УДК 681.3

С. Д. Погорілий, С. І. Лозицький

Формальні методи розпаралелювання алгоритму Тар'яна

(Представлено членом-кореспондентом НАН України В. В. Скопецьким)

We present a method for optimization of Tarjan's algorithm for the detection of strongly connected components in a direct graph. The approach to its parallel implementation is offered, and the theoretical synthesis of the respective formula of the algorithm is formulated in systems of the modified algorithmic algebras of V. M. Glushkov. The theoretical estimations of increasing the productivity of the algorithm are obtained. These estimations have been checked up and confirmed in the experiment.

Розглянемо орієнтований граф $G(V, E)$, де V — множина вершин і $E \subseteq V \times V$ — множина ребер.

Шляхом з вершини v_0 у вершину v_k в графі $G(V, E)$ називається послідовність вершин та ребер $[v_0\ (v_0,\ v_1),\ v_1\ (v_1,\ v_2),\ \ldots,\ v_{k-1}\ (v_{k-1},\ v_k)]$.

Дві вершини v та w називаються еквівалентними за шляхом, якщо існує шлях з вершини v у вершину w і з вершини w у вершину v.

Граф $G_1(V, E)$ називається сильно зв'язаним, якщо довільні вершини v, w $\in$ V еквівалентні за шляхом. Іншими словами, у сильно зв'язаному графі кожна вершина є досяжною з будь-якої іншої вершини цього графа.

Довільний граф можна розділити на множину компонент, що не перерізаються між собою. Кожна така компонента називається сильно зв'язаною компонентою — SCC (Strongly Connected Component) і являє собою максимальну множину еквівалентних за шляхом вершин вихідного графа (максимальну в тому розумінні, що якщо до цієї множини додати будь-яку іншу вершину з V, то отримана компонента вже не буде сильно зв'язаною). Сильно зв'язана компонента, яка містить лише одну вершину, називається тривіальною сильно зв'язаною компонентою.

Задача розділення графа на множину SCC є однією з основних в теорії графів. Існує декілька класичних алгоритмів для її розв'язання [1, 2], які неодноразово підлягали оптимізації та удосконаленню [3]. Одним з напрямків ефективного розв'язання вищезгаданої задачі є паралельна реалізація відповідних алгоритмів [4, 5]. Нижче запропоновано метод підвищення швидкодії класичного алгоритму Тар'яна для пошуку SCC шляхом формального розпаралелювання.

Формалізація алгоритму Тар'яна. Послідовна регулярна схема алгоритму. Роберт Тар'ян (Robert Tarjan) запропонував алгоритм [6], який знаходить сильні компоненти графа $G(V, E)$ за час $O(|V|+|E|)$, де $|V|$ — кількість вершин і $|E|$ — кількість ребер у вихідному графі. Він складається з двох логічних частин, кожна з яких виконує відповідні дії. Перша частина за допомогою пошуку в глибину (DFS) будує ліс DFS графа. Друга частина алгоритму виконує дії щодо виділення сильної компоненти, а саме:

маркує відповідні вершини як належні до певної компоненти;

видаляє зі стеку вершини знайденої компоненти.

Одним із потужних методів теоретичного дослідження алгоритмів є математичний апарат модифікованих систем алгоритмічних алгебр (САА-М) [7] В. М. Глушкова та заснований на ньому багаторівневий структурний синтез алгоритмів.

Використовуючи САА-М, сформуємо регулярну схему алгоритму (РСА) для алгоритму Тар'яна, викладеного на теоретико-множинному рівні у [6]. Для ясності записів вводяться такі позначення для операторів, а також для даних:

1) N_i — елемент масиву nodes[i];

2) G_{vi} — елемент матриці суміжності g[v][i] (ребро графа);

3) E — порожній оператор.

