УДК 681.3

С. Д. Погорілий, А. Д. Гусаров

Паралельна реалізація алгоритму Едмондса–Карпа

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

Formalization of the Edmonds-Karp algorithm is made using a mathematical apparatus of modified systems of algorithmic algebras, and its sequential scheme is constructed. An approach to paralleling the sequential scheme is suggested, and, as a result, several parallel schemes are obtained. An analysis of the results of scheme modeling is performed by using process-oriented and thread-oriented paradigms.

Група з розвитку Internet IRTF (Internet Research Task Force), яка координує довгострокові дослідницькі проекти за стеком протоколів TCP/IP, займається створенням нових протоколів та пошуком нових, перспективних методів розв'язання задачі маршрутизації для подолання існуючих недоліків на якісно новому рівні.

Алгоритм Едмондса–Карпа [1, 2] розглядається як альтернативний варіант застосування в перспективних протоколах маршрутизації [3, 4]. Існує багато архітектур, що можуть забезпечити підвищення швидкодії цього алгоритму, і важливим класом серед них є паралельні. У зв'язку з цим далі викладено застосування математичного апарату модифікованих систем алгоритмічних алгебр В. М. Глушкова (САА-М) з метою створення схем паралельних версій алгоритму Едмондса–Карпа.

Метою роботи є отримання спектра паралельних регулярних САА-М-схем алгоритму та їх моделювання для аналізу методів підвищення швидкодії паралельного алгоритму.

Формування схеми паралельного алгоритму Едмондса–Карпа.

Використовуються такі позначення: $G = G(V, E)$ — орієнтований зважений граф (графоїд); $V = V(G)$ — множина вершин графа ($|V| = n$); $E = E(G)$ — множина його ребер ($|E| = m$); s(source) — виток; t(target) — сток ($s \neq t$); $c = c(e)$ — пропускна здатність ребра e; $f = f(e)$ — потік через ребро e ($e \in E$); f^0 — повний потік в графі G.

Визначимо базові початкові умови і структури даних:

алгоритм використовує:

1) дві матриці $n \times n$, що характеризують мережу, — матрицю суміжностей (містить ваги ребер) і матрицю потоків (містить потоки через всі ребра);

2) масив, який містить всі вершини графа;

3) FIFO чергу;

вершини пронумеровані від 0 до $n - 1$ в довільному порядку;

кожна вершина графа окрім свого номера має ще дві властивості: dad (містить номер вузла, з якого було досягнуто цю вершину) та vstd (visited, в класичному алгоритмі пошуку в ширину вона може набувати лише два значення: true або false; надалі її тип буде змінено);

звернення до властивостей відбувається так само, як до полів структур у мовах програмування; якщо ім'я вершини використовується без вказування властивості, мається на увазі її номер.

У найстислішому вигляді алгоритм Едмондса–Карпа можна подати у вигляді такої регулярної САА-схеми:

$$\text{Edmonds-Karp} = \underset{\alpha(BFS)}{\{} Update\}, \tag{1}$$

де предикат $\alpha(BFS)$ набуває значення *істина*, якщо в результаті пошуку у залишковій мережі не знайдено шляху із s в t; *Update* — включає в себе необхідні модифікації в матриці потоків і в повному потоці f^0 і може бути поданий схемою

$$Update = (f^0 := f^0 + t.vstd) * (v := t) \underset{[v=s]}{\{} (d := v.dad) *$$

$$* (f(d,v) := f(d,v) + t.vstd) * (f(v,d) := f(v,d) - t.vstd) * (v := d)\}. \tag{2}$$

Відмінною і ключовою частиною алгоритму є пошук шляху (BFS) [5], послідовний алгоритм якого можна подати регулярною інтерпретованою САА-схемою:

$$BFS0.1 = (Q := \{s\}) * Reset\_visited1 * \underset{[Q=\varnothing]\vee[t.vstd]}{\{} (v := 0) * \underset{[v\geqslant n]}{\{} [(q,v) \in E(G)] \wedge$$

$$\wedge [\overline{v.vstd}] \wedge [c(q,v) > f(q,v)](Include(Q,v) \vee \varnothing) * inc(v)\} * Exclude(Q,q)\}, \tag{3}$$

де Q — черга; $\varnothing$ позначає порожню множину, а q завжди вказує на її голову черги; $Reset\_visited1$ — присвоює значення $false$ властивості $vstd$ всіх вершин, окрім s, якій надається значення $true$; Œ — позначає тотожний оператор; $c(q,v)$ — вага ребра (q,v); $f(q,v)$ — потік через відповідне ребро;

$Include(Q,v)$ — додає в кінець черги Q вершину з номером v, а також позначає її як $visited(v.vstd := true)$ і запам'ятовує, що її було досягнуто з вершини $q(v.dad := q)$;

$Exclude(Q,q)$ — вилучає з голови черги вершину.

