Параллельное построение наибольшего независимого множества

Марк Голдберг, Томас Спенсер.

РЕФЕРАТ

Рассмотрена задача параллельного построения наибольшего независимого множества данного графа. Представлен новый детерминированный NC-алгоритм для модели EREW PRAM. Для графов с n вершинами и m ребрами он использует O((n+m)/lg*n*) процессоров и завершает работы за время O($\log^3$n), уменьшая в log*n* раз и время, и количество используемых процессоров, по сравнению с предыдущими детерминированными алгоритмами, решающими задачу, используя линейное количество процессоров.

Ключевые слова: параллельные вычисления, NC, граф, наибольшее независимое множество, детерминированность.

1. Введение

Задача параллельного построения наибольшего независимого множества данного графа, HHM, исследовалась в нескольких недавних статьях. Карп и Вигдерсон [KW] доказали, что задача принадлежит к классу NC. Их алгоритмы находят HHM графа с n вершинами за время O($\log^4$n), используя O($n^3/\log^3$n) процессоров. В других статьях авторы предлагают алгоритмы, работающие быстрее, или требующие меньше процессоров. Луби в [L1] и Алон и др. в [ABI] представили вероятностный алгоритм с временной сложностью O($\log^2$n) для EREW PRAM с линейным числом процессоров. Луби также описал технику преобразования вероятностных алгоритмов в детерминированные; метод сохраняет временную сложность, но увеличивает число используемых процессоров до O(n^4m). Первый детерминированный NC-алгоритм с линейным числом процессоров (модель EREW) был построен в [SG]; его временная сложность составляет O($\log^4$n). Недавно, Луби [L2] предложил общий метод для преобразования вероятностных алгоритмов в детерминированные, который не требует увеличения числа процессоров. В случае HHM, метод дает новый алгоритм, требующий линейное число процессоров в течении полилогарифмического времени; однако, он работает медленнее, чем описанный в [GS]. Все NC-алгоритмы для HHM, упомянутые ранее, работают по следующему принципу, предложенному в [KW]:

> **Изначально, множество I пусто. Находим независимое множество I', добавляем его к I, и удаляем I' и все вершины, смежные с вершинами I' из графа. Повторять предыдущие шаги, пока граф не пуст.**

Назовем процедуру, строящую I' FINDSET. Легко доказать, что такая структура принадлежит к классу NC, если FINDSET разработана так, что на любом графе на n вершинах (n>0), она требует полилогарифмическое время и выдает такое независимое множество C, что $\left|C \cup N(C)\right| = \Omega\left(n / \log^s n\right)$ для некоторого фиксированного $s \geq 0$; (N(C) – множество соседей C – функция соответствия).

В этой статье, мы представляем новый детерминированный алгоритм для HHM, который улучшает временную сложность алгоритма из [GS] в logn раз и также уменьшает число требуемых процессоров в logn раз. Алгоритм реализован в вычислительной модели EREW. Таким образом, процессорное время получения результата алгоритмом из [GS] улучшено в $\log^2$n раз. Это получается благодаря новой версии процедуры FINDSET. Новая процедура находит за время O($\log^2$n) независимое множество I, такое, что удаление $I \cup N(I)$, уменьшает размер графа вдвое. Таким образом, процедура FINDSET вызывается только O(logn) раз.

Чтобы уменьшить число необходимых процессоров, мы изменили определение размера графа так, что оно учитывает количество удаленных ребер. В следствие

модификации можно применить метод уменьшения числа процессоров Миллера и Рейфа [MR], для уменьшения числа необходимых процессоров в O(logn) раз.

Оба алгоритма (из [GS] и рассматриваемый в этой статье)используют идею частичной раскраски. Частичная раскраска – присваивание некоторым, не обязательно всем, вершинам цветов так, что любые две смежные вершины имею разный цвет. Примеры частичной раскраски включают тривиальную раскраску, когда каждой вершине назначен свой собственный цвет, и пустую раскраску, когда не раскрашена ни одна вершина. Если раскраска назначает цвета всем вершинам, она называется полной. Множество вершин с одинаковым цветом называется цветовым классом.

FINDSET строит последовательность частичных раскрасок, начиная с тривиальной. Цель – найти «большой» цветовой класс C (термин «большой» определен далее). Если «большой» класс C найден, $C \cup N(C)$ удаляется из графа; если нет «большого» класса, некоторые цветовые классы объединяются. Побочный эффект этого – некоторые вершины остаются неокрашенными. FINDSET останавливается, когда все вершины либо удалены, либо обесцвечены. Функция возвращает сумму всех найденных «больших» цветовых классов.