Відповідна схема для функції VISIT має вигляд

$$\begin{aligned}\mathrm{VISIT(v)} = (\mathrm{N_v.visit} = 1) * (\mathrm{N_v.incomp} = 0) * \mathrm{PUSH(v, SP)} *_{\mathrm{i>N}} \{_{\mathrm{Gvi}} (_{\mathrm{!Nv.visit}} (\mathrm{VISIT(i)VE}) * \\ * _{\mathrm{!Nv.incomp}} ((\mathrm{N_i.root} = \mathrm{MIN(N_v.root, N_i.root)})\mathrm{VE}) * \mathrm{VE}) * \mathrm{i}{+}{+}\} \\ *_{\mathrm{Nv.root==v}} (\{(\mathrm{w} = \mathrm{POP(SP)}) * (\mathrm{N_w.incomp} = 1)\}_{\mathrm{w==v}} \mathrm{VE}) \qquad (1)\end{aligned}$$

(тут і далі знак ! означає заперечення умови).

Отриману РСА умовно можна розбити на дві логічні частини, кожна з яких виконує відповідні дії:

перша частина виконує пошук в глибину по графу та селекцію вершини-кореня для поточної вершини (1 та 2 рядки);

друга частина виконує безпосереднє виділення сильної компоненти шляхом видалення вершин поточної SCC зі стека та встановлення для них маркера належності до компоненти в логічну одиницю (3 рядок).

Метод розпаралелювання алгоритму Тар'яна для деяких типів графів та формування відповідної ПРСА. Алгоритм Тар'яна грунтується на методі обходу графа в глибину (DFS), який в свою чергу реалізований з використанням рекурсивної процедури. Процес рекурсії як такий є природно послідовним з точки зору розпаралелювання і викликає доволі значні труднощі. Проте алгоритм в цілому можна розглядати фактично як алгоритм пошуку в глибину, на який накладено ще додаткові дії (робота зі стеком, селекція кореня для відповідної вершини $\mathrm{N_i.root} = \mathrm{MIN(N_v.root, N_i.root})$, виділення сильної компоненти $_{\mathrm{Nv.root==v}}(\{(\mathrm{w} = \mathrm{POP(SP)}) * (\mathrm{N_w.incomp} = 1)\}_{\mathrm{w==v}}\mathrm{VE})$). Очевидно, що зазначені дії займають значну частину ресурсів під час виконання алгоритму. Розділити ці дії і пошук в глибину та сумістити їх у часі є однією з головних ідей паралельної реалізації, яка буде розглянута нижче.

В роботі пропонується розпаралелювання за даними, що полягає у розбитті графа на підграфи та подальшій обробці кожного з них. Розбити граф довільно неможливо, оскільки може статися ситуація, коли вершини однієї компоненти потраплять в різні частини і існування цієї SCC не буде встановлено взагалі. Тому при розбитті графа слід враховувати систему зв'язків між вершинами і вибрати частини так, щоб виключити вищезгадане розбиття сильної компоненти. На перший погляд ця задача є нетривіальною і досить складною. Проте для певної категорії графів вона вирішується з використанням того ж таки абстрактного пошуку в глибину. Як уже відзначалося, під час виконання останнього формується ліс DFS, що складається з багатьох дерев DFS. Саме ці дерева DFS і є розділенням графа на шукані частини. Для реальних практичних задач, кількість дерев у лісі DFS є значною і, як наслідок, граф може бути розділений на достатню кількість частин, в кожній з яких потрібно виконати виділення сильних компонент.

Запропонований метод має зміст, коли виконується таке твердження: для орієнтованого графа $G(V, E)$ вершини кожної сильної компоненти знаходяться в межах одного дерева DFS. Це твердження доводиться з використанням методу від супротивного.

Припустимо, що дві вершини v та w графа $G(V, E)$ містяться у різних деревах лісу DFS і належать одній сильній компоненті C цього ж графа. Тоді, внаслідок означення сильної компоненти, існують шляхи з вершини v в w та з w у v. Якщо такі шляхи існують, то в процесі пошуку в глибину неодмінно можна досягнути вершини v з w або навпаки в межах одного дерева DFS. Таким чином, ми дійшли протиріччя. Отже, вершини v та w, які належать сильній компоненті C, містяться в одному дереві лісу DFS для графа $G(V, E)$.