З урахуванням особливості алгоритму Едмондса-Карпа пропонується модифікувати алгоритм "пошуку в ширину" таким чином:

$$BFS0.2 = (Q := \{s\}) * Reset\_visited2 * \underset{[Q=\varnothing]\vee[t.vstd>0]}{\{} (v := 0) *$$

$$* \underset{[v\geqslant n]}{\{} ([(q,v) \in E(G)] \wedge [v.vstd = 0] \wedge [c(q,v) > f(q,v)]Include2(Q,v) \vee$$

$$\vee \varnothing) * inc(v)\} * Exclude(Q,q)\}, \tag{4}$$

де $Reset\_visited2$ — присвоює значення 0 властивості $vstd$ всіх вершин, окрім s, якій надається максимально можливе значення;

$Include2(Q,v)$ — додає в кінець черги Q вершину v, а також модифікує її властивості: $v.vstd := \min(q.vstd, c(q,v) - f(q,v))$, $v.dad := q$.

Обгрунтування змін очевидне: тепер після знаходження шляху ми одразу маємо максимальне значення, на яке можна збільшити потік, — воно збережене у властивості *visited* стоку $(t.vstd)$.

Критичним місцем алгоритму є пошук шляху, де маємо найбільшу ітераційну вкладеність, тому увагу зосереджено на оптимізації саме цієї частини. Пропонується керуватися ідеєю "паралелізму за даними", щоб розпаралелити BFS. Для цього необхідно розбити дані

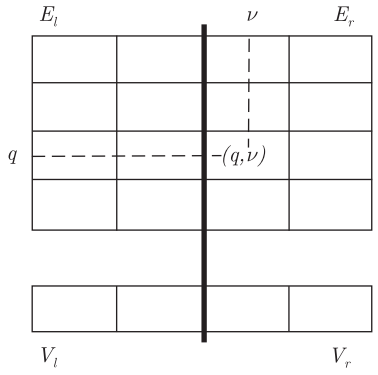

Рис. 1

на неперетинні підмножини, які будуть оброблятися асинхронно [6]. Розіб'ємо внутрішній цикл (по v) на два: $v \in \left[0, \left]\frac{n}{2}\right] - 1\right]$ та $v \in \left[\left]\frac{n}{2}\right], n-1\right]$ (тут $]i]$ позначає цілу частину i). Це відповідає розбиттю множини $E(G)$ на дві підмножини: E_l $\left(\text{там де } v < \left]\frac{n}{2}\right]\right)$ та E_r $\left(\text{там де } v \geqslant \left]\frac{n}{2}\right]\right)$, і множини вершин $V(G)$ відповідно на V_l та V_r (рис. 1). Але ще залишається черга Q, яку повинні модифікувати обидва цикли. Очевидно, щоб уникнути одночасного доступу до черги (що може призвести до невизначеності під час роботи алгоритму), необхідно синхронізувати доступ до неї. Для цього в САА-М існують відповідні засоби — синхронізатори [6], які являють собою α-ітерацію $S(\alpha)\underset{\alpha}{\{}\varnothing\}$, де α — логічна функція, яка залежить від умов синхронізації:

$$BFS1.1 = (Q := \{s\}) * Reset\_visited2 * (\alpha_l := true) * (\alpha_r := true) *$$

$$* \underset{\overline{\alpha_l} \wedge \overline{\alpha_r} \vee [t.vstd > 0]}{\{} S(\alpha_l) * search_l1 * lock * nextq_l * [q_l = \times]((\alpha_l := false) \vee \varnothing) *$$

$$* unlockS(\alpha_r) * search_r1 * lock * nextq_r * [q_r = \times]((\alpha_r := false) \vee \varnothing) * unlock\}. \quad (5)$$

Тут q_l, q_r — локальні в кожному процесі змінні, які містять елемент черги, що обробляється у відповідному потоці (спочатку $q_l = q_r = s$); операції $nextq_l$, $nextq_r$ присвоюють змінним q_l, q_r наступні елементи з черги; якщо наступного елементу немає, то присвоюється значення $\times$. Операції $search_l1$ та $search_r1$ можна подати такими схемами:

$$search_l1 = (v_l := 0) * \underset{v_l in V_l}{\{} ([(q_l, v_l) \in \varnothing_l] \wedge [v_l.vstd = 0] \wedge$$

$$\wedge [c(q_l, v_l) > f(q_l, v_l)](lock * Include(Q, v_l) * (\alpha_r := true) * unlock) * inc(v_l)\}; \quad (6a)$$

$$search_r1 = \left(v_r := \left]\frac{n}{2}\right]\right) * \underset{v_r in V_r}{\{} ([(q_r, v_r) \in \varnothing_r] \wedge [v_r.vstd = 0] \wedge$$

$$\wedge [c(q_r, v_r) > f(q_r, v_r)](lock * Include(Q, v_r) * (\alpha_l := true) * unlock) * inc(v_r)\}. \quad (6б)$$