Успешность такого подхода определяется определением «больших» цветовых классов и эффективностью их объединения. Когда алгоритм решает объединять, он должен делать это «быстро» и так, чтобы получалось «не слишком много» неокрашенных вершин. Формальный способ, в котором выполняется эта задача, – сравнение двух представлений множества ребер E раскрашенного графа. Первое представляет E как объединение множеств L(C) ребер, смежных вершиной цветового класса C, где C изменяется в пределах множества всех цветовых классов. Грубо говоря, L(C) показывает размер части графа, которая будет удалена, если C добавить к I. Второе представление E делит ребра на классы так, что принадлежность ребра к классу определяется по цветовым классам обеих смежных вершин. Каждый класс второго представления определяет размер части графа, которая станет неокрашенной, если смешивание будет проводиться согласно этому классу. Теоремы 1 и 2 определяют отсутствие больших классов, затем эффективное смешивание.

Мы ожидаем, что рассмотрение этих представлений сможет помочь при разработке параллельных алгоритмов для других задач на графах.

2. Терминология

Определения класса NC и вычислительных моделей можно найти в [P], [V], [KR]. Терминология теории графов в этой статье используется в соответствии с [BM]. Степень вершины v подграфа H обозначается как $\deg_H(v)$. Для данного множества вершин $K \subset V(G)$, $\sigma_H(K) = \sum_{v \in K}(1 + \deg_H(v))$ называется весом K в H. Если понятно, о каком графе идет речь, – индексы в $\deg_H(v)$ и $\sigma_H(K)$ опущены. Мы используем вес как определение размера, когда мы говорим, что FINDSET находит независимое множество I, такое, что удаление $I \cup N(I)$ уменьшит размер H вдвое.

Пусть $C = \{C_o, ..., C_{r-1}\}$ – набор цветовых классов частичной раскраски ϕ. Тогда L_i означает множество ребер, смежных с одной вершиной из C_i, а N_i определяет множество вершин, смежных с C_i. Рассмотрение L_i-х выводов– первый метод классификации ребер. Чтобы понять второй метод, нам необходимо определить функции:

$$\chi(p) = \begin{cases} p, & \text{если } p \text{ нечетное} \\ p-1, & \text{если } p \text{ четное} \end{cases}$$

$$rev(i, q; p) = (q - i) \bmod \chi(p)$$

$$index(i,j;p)=\begin{cases}(i+j)\bmod\chi(p) & \text{если } 0\le i,j\le\chi(p)-1\\ 2i\bmod\chi(p) & \text{если } j=\chi(p)\\ 2j\bmod\chi(p) & \text{если } i=\chi(p)\end{cases}$$

Легко доказать, что для каждого q и i ($0\le i,j\le\chi(p)-1$), $j=rev(i,q;p)$ единственное целое число в интервале $[0,\chi(p)-1]$, такое что $index(i,j;r)=q$.

Пусть ϕ – раскраска вершин, а $C=\{C_o,...,C_{r-1}\}$ – множество цветовых классов. Пусть, для каждого $k=0,...,\chi-1$, $\pi=\{(C_i,C_j):index(i,j;r)=k\}$ – разбиение C на $\lfloor r/2\rfloor$ пар, и, возможно, один единичный класс. Эти χ разбиений C называют формальным разбиением C. Легко проверить, что для всех $i\ne j,\pi_i\ и\ \pi_j$ не имеют общих пар (элементов). Таким образом, множество $\{\pi_0,...,\pi_{\chi-1}\}$ представляет собой минимальную реберную раскраску полного графа с вершинами $C_0,...,C_{r-1}$.

3. Схема алгоритма

В этом разделе мы представляем описание FINDSET и доказываем, что каждый вызов FINDSET уменьшает вес графа вдвое.

Пусть G – граф с n вершинами и m ребрами. FINDSET начинает определением пустого независимого множества I и тривиальной раскраски вершин G. Затем, она продолжается итерационно. На каждой итерации, выполняется одно из следующих действий:

- <1> Если найден цветовой класс C*, такой что $\sigma(C^*)\ge(n+m)/\log n$, то все вершины из C* добавляются к I, а все вершины из $C^*\cup(N(C^*))$ удаляются из графа.
- <2> Цветовые классы разбиваются на пары (C_i,C_i'), с, возможно, одним свободным классом. Два цветовых класса каждой пары смешиваются. Чтобы сделать смешенное множество независимым, вершины множеств $C_i\cap N(C_i')$ и $C_i'\cap N(C_i)$ сравниваются, и вершины множества с меньшим весом теряют цвет.