Спираючись на доведене твердження, за допомогою DFS можна розділити граф на множину підграфів, що не мають спільних вершин. Далі в кожній з цих частин треба застосувати алгоритм Тар'яна для знаходження сильних компонент. Причому останні дії можуть бути виконані у паралельному варіанті одночасно для декількох підграфів. Таке розпаралелювання за даними є дещо специфічним і застосовується для графів з достатньою кількістю дерев у лісі DFS. Також слід відзначити, що обробку підграфа (дерева DFS) можна починати одразу ж після його знаходження.

Алгоритм рекурсивної процедури звичайного пошуку в глибину для графа $G(V, E)$ формалізовано у такий спосіб:

$$\mathrm{DFS(v)} = (\mathrm{N_v.dfs} = 1) * (\mathrm{dfs[clk++]} = \mathrm{v}) *_{\mathrm{i>N}} \{_{\mathrm{Gvi}} (_{\mathrm{!Nv.dfs}} (\mathrm{DFS(i)VE}) \mathrm{VE}) * \mathrm{i++}\} \qquad (2)$$

вектор dfs[N] відтворює порядок відвідування вершин графа в процесі пошуку в глибину;

глобальна змінна clk є лічильником відвіданих вершин. Початкове значення clk = 0.

Основна гілка програми, яка буде виконувати пошук в глибину на графі, по мірі знаходження дерев DFS повинна передавати інформацію про них задачам, що безпосередньо займатимуться виділенням компонент. Ця інформація являє собою два числа ind1 та ind2, що індексуватимуть у векторі dfs першу та останню вершини (відповідно dfs[ind1] та dfs[ind2]) знайденого дерева DFS. Для забезпечення такого зв'язку між паралельними гілками алгоритму введемо деякий спільний ресурс — спільну область пам'яті. Його найзручніше спроектувати у вигляді черги типу FIFO (First In First Out).

Запропонована черга легко реалізується у вигляді двонаправленого списку (подібно до реалізації стека для алгоритму Тар'яна). Така структура даних є головним каналом обміну даними між задачами і є критичним спільним ресурсом для всіх задач. Тому при роботі з цим об'єктом необхідно забезпечити синхронізацію. В загальному випадку черга повідомлень пов'язана з такими об'єктами:

вміст черги (черга з повідомлень, що являють собою значення змінних ind1 та ind2);

черга задач, які чекають на появу повідомлень;

механізм синхронізації, який забезпечує вза'ємовиключення задач при роботі з чергою повідомлень.

Для роботи з чергою повідомлень визначено дії, які можливо виконувати над нею:

PUSH_FIFO(FIFO, ind1, ind2) — додати повідомлення (ind1 та ind2) в кінець черги. У розглядуваному методі дану операцію викликатиме лише задача, що здійснює розбиття графа на частини;

POP_FIFO(FIFO, &ind1, &ind2) зчитати повідомлення у змінні ind1 та ind2 з черги. Задача, яка викликала дану операцію, може бути поставлена в кінець черги задач, що чекають повідомлення, в тому разі, якщо черга повідомлень порожня; при надходженні

повідомлення в чергу активізуються всі задачі черги очікування і при цьому реалізується механізм взаємовиключення задач при зверненні до черги повідомлень;

KILL(FIFO) знищити чергу повідомлень. Після того як виконана дана операція, черга повідомлень припиняє існування. Всі задачі, які перебували в черзі очікування, завершуються; будь-яка задача, яка викличе одну з двох попередніх операцій, буде завершена.