Операції *lock* та *unlock* реалізують взаємовиключення при роботі з критичною секцією [7], якою є черга. Але таке розпаралелювання далося великою ціною: в тілі найглибшого циклу

(по v_l та, відповідно, по v_r) збільшилася кількість операцій — додалися операції синхронізації при доступі до черги і операція повідомлення іншому процесу про появу в черзі нового елемента ($\alpha_{l,r} := true$). Наступним кроком бажано було б винести всі маніпуляції з чергою поза цей цикл:

$$\begin{aligned} BFS1.2 = {} & (Q := \{s\}) * Reset\_visited2 * (\alpha_l := true) * (\alpha_r := true) * \\ & * \underset{\overline{\alpha_l} \wedge \overline{\alpha_r} \vee t.[vstd>0]}{\{} S(\alpha_l) * search_l2 * lock * [Q_l \neq \varnothing]((Q := Q + Q_l) * nextq_l * \\ & * (\alpha_r := true) \vee nextq_l * [q_l = \times]((\alpha_l := false) \vee \varnothing)) * unlock \\ & \bullet \\ & \vee \\ & S(\alpha_r) * search_r2 * lock * [Q_r \neq \varnothing]((Q := Q + Q_r) * \\ & * nextq_r * (\alpha_l := true) \vee nextq_r * [q_r = \times]((\alpha_r := false) \vee \varnothing)) * unlock\}. \end{aligned} \tag{7}$$

Замість операцій $search_l1$ та $search_r1$ довелося використати їх модифіковані варіанти відповідно $search_l2$ та $search_r2$:

$$search_l2 = (Q_l := \varnothing) * (v_l := 0) * \underset{v_l in V_l}{\{} ([(q_l, v_l) \in E_l] \wedge [v_l.vstd = 0] \wedge$$

$$\wedge [c(q_l, v_l) > f(q_l, v_l)](Include(Q_l, v_l) \vee \varnothing) * inc(v_l)\} \tag{8а}$$

$$search_r2 = (Q_r := \varnothing) * (v_r := 0) * \underset{v_r in V_r}{\{} ([(q_r, v_r) \in E_r] \wedge [v_r.vstd = 0] \wedge$$

$$\wedge [c(q_r, v_r) > f(q_r, v_r)](Include(Q_r, v_r) \vee \varnothing) * inc(v_r)\}. \tag{8б}$$

Ці зміни направлені на розвантаження найглибшого циклу і, як результат, значне зменшення кількості інструкцій процесора при моделюванні схеми.

Аналізуючи запропонований підхід, можна прийти до висновку, що принципово ніщо не заважає зробити більшу кількість паралельних ланок роботи алгоритму. Для цього необхідно, по-перше, розбити дані на необхідну кількість неперетинних підмножин (що відповідає розбиттю циклу по v в схемі (4)), по-друге — забезпечити належну синхронізацію ланок. Якщо взяти кількість ланок p ($p > 1$), то першу вимогу можна задовольнити, розбивши цикл по v на інтервали $v_i \in \left[\left]\frac{n*i}{p}\right], \left]\frac{n*(i+1)}{p}\right] - 1\right]$, де ($i \in [0, p-1]$). Для нормальної синхронізації досить при додаванні в чергу нових вершин повідомляти про це всі інші процеси (рис. 2). Введемо тепер більш зручні короткі позначення, щоб перейти до неінтерпретованої універсальної паралельної регулярної схеми:

$$\begin{aligned} & A \overset{def}{=} (Q := \{s\}) * Reset\_visited2, \\ & B \overset{def}{=} \text{for} k := 0 \text{to} p - 1 \text{do} \alpha_k := true \text{ (сповіщаємо всіх)}, \\ & L \overset{def}{=} lock, \\ & U \overset{def}{=} unlock, \\ & C_i \overset{def}{=} Include(Q, Q_i), \end{aligned} \tag{9}$$

