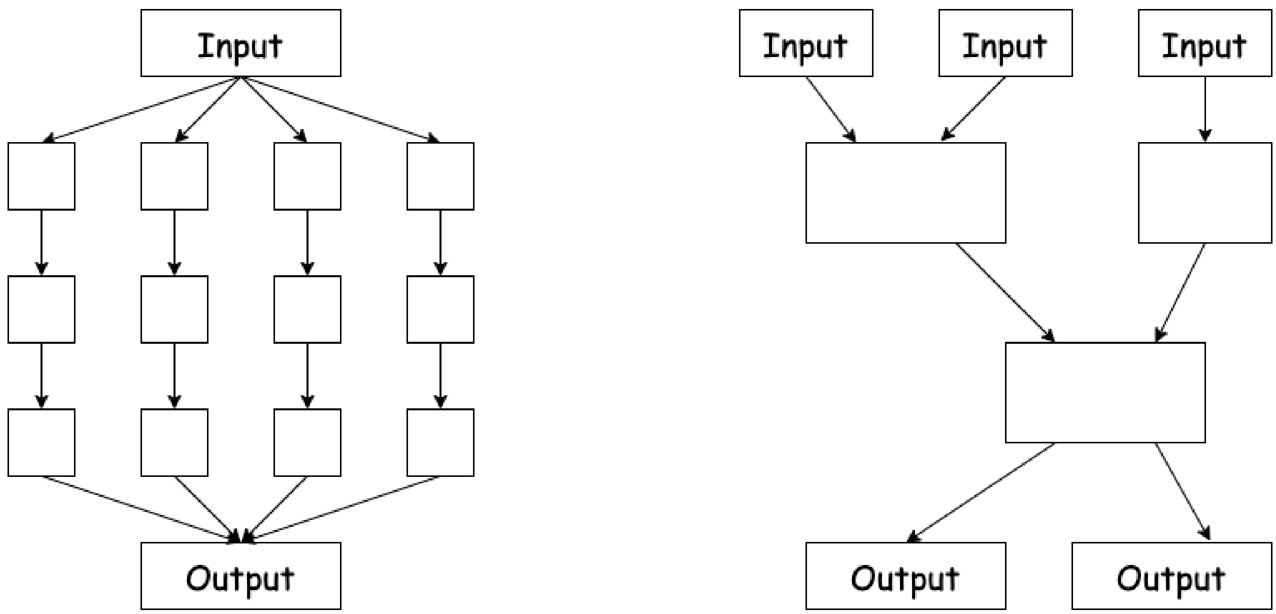


Рис. 1. Параллельные и конкурентные алгоритмы

Разница между параллельными и конкурентными алгоритмами в том, что у параллельных алгоритмов вход и последующая работа синхронизирована, и им не нужно конкурировать за ресурсы, а надо лишь распараллеливать уже имеющуюся работу, тогда как у конкурентных такой синхронизации нет, и им нужно дополнительно о ней заботиться.

PRAM

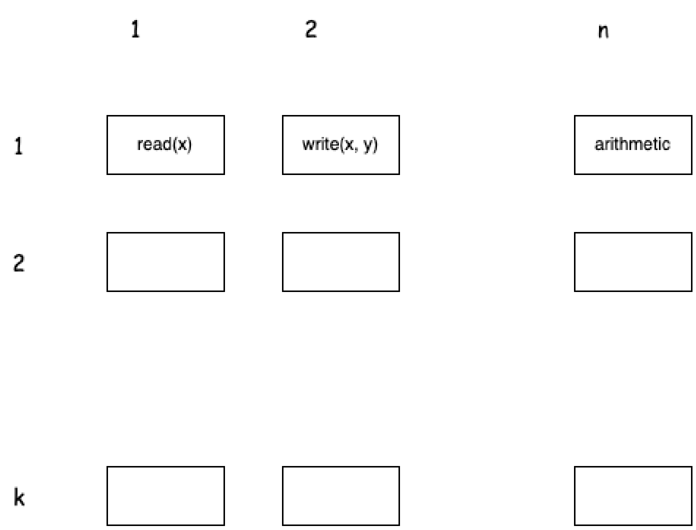


Рис. 2. Parallel Random Access Machine

Изначально была придумана модель PRAM, в которой все операции были разбиты на шаги, и каждый такт n процессов выполняли один из шагов $1, 2, \ldots, k$. Однако в скоре стало ясно, что такая модель достаточно дорогая и требует синхронизации для n потоков каждого шага.

