УДК 681.3

С.Д. ПОГОРІЛИЙ, В.А. МАР'ЯНОВСЬКИЙ, Ю.В. БОЙКО, О.А. ВЕРЕЩИНСЬКИЙ

ДОСЛІДЖЕННЯ ПАРАЛЕЛЬНИХ СХЕМ АЛГОРИТМУ ДАНЦИГА ДЛЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ ЗІ СПІЛЬНОЮ ПАМ'ЯТТЮ

Abstract: *Formalization of Dantzig algorithm for the shortest ways search in connected oriented graph using mathematical means of V.M. Glushkov modified systems of algorithmic algebras is performed. Conceptions of paralleling for architectures with shared memory, which are based on minimization of loss on synchronization and parallel data proceeding are proposed. Transformation of algorithm is performed, set of parallel schemes are obtained and their comparative analysis is performed.*
Key words: *routing, Dantzig algorithm, modified systems of algorithmic algebras, parallel regular scheme, paralleling.*

Анотація: *Виконано формалізацію алгоритму Данцига пошуку найкоротших шляхів у зв'язному орієнтованому графі з використанням математичного апарата модифікованих систем алгоритмічних алгебр В.М. Глушкова. Запропоновано концепції розпаралелювання для архітектур зі спільною пам'яттю, що ґрунтуються на мінімізації витрат на синхронізацію та паралельну обробку даних. Проведено трансформацію алгоритму, отримано набір паралельних схем та виконано їх порівняльний аналіз.*
Ключові слова: *маршрутизація, алгоритм Данцига, системи алгоритмічних алгебр, паралельна регулярна схема алгоритму, розпаралелювання.*

Аннотация: *Выполнена формализация алгоритма Данцига поиска кратчайших путей в связном ориентированном графе с использованием математического аппарата модифицированных систем алгоритмических алгебр В.М. Глушкова. Предложены концепции распараллеливания для архитектур с общей памятью, которые основаны на минимизации потерь на синхронизацию и параллельную обработку данных. Проведена трансформация алгоритма, получен набор параллельных схем и выполнен их сравнительный анализ.*
Ключевые слова: *маршрутизация, алгоритм Данцига, системы алгоритмических алгебр, параллельная регулярная схема алгоритма, распараллеливание.*

1. Вступ

Однією із найважливіших і найактуальніших у сьогоденні областей застосування задачі пошуку найкоротших шляхів у ґрафах є маршрутизація пакетів у комп'ютерних мережах. Деякі з перших INTERNET-протоколів маршрутизації, наприклад, RIP (Routing Information Protocol) [1], використовували так званий вектор відстаней. Така схема роботи гарантувала збіжність оцінених маршрутизаторами відстаней до істинних по проходженні певного часу, надаючи, таким чином, інформацію про найкращі шляхи. Для подолання певних проблем, пов'язаних з цим алгоритмом, була розроблена маршрутизація з урахуванням станів каналів зв'язку (Link-State Routing). При такій маршрутизації кожен маршрутизатор підтримує інформацію про ґрафове представлення топології всієї мережі і обчислює свою таблицю маршрутизації, використовуючи алгоритм пошуку всіх найкоротших шляхів у ґрафі. Недоліком такого підходу є додаткові витрати пам'яті та часу на обчислення, але, не зважаючи на це, такий вид маршрутизації визнано ефективним і формалізовано у вигляді протоколу маршрутизації з визначенням найкоротшого маршруту (Open Shortiest Path First protocol, OSPF) [1].

Стрімке зростання досліджень у галузі комп'ютерних мереж, виникнення нових архітектур сучасних мікропроцесорів дозволяють розв'язати задачу маршрутизації на якісно новому рівні. У статті формуються паралельні схеми алгоритму Данцига для застосування у маршрутизаторах нового покоління.

2. Системи зі спільною пам'яттю

Багатопроцесорні та багатоядерні системи на сьогоднішній день належать до класу систем, здатних обробляти великі обсяги інформації в режимі реального масштабу часу. Одним із можливих

ISSN 1028-9763. Математичні машини і системи, 2009, № 4

варіантів архітектури є системи зі спільною пам'яттю (Symetric Multiprocessing, SMP) [2]. Основна ідея такої архітектури полягає у наявності спільної ділянки пам'яті для кількох процесорів. Вона використовується для обміну інформацією, і всі процесори здійснюють однакову адресацію для всіх комірок пам'яті. Саме тому така система називається симетричною.