Враховуючи все викладене вище та використовуючи формулу для рекурсивної процедури пошуку в глибину (2), запишемо формулу для задачі, що виконує пошук в глибину на графі та додає інформацію про знайдені дерева DFS до черги повідомлень:

$$\begin{aligned}\mathrm{GLOB} = {}_{\mathrm{i>N}}\{(\mathrm{ind1} = \mathrm{clk}) * {}_{\mathrm{Gvi}}(\mathrm{DFS(i)VE}) * (\mathrm{ind2} = \mathrm{clk} - 1) * \mathrm{PUSH\_FIFO} \\ (\mathrm{FIFO}, \mathrm{ind1}, \mathrm{ind2}) * \mathrm{i}++\} \end{aligned} \tag{3}$$

Змінні ind1 та ind2 є локальними і видимі лише всередині процедури GLOB.

Тепер приступимо до реалізації формули задач, які будуть обробляти знайдені дерева та знаходити сильні компоненти. Функції такої задачі полягають у тому, щоб зчитувати з черги повідомлень індекси ind1 та ind2 по мірі їх надходження та обробляти за алгоритмом Тар'яна відповідні дерева DFS. Для цього рекурсивну процедуру VISIT(v), яка подана формулою (1), потрібно модифікувати таким чином:

$$\begin{aligned}\mathrm{VISIT[k](v, ind1, ind2)} = (\mathrm{N_v.visit} = 1) * (\mathrm{N_v.incomp} = 0) * \mathrm{PUSH(v, SP[k])} * \\ * {}_{\mathrm{!(ind2=>m>=ind1)}}\{(\mathrm{i} = \mathrm{dfs[m]}) * {}_{\mathrm{Gvi}}({}_{\mathrm{!Nv.visit}}(\mathrm{VISIT[k](i, ind1, ind2)VE}) * \\ * {}_{\mathrm{!Nv.incomp}}((\mathrm{N_i.root} = \mathrm{MIN(N_v.root, N_i.root)})\mathrm{VE})\mathrm{VE}) * \mathrm{i}++\} * \\ * {}_{\mathrm{Nv.root==v}}(\{(\mathrm{w} = \mathrm{POP(SP[k])}) * (\mathrm{N_w.incomp} = 1)\}_{\mathrm{w==v}}\mathrm{VE}) \end{aligned} \tag{4}$$

Головною відмінністю співвідношення (4) формули від послідовної схеми алгоритму є та, що тепер цикл внутрішнього пошуку в глибину виконується не по всьому графу, а лише по певному дереву DFS. Вершини цього дерева розташовані у топологічному порядку у частині вектора dfs, що починається з вершини dfs[ind1] і закінчується вершиною dfs[ind2]. Обробка в такому порядку займає найменший час. Оскільки задач, що використовують формулу (4), буде декілька, то кожна з них повинна мати свій стек. Для цього введено вектор вказівників SP[k] на стеки всіх задач.

Користуючись модифікованою схемою процедури VISIT[k] (4), отримуємо формулу для задачі клієнта:

$$\begin{aligned}\mathrm{TAR[k]} = {}_{0}\{\mathrm{POP\_FIFO(FIFO, \&ind1, \&ind2)} * \\ * {}_{\mathrm{ind2==N-1}}(\mathrm{KILL(FIFO)VE}) * \mathrm{VISIT[k](dfs[ind1], ind1, ind2)}\} \end{aligned} \tag{5}$$

Процедуру реалізовано у вигляді нескінченного циклу. Альтернатива перевіряє повідомлення, і якщо воно останнє (ind2 == N − 1), знищує чергу повідомлень, термінуючи тим самим всі задачі, які перебувають в черзі очікування. Задача, яка викликала KILL(FIFO), завершується під час наступної спроби зчитати повідомлення із вже не існуючої черги повідомлень. Слід зазначити, що задача TAR стає у чергу очікування і не займає ресурсів у тому разі, коли черга повідомлень порожня.