Среди возможных операций были **read(x)** - чтение, **write(x, y)** - запись и **arithmetic** - арифметические операции. Как следствие возникают различные уровни доступа.

- **EREW** - Exclusive reads, Exclusive writes. В каждый момент времени переменную могут либо читать, либо писать, и только лишь один поток.
- **CREW** - Concurrent read, Exclusive write. В каждый момент времени несколько процессов могут читать переменную, но лишь один может писать.
- **CRCW** - Concurrent reads, Concurrent writes. Во время записи будет один победитель, значение которого запишется.
  - *common* - Все процессы пишут одинаковое значение.
  - *priority* - Запишется значение приоритетного потока.
  - *random* - Запишется значение рандомного потока.

Work - сколько работы было сделано. В PRAM это $n \cdot k$
Time - сколько времени работал алгоритм. В PRAM это k

CRCW можно проэмулировать на EREW, однако Time станет равным $k \cdot \log n$. На каждом шаге у нас есть n процессов, мы делаем операции write во временные ячейки, после чего мы начинаем их агрегировать, что можно сделать за $\log n$ времени.

Пример приведён на рисунке.

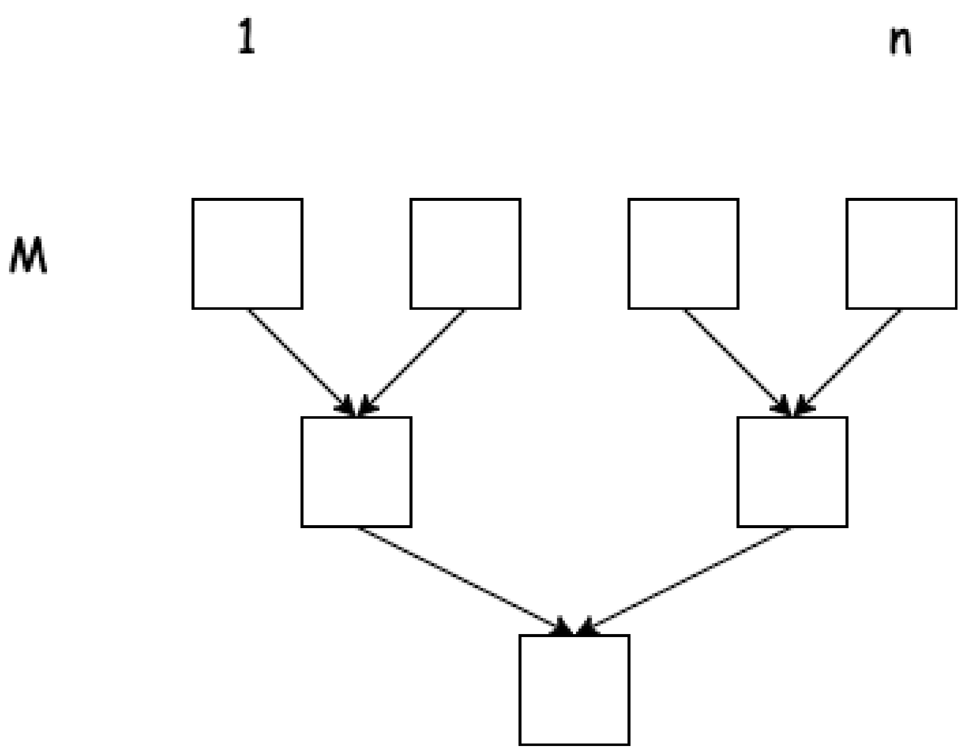


Рис. 3. Перевод из EREW в CRCW

Алгоритм работы PRAM можно видеть ниже. Здесь мы считаем сумму элементов в массиве. Количество элементов равно количеству потоков, равно n. `Time` $= \log n$, `Work` $= n \cdot \log n$.

```
id
read(a, A[id])
write(B[id], a)
for h = 1...log(n):
    if id <= n / (2^h):
        read(B[2 * id - 1], x)
        read(B[2 * id], y)
        z = x + y
        write(B[id], z)
    else:
        skip(4)
```

Листинг 1. Пример подсчёта суммы элементов в массиве

Однако стоит заменить, что у нас на каждом шаге работают лишь $\frac{n}{2^h}$ процессов, как следствие есть идея заменить такую структуру на `pfor`.

```
for h = 1...log(n):
    pfor i = 1...n/(2^h):
        read(B[2 * id - 1], x)
        read(B[2 * id], y)
        z = x + y
        write(B[id], z)
```

Листинг 2. Работа pfor

Однако в реальных моделях создать `pfor` за один такт достаточно проблематично и дорого, поэтому появилась модель `fork-join`, которая порождает лишь 2 потока за раз.