Однією із найбільших переваг SMP є простота та універсальність у програмуванні. Звичайно використовують модель кількох паралельних гілок (проте, зауважимо, ніщо не заважає організувати міжпроцесорний обмін даними). Створюючи процеси (потоки) і плануючи їх роботу, можна досягти значного виграшу в часі роботи. Уся підсистема працює під керуванням єдиної операційної системи. Зокрема, у Windows підтримка SMP забезпечується, починаючи з версії NT 4.0 [3].

Типовим прикладом SMP є сучасні багатоядерні процесори фірми Intel (Core Duo, Core 2 Duo, Core 2 Quad тощо). Використання цих процесорів в універсальних обчислювальних системах, де ступінь розпаралелювання є невеликим, дозволяє одержати суттєве підвищення коефіцієнта продуктивності.

Таким чином, використання систем SMP дозволяє значно оптимізувати виконання обчислень, що дозволяє успішно використовувати їх для розв'язку задач маршрутизації, особливо при великих навантаженнях на локальні маршрутизатори.

3. Алгоритм Данцига

Опис алгоритму. Одним із найефективніших методів пошуку всіх найкоротших шляхів у зваженому орієнтованому ґрафі є алгоритм Данцига [4, 5].

Розглянемо схему роботи алгоритму. Ідея полягає в послідовному обчисленні за допомогою рекурентної процедури підматриць найкоротших шляхів D_m зростаючої розмірності $m \times m$. Кожна така матриця, фактично, є матрицею найкоротших шляхів підґрафа з вершинами від 0 до $m-1$.

Введемо низку позначень:

N – розмірність ґрафа, який відзеркалює топологію мережі; (1)

$\{i..k\}$ – множина цілих чисел від i до k;

g_{ij} – довжина ребра з вершини i в j;

d_{ij} – довжина найкоротшого знайденого шляху з i в j;

d_{ij}^m – шлях з вершини i до j через вершину m;

$isEdge(i, j)$ – предикат, значенням якого буде істина, якщо існує ребро з вершини i в j;

$isWay(i, j)$ – предикат, значенням якого буде істина, якщо існує шлях з вершини i в j;

$edgeNum(i, j)$ – кількість ребер на шляху з i в j;

$setWay(i, j, m)$ – встановлення шляху d_{ij}^m.

Кожну ітерацію алгоритму можна розділити на три послідовні кроки:

1. Пошук найкоротших шляхів з вершини m до $j < m$.

Шлях можна розбити на дві частини: від m до деякої проміжної вершини i (g_{mi}) та від i до j (d_{ij}).

Тоді

$$d_{mj} = \min_{i\in\{0,m-1\}}\left(g_{mi} + d_{ij}\right). \tag{2}$$

2. Пошук найкоротших шляхів з вершин $i < m$ до m.

Шлях можна розбити на дві частини: від i до деякої проміжної вершини j (d_{ij}) та від j до m (g_{jm}).

Тоді

$$d_{im} = \min_{j\in\{0,m-1\}}\left(d_{ij} + g_{jm}\right). \tag{3}$$

3. Пошук найкоротших шляхів з вершин $i < m$ до $j < m$ з урахуванням наявності нової вершини m.

Шлях можна розбити на дві частини: від i до вершини m (d_{im}) та від m до j (d_{mj}).

Тоді

$$d_{im} = \min\left(d_{im} + d_{mj}, d_{ij}\right). \tag{4}$$

Останній крок рекурентної процедури дає матрицю D_N, яка, власне, і буде матрицею найкоротших шляхів ґрафа.

Послідовна схема алгоритму. Формалізуємо алгоритм Данцига шляхом формування схеми з використанням апарату систем алгоритмічних алгебр Глушкова(САА) [6–8]. Використовуючи позначення (1), сформуємо ряд кон'юнктивних умов для скорочення запису формули алгоритму:

$$\alpha_1(m,i,j) = isEdge(m,i) \wedge isWay(i,j), \tag{5а}$$

$$\alpha_2(m,i,j) = isEdge(j,m) \wedge isWay(i,j), \tag{5б}$$

$$\alpha_3(m,i,j) = isWay(i,m) \wedge isWay(m,j). \tag{5в}$$

Умови α_1, α_2, α_3 вказують на існування відповідно шляхів d^i_{mj}, d^j_{im}, d^m_{ij}.