Действие <1> выполняется когда возможно; действие <2> выполняется, если нет подходящего класса C*. Действия выполняются до тех пор, пока не останется не более одного цветового класса. Если действительно остается один цветовой класс, тогда, независимо от его веса, он добавляется к I.

Действие <2> выполняется процедурой HALF. Она строит регулярное разбиение π минимального веса и смешивает каждую пару цветовых классов, соединенную π.

Следующие предположения показывают, что действии <2> не оставит неокрашенными слишком много вершин.

Предположение 1: Пусть $C=\{C_o,...,C_{r-1}\}$ – множество цветовых классов раскраски ϕ и $\{\pi_0,...,\pi_{\chi-1}\}$ - множество регулярных разбиений C. Тогда

$$\sum_{j=0}^{\chi-1}\sigma(\pi_j)\le\sum_{i=0}^{r-1}\sigma(N(C_i)).\qquad(*)$$

Доказательство: Множество разбиений $\{\pi_j\},(j=0,...,\chi-1)$, содержит каждую пару цветовых классов только единожды. Следовательно, для каждой окрашенной вершины v, ее вклад в левую часть неравенства равен $\sigma(v)\times$(количество цветовых классов, содержащих вершины, смежные с v). Очевидно, что вклад v в правую часть также велик.

Предположение 2: Пусть ϕ – раскраска вершин, а $C = \{C_o, ..., C_{r-1}\}$ – множество цветовых классов. Если для каждого $i \geq 0$, $\sigma(N(C_i)) \leq (n+m)/\log n$, то существует регулярное разбиение π на C и множество D, такие что:

1. для каждой пары C', C'' цветовых классов, соединенных π, $C' \cup C'' - D$ независимое множество;
2. $\sigma(D) \leq \dfrac{n+m}{2\log n}\dfrac{r}{\chi(r)}$

Доказательство: Если для каждого $i \geq 0$, $\sigma(N(C_i)) \leq (m+n)/\log n$, тогда из (*) следует, что:

$$\sum_{j=0}^{\chi-1} \sigma(\pi_j) \leq \frac{(n+m)}{\log n} r,$$

что, в свою очередь, предполагает существование регулярного разбиения π_k на C, такое что $\sigma(\pi_k) \leq \dfrac{n+m}{\log n}\dfrac{r}{\chi(r)}$.

Пусть $\{(C_{il}, C_{jl})\}$ - множество пар π_k. Для каждого $l = 0,...,\lfloor r/2 \rfloor - 1$ вычислим $\sigma(C_{il} \cap N(C_{jl}))$ и $\sigma(C_{jl} \cap N(C_{il}))$ и определим как D_l множество с наименьшим значением σ. Тогда объединение $D = \bigcup_l D_l$ удовлетворяет условиям 1 и 2.

Теорема. FINDSET выполняет не более O(logn) действий. Если n' и m', соответственно, число вершин и ребер графа G', порожденного множеством неокрашенных вершин, тогда

$$n'+m' \leq \left(1/2 - o(1)\right)(n+m).$$

Доказательство: Если выполнено действие <1>, то общее число удаленных вершин и ребер составляет как минимум (n+m)/logn; таким образом это действие выполняется не более logn раз. Следовательно, число выполнения действия второго типа также не более $\lceil \log n \rceil$ раз.

Пусть D^i - множество вершин, потерявших цвет во время i-ого выполнения второго действия, а $A = \bigcup D^i$. Тогда

$$n'+m' \leq \sigma(A) \leq \sum_i \sigma(D^i),$$

$$\sigma(D^i) \leq \frac{n+m}{2\log n}\frac{r_i}{\chi(r_i)},$$

и

$$\sigma(A) \leq \sum_{i=0}^{\lceil \log n \rceil} \frac{n+m}{2\log n}\frac{r_i}{\chi(r_i)} \leq \frac{n+m}{2\log n}\sum_{i=0}^{\lceil \log n \rceil}\frac{r_i}{\chi(r_i)},$$

где r_i - число цветовых классов непосредственно перед выполнением i-ой операции второго типа.