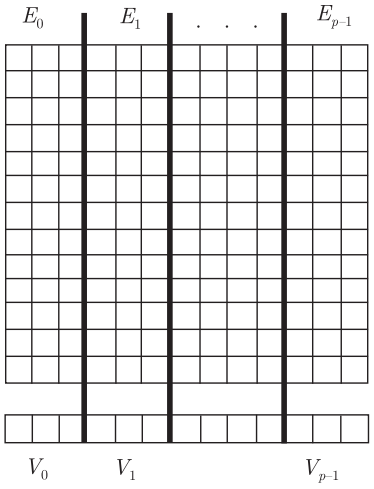

Рис. 2

$D_i \stackrel{def}{=} nextq_i,$

$R(\alpha_i) \stackrel{def}{=} (\alpha_i := false),$

$\beta \stackrel{def}{=} \prod_{k:=0}^{p-1} \overline{\alpha_k},$

$\tau \stackrel{def}{=} [t.vstd > 0],$

$\gamma_i \stackrel{def}{=} [Q_i \neq \varnothing],$

$\phi_i \stackrel{def}{=} [q_i = \times],$

$G_i \stackrel{def}{=} (Q_i := \varnothing) * \left(v_i := \left\rceil \frac{n * i}{p} \right\rceil\right),$

$\chi_i \stackrel{def}{=} \left[v_i \geqslant \left\rceil \frac{n * (i+1)}{p} \right\rceil\right],$

$\varphi_i \stackrel{def}{=} [(q_i, v_i) \in E_i] \wedge [v_i.vstd = 0] \wedge [c(q_i, v_i) > f(q_i, v_i)],$

$I_i \stackrel{def}{=} Include(Q_i, v_i),$

$H_i \stackrel{def}{=} inc(v_i),$

$P \stackrel{def}{=} Update.$

Отже, друга вимога задовольняється операцією (9). З використанням скорочених позначень можна записати паралельну схему для довільної кількості процесів p:

$$
\begin{aligned}
Edmonds\text{-}Karp1.4 = \{A * B * \underset{\beta\vee\tau}{\{} S(\alpha_0) * G_0 \underset{\chi_0}{\{} [\varphi_0](I_0 \vee Œ) * H_0\} * \\
* L * [\gamma_0](C_0 * D_0 * B(\alpha_0) \vee D_0 * [\phi_0](R(\alpha_0) \vee Œ)) * U \\
\overset{\bullet}{\vee} \ldots \overset{\bullet}{\vee} \\
S(\alpha_i) * Gi * \underset{\chi_i}{\{} [\phi_i](Ii \vee Œ) * Hi\} * L * [\gamma_i](Ci * Di * B \vee Di * [\phi_i](R(\alpha_i) \vee Œ)) * U \qquad (10) \\
\overset{\bullet}{\vee} \ldots \overset{\bullet}{\vee} \\
S(\alpha_{p-1}) * G_{p-1} * \underset{\chi_{p-1}}{\{} [\varphi_{p-1}](I_{p-1} \vee Œ) * H_{p-1}\} * \\
* L * [\gamma_{p-1}](C_{p-1} * D_{p-1} * B \vee D_{p-1} * [\phi_{p-1}](R(\alpha_{p-1}) \vee Œ)) * U\}P.
\end{aligned}
$$

Співвідношення (10) було використано як основну схему для моделювання паралельного алгоритму Едмондса–Карпа. Програмну реалізацію схеми виконано мовою C під операційною системою RedHat Linux з використанням парадигм процесів і потоків [7]. Оскільки і потоки, і процеси передбачають роботу на машинах із спільною пам'яттю, то моделювання проводилося на двопроцесорних нодах кластера Київського національного університету [8].

1. *Ford L. R., Fulkerson Jr., Fulkerson D. R.* Maximal flow through a network // Canadian J. of Math. – 1956. – **8**. – P. 399–404.
2. *Edmonds J., Karp R. M.* Theoretical improvements in algorithmic efficiency for network flow problems // J. Assoc. Comput Mach. – 1972. – **19**. – P. 248–264.
3. *http://algolist.manual.ru/maths/graphs/maxflows.*
4. *Погорілий С. Д., Камардіна О. О.* Системи алгоритмічних алгебр. Прикладний аспект // Пробл. Программирования. – 2006. – № 1–2. – С. 393–401.
5. *Сэджвик Р.* Фундаментальные алгоритмы на C++. Ч. 5: Алгоритмы на графах. – DiaSoft, 2002. – 496 с.
6. *Ющенко Е. Л., Цейтлин Г. Е., Грицай В. П., Терзян Т. К.* Многоуровневое структурное проектирование программ. – Москва: Финансы и статистика, 1989. – 208 с.
7. *Багачев К. Ю.* Основы параллельного программирования. – Москва: БИНОМ. Лаборатория знаний. – 2003. – 342 с.
8. *http://www.cluster.univ.kiev.ua.*

Київський національний університет ім. Тараса Шевченка

Надійшло до редакції 03.04.2008