Використовуючи (5), запишемо умови оптимальності шляхів d^i_{mj}, d^j_{im}, d^m_{ij} на відповідному кроці ітераційного процесу:

$$\begin{aligned}\beta_1 = &\left(\overline{isWay(m,j)}\right) \vee \left(g_{mi} + d_{ij} < d_{mj}\right) \vee \\ &\vee \left((g_{mi} + d_{ij} = d_{mj}) \wedge (edgeNum(i,j) + 1 < edgeNum(m,j))\right),\end{aligned} \tag{6а}$$

$$\begin{aligned}\beta_2 = &\left(\overline{isWay(i,m)}\right) \vee \left(g_{jm} + d_{ij} < d_{mi}\right) \vee \\ &\vee \left((g_{jm} + d_{ij} = d_{mi}) \wedge (edgeNum(i,j) + 1 < edgeNum(i,m))\right),\end{aligned} \tag{6б}$$

$$\begin{aligned}\beta_3 = &\left(\overline{isWay(i,j)}\right) \vee \left(d_{im} + d_{mj} < d_{ij}\right) \vee \\ &\vee \left((d_{im} + d_{mj} = d_{ij}) \wedge (edgeNum(i,m) + edgeNum(m,j) < edgeNum(i,j))\right).\end{aligned} \tag{6в}$$

Під оптимальністю розуміється менша загальна довжина шляху або, якщо довжини порівнюваних шляхів рівні, менша кількість ребер, які необхідно пройти.

Оперуючи умовами (5) та (6), запишемо кожен з кроків алгоритму у вигляді формули:

1. $SubDanc1(m) = \underset{j\notin\{1..m\}}{\{} \; \underset{i\notin\{1..m\}}{\{} \; (\underset{\alpha_1}{} \; (\underset{\beta_1}{} \; setWay(m,i,j) \vee E) \vee E) \times (++i)\} \times (++j)\}.$ (7а)

2. $SubDanc2(m) = \underset{i\notin\{1..m\}}{\{} \; \underset{j\notin\{1..m\}}{\{} \; (\underset{\alpha_2}{} \; (\underset{\beta_2}{} \; setWay(i,m,j) \vee E) \vee E) \times (++j)\} \times (++i)\}.$ (7б)

3. $SubDanc3(m) = \underset{i\notin\{1..m\}}{\{} \; \underset{j\notin\{1..m\}}{\{} \; (\underset{\alpha_3}{} \; (\underset{\beta_3}{} \; setWay(i,j,m) \vee E) \vee E) \times (++j)\} \times (++i)\},$ (7в)

де операція $(++)$ означає інкрементацію змінної на одиницю.

Використовуючи позначення, можна сформувати послідовну регулярну схему(ПРС) алгоритму Данциґа:

$$Dancig = \underset{m>N}{\{} \; SubDanc1(m) \times SubDanc2(m) \times SubDanc3(m) \times (++m)\}. \quad (8)$$

Принцип обробки даних. У роботі виконується парадигма розпаралелювання алгоритму за даними, тобто кожен потік обробляє певну частину інформаційної множини, не перетинаючись на цьому етапі з іншими робочими потоками. Таким чином, для подальшої роботи необхідно сформулювати деякі засади роботи алгоритму із даними, розміщеними в пам'яті.

Інформація про ґраф зберігається у вигляді двох матриць. Перша з них – матриця суміжностей ґрафа. Вона являє собою набір вхідних даних для алгоритму і є незмінною до кінця його роботи (доступ лише на читання). Друга матриця – матриця маршрутів. У комірці з координатами (i, j) зберігається номер вершини, до якої необхідно перейти після i-ої на шляху до j. Таким чином, шлях розбивається на 2 частини: з i до деякої проміжної вершини k та з k до j. Якщо $k = j$, необхідно пройти просто по відповідному ребру між вершинами. За допомогою цієї матриці в динамічному режимі розраховуються довжини найкоротших шляхів між довільними парами вершин. В остаточному варіанті одержана матриця і є результатом роботи алгоритму.