Чтобы оценить $\sum_{i=0}^{\lceil \log n \rceil}\dfrac{r_i}{\chi(r_i)}$, заметим, что $\chi(r_i) < r_i$ только для четных r_i, и для таких r_i каждое применение действия второго типа уменьшает число цветов вдвое; если $r_i \geq 3$ нечетное, $r_{i+1} \leq (r_i/2) + 1$. Следовательно,

$$\sum_{i=0}^{\lceil \log n \rceil}\frac{r_i}{\chi(r_i)} \leq \sum_{i=1}^{\lceil \log n \rceil}\frac{2^i}{2^i - 1} \leq \lceil \log n \rceil + 2,$$

что подразумевает утверждение теоремы.

Следствие. Любое применение FINDSET возвращает независимое множество I, такое что

$$\left|I \cup N(I)\right| \geq (1/2 - o(1))(m+n).$$

В следующем разделе, мы покажем, что FINDSET может быть реализована, для того чтобы работать в течении времени $O(\log^2 n)$. Это подразумевает, что общее время работы алгоритма – $O(\log^3 n)$.

4. Реализация алгоритма

Сначала мы покажем, как реализовывать FINDSET, чтобы она работала за время $O(\log^2 n)$ на n+m процессорах. Очевидно, что для каждого вызова FINDSET, действие каждого типа выполняется не более $O(\log n)$ раз. Следовательно, нам надо показать, что выполнение каждого действия требует времени только $O(\log n)$.

Граф G представлен списком L=L(G) его вершин и ребер. Каждое ребро входит в этот список дважды, по одному разу на каждое направление. Таким образом, длина L равна n+2m. Каждому элементу L сопоставлен процессор. Процессоры пронумерованы от 1 до n+2m; первые n из них соответствуют вершинам. Каждое ребро знает не только свои конечные вершины, но их цвета, а также расположение своего эквивалента в другом направлении. Почти все операции выполняются сортировкой этого списка, основанной на некотором ключе. Из [AKS, C] нам известно, что возможно отсортировать m записей за время $O(\log n)$ на m процессорах.

Для удаления цветового класса C, каждое ребро проверяет цвет своих вершин и, если хотя бы одна из них имеет цвет C, ребро удаляет себя. Затем FINDSET сортирует список для того, чтобы избавится от удаленных ребер.

Чтобы решить выполнять действие <1> или <2>, FINDSET нужно посчитать $\sigma(C_i)$ для каждого C_i в ϕ. Она может посчитать степень каждой вершины, отсортировав список ребер по их первой вершине. Затем она может отсортировать список вершин по их цвету и просуммировать степени вершин.

Когда HALF выполняет действие <2>, она находит регулярное разбиение π, которое минимизирует $\sigma(\pi)$. Для этого ей необходимо знать степень каждой вершины. Степени можно взять из предыдущего шага или вычислить их тем же методом. Чтобы посчитать $\sigma(\pi)$, HALF вычисляет, для каждого ребра с двумя окрашенными вершинами, index(i, j, r), где i и j – цвета смежных вершин. Затем ребра сортируются по index'у, группируя ребра по первой вершине. Различные первые вершины, которые встречаются с данным индексом q являются вершинами из D^q для разбиения π_q. HALF затем суммирует степени этих вершин, чтобы посчитать веса $\sigma(\pi_t)$ $(t = 0,..., \chi(r) - 1)$, и выбирает разбиение наименьшего веса.

Пусть q – индекс выбранного разбиения π. На этой стадии, HALF рассматривает список F ребер, индекс которых есть q. Для каждой пары (C_{il}, C_{jl}) из π, вершины одного из цветовых классов может потерять цвет или быть перекрашена при смешивании C_{il} и C_{jl}; мы называем этот цветовой класс приемлемым. Чтобы найти приемлемый класс, HALF вычисляет $\sigma_{il} = \sigma(C_{il} \cap N(C_{jl}))$ для каждого $l = 0,...,\lfloor r/2 \rfloor - 1$: множество C_{il} приемлемо, если $\sigma_{il} \leq \sigma_{jl}$. Как только HALF определила приемлемый цветовой класс, она просматривает все ребра и снимает окраску у тех вершин v, принадлежащих приемлемому цветовому классу C_t, которые смежны с вершиной цвета rev(t,q;r). Наконец, пары цветовых классов, определяемых q, смешиваются в один класс перекрашиванием, для каждой пары, всех вершин цветового класса с большим цветом, то есть каждая вершина v с исходным цветом c(v) меняет свой цвет на rev(c(v),q;r) если rev(c(v),q;r) $\leq$ c(v).

