Лекция 10

Параллельные вычисления

(Конспект: С. Федин)

Хочется, параллелизовав полиномиальные по времени вычисления, получить от этого «экспоненциальную выгоду». Пусть вычисление занимало $\mathrm{poly}(n)$ времени; тогда мы хотим, имея $\mathrm{poly}(n)$ процессоров, проделать это вычисление за $\mathrm{poly}(\log n)$.

Заметим, что если мы научимся это делать для большего числа процессоров, то для меньшего мы это сделать тоже сможем: будем объединять несколько процессоров в один и моделировать на нем работу каждого из них по очереди (чередуя шаги, делаемые за разные процессоры). Таким образом, нам надо параллелизовать вычисления «как можно сильнее», оставив общий объем работы разумным (т.е. полиномиальным).

Модель вычислений, которой мы будем пользоваться, — это булевы схемы. Каждый гейт схемы можно рассматривать как процессор, выполняющий одну свою операцию. Тогда временем работы этого многопроцессорного устройства, очевидно, соответствует длина самого длинного ориентированного пути от входного гейта к выходному (глубина схемы), а объему работы (т.е. общему количеству шагов, сделанному всеми процессорами) — размер схемы.

Однако, схема способна обрабатывать лишь входы заданной длины. Поэтому мы будем рассматривать *семейства схем*, по одной для каждой возможной длины входа. Чтобы такое семейство можно было использовать автоматически, необходимо, чтобы можно было *эффективно построить схему* для заданной длины входа. Мы будем требовать, чтобы существовала детерминированная машина Тьюринга, порождающая схему (в разумном виде — так, чтобы ее вычисления можно было моделировать на машине Тьюринга, которой она будет задана вместе со входом) для входов длины n по записи числа n в «палочной» системе счисления и при этом использующая лишь $O(\log n)$ ячеек памяти (и, следовательно, лишь полиномиальное от n время). Такие семейства схем называются *равномерными*. Очевидно, что равномерные схемы — полиномиального размера.

Определение 10.1.

$$\mathbf{NC}^i = \left\{ L \;\middle|\; \begin{array}{l} L \text{ принимается некоторым равномерным семейством} \\ \text{схем глубины } O(\log^i n), \text{ где } n \text{ — размер входа} \end{array} \right\}.$$

$$\mathbf{NC} = \bigcup_i \mathbf{NC}^i.$$

Хочется понять взаимосвязь нового класса с уже известными классами. Очевидно, что $\mathbf{NC} \subseteq \mathbf{P}$ (является ли это включение строгим — открытый ворпос).

Теорема 10.1. *$\mathbf{NC}^1 \subseteq \mathbf{DSpace}(\log) \subseteq \mathbf{NSpace}(\log) \subseteq \mathbf{NC}^2$.*

Доказательство. 1. Второе включение очевидно.

2. Докажем последнее включение, т.е. что $\mathbf{NSpace}(\log) \subseteq \mathbf{NC}^2$. Рассмотрим язык, лежащий в $\mathbf{NSpace}(\log)$, и недетерминированную машину Тьюринга M, которая его принимает. *Псевдоконфигурацией* в этой теореме назовем конфигурацию, в которую не включается входная лента. Рассмотрим граф всех возможных псевдоконфигураций машины M (наличие каждого из ребер в этом графе, очевидно, зависит от входа). Наша задача — проверить, есть ли (для данного входа) в этом графе путь из начальной псевдоконфигурации в принимающую. Заметим, что размер псевдоконфигурации ограничен $O(\log)$. Таким образом, нам надо решить задачу достижимости в графе, содержащем полиномиальное число вершин (обозначим это число через k).

Покажем, что эта задача лежит в $\mathbf{NC}^2$. Пусть A — матрица смежности нашего ориентированного графа. Наша задача — вычислить булеву степень[1] A^k схемами глубины $\log^2 k$.

Заметим, что для этого достаточно $\log k$ последовательных умножений пар матриц: A^2, $(A^2)^2$, …(если получим в итоге чуть бо́льшую степень, чем k — а именно, ближайшую к k сверху степень двойки — не страшно: $A^k = A^{k+1} = \ldots$). Для умножения же одной пары матриц $B \cdot C$ достаточно глубины схем $O(\log k)$: параллельно вычислим все произведения вида $b_{il} \wedge c_{lj}$, затем вычислим их суммы $\bigvee_{l=1}^k b_{il} \wedge c_{lj}$ также параллельно, затрачивая на каждую сумму лишь логарифмическое время (глубину) — сначала складываем слагаемые по два, затем полученные частичные сумму — снова по два, и т д.

[1]Булево произведение матриц — это произведение, в котором в качестве операции сложения используется дизъюнкция, а качестве операции умножения — конъюнкция.