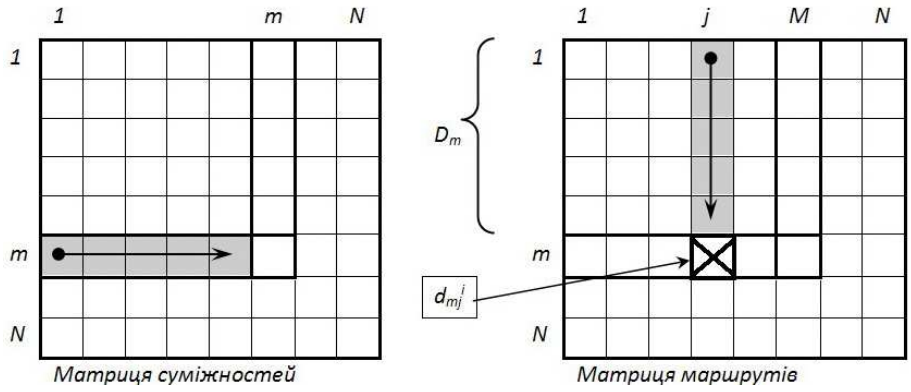
Як зазначено, алгоритм Данцига можна розбити на три незалежні кроки:

1. Пошук найкоротшого шляху d_{mj}^{i}.

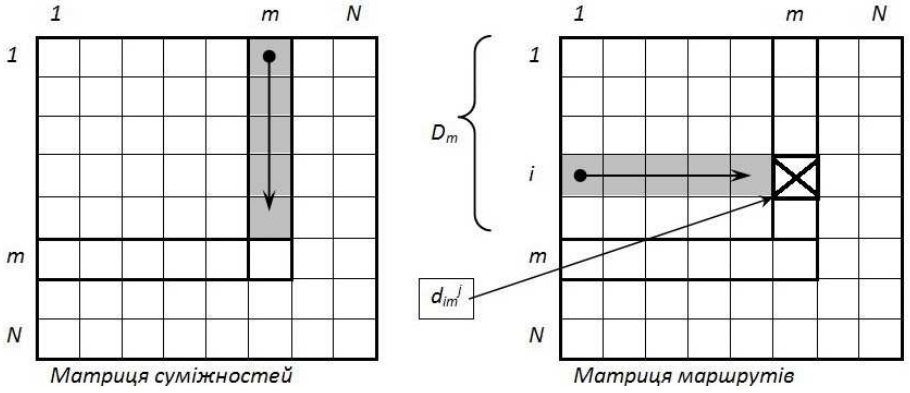
2. Пошук найкоротшого шляху d_{im}^{j}.

3. Пошук найкоротшого шляху d_{ij}^{m}.

Важливо відзначити, що крок 3 може бути виконаний лише після повного завершення двох попередніх. Перші два етапи схематично зображені на рис. 1а та рис. 1б. Як бачимо, для обрахунку кожного нового елемента d_{mj}^{i} (d_{im}^{j}) слід опрацювати цілий стовпчик (рядок) підматриці D_m (стрілками позначено дані, які слід прочитати для знаходження відповідного елемента). Отже, при плануванні роботи кількох потоків оптимальним буде саме таке розбиття вхідних даних, що дозволить уникнути зайвого обміну інформацією між потоками, а також зменшити затрати на синхронізацію.



а)

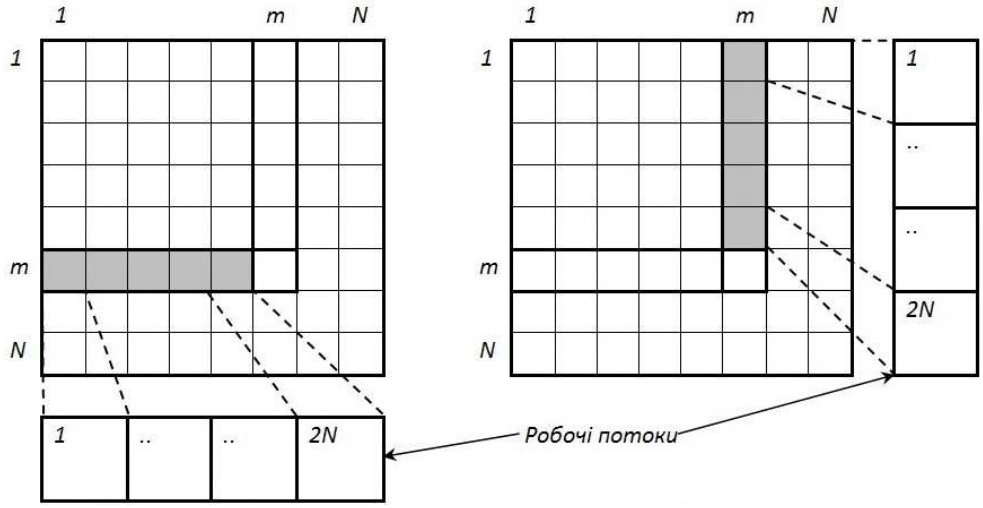


б)

Рис. 1. Обробка даних на перших двох етапах алгоритму

Далі пропонується така схема роботи.