Взаємодія задачі GLOB та низки задач TAR є асинхронною, що формалізується за допомогою асинхронної диз'юнкції ($\overset{\bullet}{\bigvee}$) з сигнатури операцій CAA-M. Таким чином, спираючись

на формули (3) та (5), отримуємо розгорнуту формулу запропонованого методу паралельної реалізації алгоритму Тар'яна:

$$\begin{aligned}
&\mathrm{PAR_TARJAN} = {}_0\{\mathrm{POP_FIFO(FIFO, \&ind1, \&ind2)} *\\
&\quad *{}_{\mathrm{ind2==N-1}}(\mathrm{KILL(FIFO)VE}) * \mathrm{VISIT\ [1](dfs[ind1], ind1, ind2)}\} \bigvee \ldots\\
&\quad \bigvee {}_0\{\mathrm{POP_FIFO(FIFO, \&ind1, \&ind2)}*\\
&\quad *{}_{\mathrm{ind2==N-1}}(\mathrm{KILL(FIFO)VE}) * \mathrm{VISIT[k](dfs[ind1], ind1, ind2)}\} \bigvee\\
&\quad \bigvee {}_{\mathrm{i>N}}\{(\mathrm{ind1 = clk}) *{}_{\mathrm{Gvi}}(\mathrm{DFS(i)VE}) * (\mathrm{ind2 = clk - 1})*\\
&\quad * \mathrm{PUSH_FIFO(FIFO, ind1, ind2)} * \mathrm{i++}\}
\end{aligned} \tag{6}$$

Формули (1) та (6) є еквівалентними в тому розумінні, що забезпечують однаковий результат роботи при ідентичних вхідних даних.

Теоретична оцінка приросту швидкодії. Проведемо теоретичну оцінку приросту швидкодії алгоритму Тар'яна в паралельному виконанні, наведеного формулою (6), порівняно з послідовною схемою (1). Припустимо, що граф

$G(V, E)$ має n однакових дерев лісу пошуку в глибину, кожне з яких містить V/n вершин; густина ребер є сталою в усіх n деревах та визначається як $\rho = En^2/V^2$.

Час виконання в межах одного дерева пошуку в глибину (2), модифікованої (4) та послідовної (1) схем процедури VISIT, лінійно залежить від величини $V + E$. Відповідно для цих часових інтервалів маємо таку оцінку:

$$\tau_1 = \alpha_1\left(\frac{V}{n} + \rho\frac{V^2}{n^2}\right) + \alpha_2, \quad \tau_2 = \beta_1\left(\frac{V}{n} + \rho\frac{V^2}{n^2}\right) + \beta_2, \quad \tau_3 = \gamma_1\left(\frac{V}{n} + \rho\frac{V^2}{n^2}\right) + \gamma_2, \tag{7}$$

де $\alpha_{1,2}$, $\beta_{1,2}$, $\gamma_{1,2}$ — деякі додатні константи. Співвідношення між часами виконання вищезгаданих формул алгоритму зображено на рис. 1, *а*. На рис. 1, *б* наведено часову діаграму виконання алгоритму Тар'яна, що реалізований відповідно за формулами (1) та (6), де припущено, що $\tau_1 < \tau_2 < 2\tau_1$.

У загальному випадку для ефективної реалізації паралельної схеми (6) достатньо k задач TAR[i] (5), якщо $\tau_1 < \tau_2 < k\tau_1$.

Враховуючи все сказане вище, можемо оцінити повний час роботи послідовної (1) та паралельної (6) схем алгоритму Тар'яна:

$$\mathrm{T_{mono}} = n\tau_3 = \gamma_1\left(V + \rho\frac{V^2}{n}\right) + n\gamma_2, \tag{8}$$

$$\mathrm{T_{par}} = n\tau_1 + \tau_2 = \alpha_1\left(V + \rho\frac{V^2}{n}\right) + n\alpha_2 + \beta_1\left(\frac{V}{n} + \rho\frac{V^2}{n^2}\right) + \beta_2. \tag{9}$$