Итак, мы доказали, что достижимость в графе можно выяснить в $\mathbf{NC}^2$. На вход этой схеме мы должны подать элементы матрицы смежности, т.е. ребра графа (1 = ребро есть, 0 = ребра нет). Понятно, что для любых двух псевдоконфигураций мы можем в $\mathbf{NC}^2$ выяснить, получена ли одна из них из другой (входом для этой схемы являются биты входной ленты! псевдоконфигурации входами не являются: для каждой интересной нам пары псевдоконфигураций мы строим свою подсхему).

3. Докажем теперь, что $\mathbf{NC}^1 \subseteq \mathbf{DSpace}(\log)$.

Отступление. Пусть у нас имеется две машины Тьюринга M_1 и M_2, вычисляющие некоторые функции и затрачивающие лишь логарифмическую память на свои вычисления. Нам хотелось бы вычислить композицию этих функций.

Первая мысль, которая приходит в голову — сделать выходную ленту одной машины входной лентой для другой и запустить их по очереди. Однако так сделать нельзя, поскольку входная и выходная ленты не учитываются в ограничении по памяти M_1 и M_2 и могут (и будут) иметь длину большую, чем логарифм от входа первой машины; при таком же объединении машин эти ленты становятся рабочими и память на них следует учитывать.

Поэтому поступим иначе: вместо того, чтобы хранить выход M_2 и вход M_1, мы будем хранить лишь счетчики позиций на соответствующих лентах, и изменять их соответственно тому, как двигались бы головки. Именно, запустим M_2 и, когда ей понадобится очередной i-й бит входа (i хранится в ее счетчике), прервем ее вычисление, запустим M_1 (с самого начала) и дадим ей доработать до тех пор, пока ее счетчик позиции на выходной ленте не станет равен i и она захочет записать на выходную ленту очередной символ[2]. Этот символ мы и используем, чтобы совершить очередной переход (определить следующую конфигурацию) машины M_2, и продолжим вычисления.

Пусть есть теперь язык $L \in \mathbf{NC}^1$. Докажем, что $L \in \mathbf{DSpace}(\log)$. Пусть на вход нам подали x. Надо выяснить, верно ли, что $x \in L$. Мы сделаем это, построив композицию трех функций, вычисляемых с логарифмической памятью.

Первая функция строит схему для входов длины той же, что у x. Существование соответствующей логарифмической по памяти машины Тьюринга следует из определения $\mathbf{NC}^1$.

Вторая функция преобразует эту схему в формулу (т.е. делает из ориентированного ациклического графа, представляющего схему, дерево). Сделаем мы это следующим образом: каждый гейт формулы будет

[2]Можно считать, что в каждую позицию выходной ленты символ записывается лишь однажды.

обозначаться битовой строчкой, обозначающей путь к некоторому гейту исходной схемы от ее выходного гейта. (В этой строчке 1 будет означать, что мы пошли направо, а 0 — налево.) Понятно, что перечислить эти вершины, их типы ($\vee, \wedge, \neg$) и ребра между ними мы можем, ограничившись логарифмической памятью, устроив в исходной схеме поиск в глубину (по ребрам, обратным заданным) (правда, чтобы вернуться к предыдущей вершине, придется пройтись еще раз от корня до нее, поскольку мы можем хранить только «куда мы пошли» — налево или направо — но не «откуда пришли»).

Третья функция вычисляет значение формулы на входе x. Для этого будем обходить дерево и вычислять значение левого поддерева, а, когда надо, и правого. «Когда надо» здесь означает следующее: если вершина помечена «$\vee$», то правое поддерево надо вычислять только в том случае, если значение в левом поддереве 0, а если помечена «$\wedge$», то надо вычислять только в том случае, если значение в левом поддереве 1.

То, что для этого достаточно логарифмической памяти, очевидно. Вот, впрочем, формальное доказательство.

Заметим, что нам достаточно хранить только битовую строчку, обозначающую текущий гейт c и переменную b, принимающую одно из пяти значений:

$b = 1$, если мы приступили к обработке данной вершины,

$b = 2$, если уже обработали левое поддерево с результатом 0,

$b = 3$, если уже обработали левое поддерево с результатом 1,

$b = 4$, если уже обработали правое поддерево с результом 0,

$b = 5$, если уже обработали правое поддерево с результом 1.

Начинаем мы с пустой строки c (соответствующей выходному гейту) и $b = 1$. Опишем работу алгоритма на гейте, вычисляющем «$\wedge$» (для «$\vee$» — аналогично).

Если $b = 1$, то $c := c0$, $b := 1$;

если $b = 2$, то «обрезаем» последний бит c, и

> **если** этот бит был 0, то $b := 2$, а
> **если** этот бит был 1, то $b := 4$;

если $b = 3$, то $c := c1, b := 1$;

если $b = 4$, то поступаем аналогично случаю $b = 2$;

если $b = 5$, то «обрезаем» последний бит c и

> **если** этот бит был 0, то $b := 3$, а

если этот бит был 1, то $b := 5$.

Понятно, что памяти мы затратили снова лишь $O(\log)$ на хранение c и b.

$\square$