Обычно, цветовые классы не обязательно пронумерованы по порядку. Чтобы исправить это, вершины отсортированы по цвету, так что множество используемых цветов можно может быть определено. Далее, HALF перенумеровывает цвета и назначает каждой вершине новый цвет. Чтобы обновить цвета первых вершин, сохраненных в ребрах, список ребер отсортирован по первой вершине. Затем, через указатели на повторное вхождение ребер обновляются цвета вторых вершин.

Каждое выполнение основного цикла FINDSET требует между 1 и 3 сортировками, и время $O(\log n)$ тратится на другую работу. Таким образом, FINDSET требует только $O(\log^2 n)$ времени. Число вызовов FINDSET $O(\log n)$ подразумевает, что временная сложность алгоритма $O(\log^3 n)$.

На данный момент мы приняли, что используется m+n процессоров. Однако, используя технику уменьшения процессоров Миллера и Рейфа [MR], алгоритм может быть выполнен на $(m+n)/\log n$ процессорах, с увеличением временной сложности не более чем на константу. Для случайного k>1, если есть только (n+m)/k процессоров, то каждый реальный процессор может эмулировать k виртуальных процессоров в алгоритме. Так как вызов FINDSET уменьшает вдвое значение n+m графа, число виртуальных процессоров, эмулируемых реальным процессором, уменьшается вдвое после каждого вызова FINDSET. Таким образом, существует константа C>0, такая что для всех i=1,2,…, i-ый вызов FINDSET выполняется за время $\le C2^{-i}\log^3 n$. Это увеличивает время работы алгоритма вдвое.

Размещение виртуальных процессоров по реальным очевидно. Каждый виртуальный процессор отвечает за один элемент множества L вершин и ребер графа. Необходимо только перераспределять виртуальные процессоры после каждого вызова FINDSET. Новое представление $G-(I\cup N(I))$ можно легко найти посредством сортировки. Перераспределение можно выполнять даже после выполнения каждого цикла в FINDSET. Того, что это ускорит алгоритм для типичных графов, а для наихудших графов не сделает его медленнее, достаточно для улучшения асимптотического ускорения алгоритма.

Заметим, что такой же прием можно применить для уменьшения в $\log n$ раз числа процессоров, используемых вероятностными алгоритмами из [L1] и [ABI].

5. Выводы

Важным свойством параллельного алгоритма является общая работа W, которую он выполняет. Очевидно, что $W \le P\times T$, где P – число процессоров, используемых алгоритмом, а T – время выполнения. Таким образом, наш детерминированный алгоритм для MIS выполняет работу $O((m+n)\log^2 n)$, которая в $\log^2 n$ больше работы, выполняемой очевидными последовательными алгоритмами.

Было бы неплохо найти алгоритм, выполняющий меньше работы. Одним из подходов может быть улучшение используемого алгоритма сортировки. Это возможно при условии , что ключи всех сортировок являются малыми целыми. В действительности, у Рейфа есть алгоритм, сортирующий n малых целых чисел, выполняющий работу только $O(n)$ на случайных PRAM с параллельным чтением, параллельной записью [R]. Однако, если в модели представлена рандомизация , предпочтительно использовать алгоритмы из [L1] и [ABI]. Таким образом, реальное улучшение может быть достигнуто если использовать лучший алгоритм сортировки. Очевидно, что это также представляет интерес и для других задач. Другим подходом может быть уменьшение числа сортировок. Оба подхода оказываются очень трудными.

Более обещающий путь улучшения алгоритма – уменьшить время выполнения без чрезмерного (или вообще) увеличения выполняемой работы. Может оказаться возможным уменьшить время выполнения алгоритма параллельным выполнением различных вызовов

FINDSET. Для этого есть несколько путей; сложность представляется в разработке лучшей аналитической техники для оценки времени выполнения.

Двойное представление ребер может также быть полезным для других задач, среди первых кандидатов раскраска ребер и раскраска вершин. Другие потенциальные продуктивные приложения этого представления – задача построения независимого множества гарантированного размера. В [G], был описан алгоритм, выполняющийся за время $O(\log^3 n)$ на $O(n+m)$ процессорах, и конструирующий независимое множество из $\geq n^2/(32m)$ $(m \geq n/2)$ вершин. Предположительно, при подходящем изменении определения веса множества превратит наш алгоритм в алгоритм, строящий независимое множество, содержащее $\geq n^2/(2m+n)$. Это гарантировано теоремой Турана [T] которая также полагает, что эта граница является наилучшей в терминах m и n.