В реальной жизни модель `Time` уже не подходит, время будет сильно зависеть от машин, от того что делается в каждом `fork-join` ..., поэтому `Time` заменяется на **Span** - глубину графа.

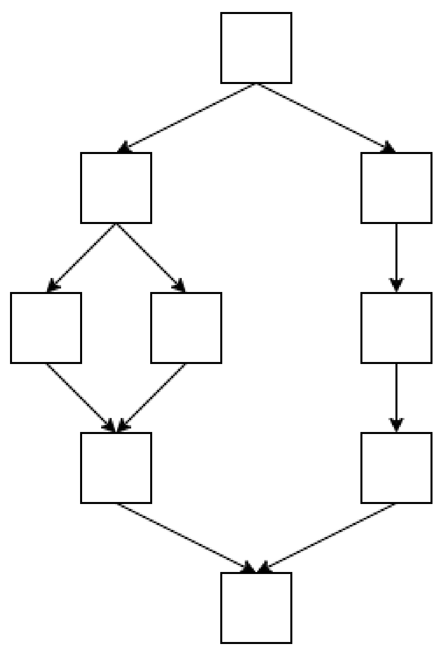


Рис. 4. Пример работы fork-join

На изображении выше `Work` $= 9$, а `Span` $= 5$. Для составления такого графа нужны планировщики.

Теорема Брента

Время работы `level-by-level` планировщика $T \leq \frac{W}{p} + S$, где W — `Work`, S — `Span`, p — количество потоков.

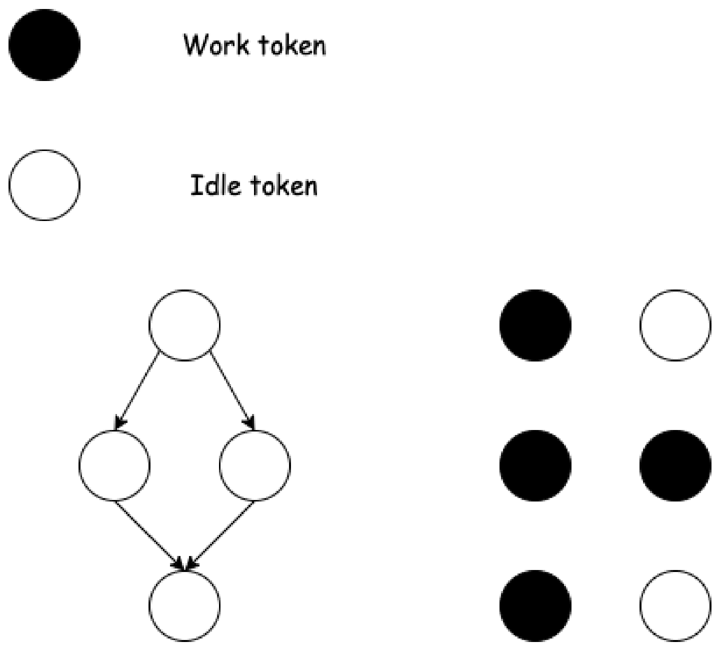
Доказательство

На каждом уровне время работы $= \lceil \frac{W_i}{p} \rceil$, всего уровней у нас `Span`. Суммарное время работы $T = \lceil \frac{W_1}{p} \rceil + \lceil \frac{W_2}{p} \rceil + \ldots \lceil \frac{W_s}{p} \rceil \leq \frac{W_1}{p} + 1 + \frac{W_2}{p} + 1 + \cdots + \frac{W_s}{p} + 1 = \frac{W_1 + W_2 + \cdots + W_s}{p} + S$

Теорема

$\forall$ `greedy-scheduler` $T \leq \frac{W}{p} + \frac{p-1}{p} \cdot S$

Доказательство:



Введём определения

- `Work Token` - платим, когда мы работаем.
- `Idle Token` - платим, когда мы отдыхаем.

Когда мы платим хотя бы один `Idle Token` наш `Span` уменьшается на 1. За каждый уровень мы платим $\leq S \cdot (p-1)$ (хотя бы один процесс должен работать). Так как каждую итерацию нам приходит p токенов (у нас p процессов), то $T \leq \frac{W}{p} + \frac{p-1}{p} \cdot S$

Как следствие $T \leq \frac{W}{p} + S$, а значит брать $p \geq \frac{W}{S}$ имеет мало смысла.