Перевага у часі виконання паралельної схеми над послідовною становить

$$\Delta\mathrm{T} = V^2\left(\gamma_1 - \alpha_1 - \frac{\beta_1}{n}\right)\frac{\rho}{n} + V\left(\gamma_1 - \alpha_1 - \frac{\beta_1}{n}\right) + n(\gamma_1 - \alpha_1) - \beta_2, \tag{10}$$

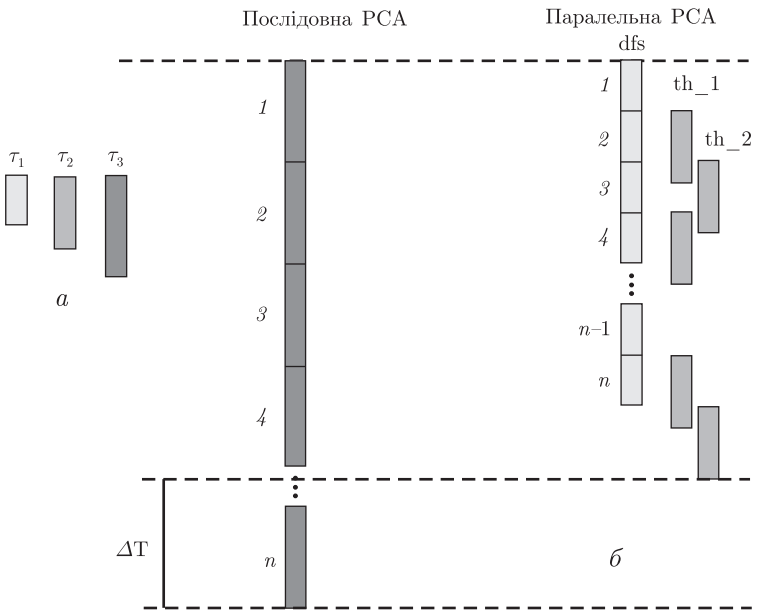

Рис. 1. Співвідношення між часами виконання формул алгоритму (*а*) та часова діаграма виконання алгоритму Тар'яна (*б*)

звідки видно, що при постійній кількості дерев лісу DFS та постійній густині ребер графа множники при степенях V є сталими. Тоді формула для часового приросту набуде вигляду

$$\Delta \mathrm{T} = AV^2 + BV + C. \tag{11}$$

Як бачимо, приріст у швидкодії ПРСА (6) порівняно з послідовною РСА квадратично залежить від V при сталій густині ρ ребер графа. Також з (10) видно, що при $V = \mathrm{const}$ має місце лінійна залежність приросту швидкодії від ρ.

Часові оцінки (8), (9) та (10) було перевірено експериментально на кластері Київського національного університету ім. Тараса Шевченка.

1. *Седжвик Р.* Фундаментальные алгоритмы на C++. Алгоритмы на графах / Пер. с англ. – СПб: ООО "ДиаСофтЮП", 2002. – 496 с.
2. *Aho A. V., Hopcroft J. E., Ullman J. D.* Data structures and algorithms. – Addison-Wesley: Reading, Mass., 1983.
3. *Nuutila E., Soisalon-Soininen T.* On finding the strongly connected components in a directed graph: Laboratory of Information Processing Science, Helsinki University of Technology Otakaari 1, SF – 02150 Espoo, Finland.
4. *Rho Min K., Gonzalez-Gutierrez A.* Finding strongly connected components in parallel: Department of Computer Science, University of California, Santa Barbara, CA, 2006. – 14 c.
5. *Fleischer L. K., Hendrickson B., Pinar A.* On identifying strongly connected components in parallel // Lecture Notes in Computer Science 1800. – 2000. – **505**.
6. *Tarjan R.* Depth first search and linear graphs algorithm // SIAM J. of Computing. – 1972. – No 1(2). – P. 146–160.
7. *Ющенко Е. Л., Цейтлин Г. Е., Грицай В. П., Терзян Т. К.* Многоуровневое структурное проектирование программ. – Москва: Финансы и статистика, 1989.

Київський національний університет ім. Тараса Шевченка

Надійшло до редакції 03.04.2008