Лекция 18

Параллельные алгоритмы

(Конспект: К. Орлова)

18.1 Параллельные вычисления

Введем следующие обозначения:
- $c(n)$ — количество процессоров,
- $T(n)$ — время работы,
- $W(n) = c(n)T(n)$ — общая работа.

Заметим, что если у нас есть программа, написанная для $c(n)$ процессоров, то мы всегда сможем переписать ее для меньшего количества $c_1(n)$; полученная программа будет работать время $O(W(n)/c_1(n))$.

Модель вычислений можно выбрать, например, такую: RAM-машина с $c(n)$ процессорами; у каждого — одна и та же программа; нулевой регистр у каждого свой (перед началом работы там записан номер процессора), остальные — общие. У нас может возникнуть проблема коллизий, то есть два процессора решат одновременно записать данные в один регистр. Ее можно решить двумя способами:
- запретить такие програмы;
- записывать то, что записал процессор с большим номером.

18.2 Достижимость в графе

Пусть A — матрица смежности. Если мы умножим ее булевски на себя, то получим те ребра, которые достижимы за два шага. Если в матрицу A записать на диагональ единички, то квадрат даст нам вершины, достижимые за два или менее шагов. Матрицу A^2 умножим ее еще раз на себя: мы получим вершины, достижимые за четыре или менее шагов, и т. д. Итого: нам нужен алгоритм, возводящий A в $(n-1)$-ую (или большую) степень. Будем считать[1], что $n - 1 = 2^k$.

[1] Ясно, что это несущественно.

```
A  A  A  A ... A  A
 \/    \/       \/

 A^2   A^2  ...  A^2
....................
         \/

       A^(n-1)
```

n процессоров сделают сие действо за $O(\log(n))$ операций над матрицами. Но произведение матриц не есть элементарная операция. Пусть мы хотим умножить матрицы B и C, из булево произведение — D; тогда $d_{ij} = \bigvee\limits_{k=1}^{n} b_{ik} \wedge c_{kj}$ можно вычислить следующим способом:

```
b_i1 ∧ c_1j   b_i2 ∧ c_2j  ...  b_in ∧ c_nj
         \ /    ...    \ /
          \   /     \   /
           \/        \/
            \        /
             \     /
              \  /

             d_ij
```

При этом каждый из элементов d_{ij} мы вычисляем на своем наборе процессоров. За $O(\log(n))$ операций и n^3 процессоров мы перемножим две матрицы. Всего же мы управимся за время $O(\log^3 n)$ и $O(n^4)$ процессоров.

18.3 Максимальное по включению независимое множество

Определение 18.1. $V' \subseteq V$ — *независимое множество* в графе (V, E), если $\forall u, v \in V'$ $\{u, v\} \notin E$.

Определение 18.2. Обозначим через $deg(v)$ степень (количество соседей) вершины v.

Алгоритм.
Вход: граф $G = (V, E)$.
Выход: независимое множество S.

Инициализация: $S := \emptyset$.

На каждой итерации мы будем делать следующее.

1. $\forall v \in V$:
   если $deg(v) = 0$, то добавить v в S;
   если $deg(v) \neq 0$, то пометить v с вероятностью $1/(2deg(v))$.
2. $\forall e \in E$:
   если оба конца ребра e помечены, то снять пометку с того, у которого степень меньше.
3. Запихать в S все помеченные вершины.
4. $V := V \setminus \{S \cup \text{соседи вершин из } S\}$ (при этом надо и E обновить соответственно).

Повторять эти шаги будем, пока граф не станет пустым.

Анализ алгоритма. Корректность алгоритма очевидна. Шаги этого алгоритма можно выполнять параллельно для всех вершин или ребер, при этом (при разумной реализации) на один шаг будет тратиться время $O(\log n)$. Остается выяснить, сколько будет итераций: если их также $O(\log n)$, то мы получили искомый «быстрый» параллельный алгоритм. В дальнейшем этот факт не будет доказан полностью; однако, будут приведены соображения, подтверждающие это интуитивно.

Будем называть вершину *хорошей*, если $\geqslant 1/3$ ее соседей имеют степень меньшую, чем степень этой вершины.

Лемма 18.1. *Пусть v — хорошая вершина. Тогда вероятность, что один из ее соседей помечен $\geqslant 1 - e^{-1/6}$.*

Доказательство. Действительно, вероятность, что это не так, не превосходит $(1 - 1/2deg(v))^{deg(v)/3}$, что, в свою очередь, не больше $e^{-1/6}$. □

Лемма 18.2. *Вероятность снятия пометки — не более $1/2$.*

Доказательство. Данная вероятность не превосходит вероятности того, что помечен один из соседей большей степени; что не меньше, чем $deg(v) \cdot 1/(2deg(v))$, то есть $1/2$. □

Следствие 18.1. *Если v — хорошая, то с вероятностью $\geqslant (1-e^{-1/6})/2$ хотя бы один из ее соседей попадет в S.*

Ребро e будем называть *хорошим*, если хотя бы один из его концов — хороший.

Лемма 18.3. *Количество хороших ребер $\geqslant |E|/2$.*

Доказательство. Направим ребра из вершин с меньшей степенью в вершины с большей[2] (теперь наш граф — ориентированный; пусть $deg_{in}(v)$ и $deg_{out}(v)$ — входная и выходная степени вершины v соответственно).

Через $E(V_1, V_2)$ обозначим количество ребер, идущих из вершин множества V_1 в вершины множества V_2; пусть V_g — множество всех хороших вершин, V_b — множество всех плохих вершин.

[2]И как угодно для ребер, соединяющих вершины одинаковой степени.

$$
\begin{aligned}
&2E(V_b, V_b) + E(V_b, V_g) + E(V_g, V_b) \leqslant \\
&\sum_{v \text{ — плохая}} deg(v) \leqslant \\
&3 \sum_{v \text{ — плохая}} (deg_{out}(v) - deg_{in}(v)) = \\
&3(E(V_b, V_g) + E(V_b, V_b) - E(V_g, V_b) - E(V_b, V_b)) \leqslant \\
&\qquad\qquad\qquad 3(E(V_b, V_g) + E(V_g, V_b)).
\end{aligned}
$$

Сократим начало и конец этого неравенства. Итого, у нас хороших ребер больше, чем плохих, а значит, их больше половины. □

Хороших ребер больше половины; каждое из них пропадает с вероятностью $\geqslant (1 - e^{-1/6})/2$; значит, мат. ожидание количества пропавших ребер $\geqslant (1 - e^{-1/6})|E|/4$. Выходит, что на каждой итерации количество ребер сокращается в константное число раз[3]; значит, алгоритм будет совершать лишь логарифмическое число итераций.

[3] Увы, лишь в среднем. Поэтому последующий вывод доказывается не так просто.