Нехай маємо $2N$ робочих потоків. На першому етапі роботи вони рівномірно поділяють матрицю на рядки (рис. 2а), а на другому – на стовпчики (рис. 2б). При цьому області даних, що обробляються різними потоками, не перекриваються навіть при читанні, що теж може дещо уповільнити роботу алгоритму.



а) б)

Рис. 2. Схема роботи алгоритму на перших двох етапах

Другий крок виконується лише після повного завершення першого, що змушує вводити додаткову точку синхронізації. Для усунення цього доцільно дещо вдосконалити схему, зображену

на рис. 2. Розіб'ємо робочі потоки на 2 групи по N потоків. При цьому перша група виконує крок 1, а друга, відповідно, крок 2. При цьому кількість інформації, яка припадає на один потік, зростає (рис. 3).

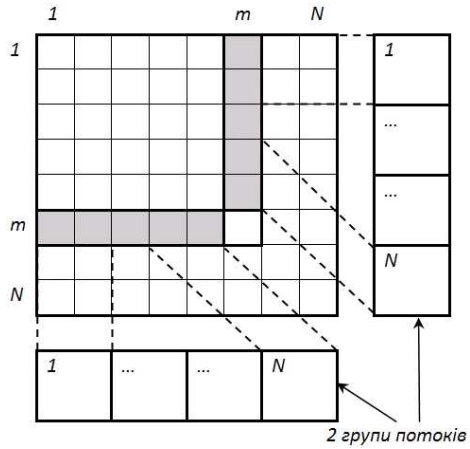


Рис. 3. Альтернативна схема роботи

На третьому кроці, на кожній ітерації циклу, порівнюються лише 2 елементи матриці. Тому в даному випадку розбиття може бути майже довільним, наприклад, на квадратні підматриці.

Але для уникнення затримок, внаслідок одночасного доступу різних потоків до одних і тих же даних, оптимальним буде розбиття на рядки, як показано на рис. 2а.

4. Формування паралельної схеми алгоритму Данцига

Розбиття матриці даних. Введемо деякі нові поняття, які знадобляться при реалізації розпаралелювання алгоритму.

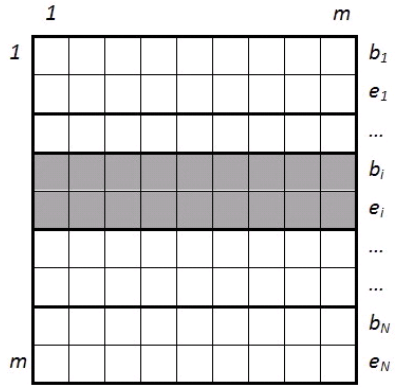


Рис. 4. Розбиття матриці даних

Нехай потрібно розділити матрицю $m \times m$ на N приблизно рівних частин прямокутної форми, як зображено на рис. 4 [9].

b_i позначає перший рядок i-ї частини, а e_i – останній. Їх можна обчислити за такими співвідношеннями:

$$b_i^{m,N} = (i-1) \times]m / N[+ 1, \tag{9}$$

$$e_i^{m,N} = i \times]m / N[. \tag{10}$$

Таким чином, отримано межі областей, які визначатимуть область пам'яті, що оброблятиметься одним із паралельних потоків (зрозуміло, в аналогічний спосіб можна розбити матрицю на стовпчики).

Синхронізація потоків. Введемо узагальнену точку синхронізації кількох паралельних потоків – фактично, бар'єр який затримує виконання потоків, поки їх не набереться достатня кількість. Такий об'єкт можна представити формулою

$$B_N(i) = T(\lambda_i) \times S(\lambda_1) \times ... \times S(\lambda_{i-1}) \times S(\lambda_{i+1}) \times ... \times S(\lambda_N) = T(\lambda_i) \times \left(\prod_{\substack{j=1 \\ j \neq i}}^{N} S(\lambda_j) \right), \tag{11}$$

де $T(\lambda)$ – контрольна точка;

$S(\lambda)$ – точка синхронізації (може інтерпретуватися як порожній цикл);

N – кількість потоків;

i – номер потока, який синхронізується.

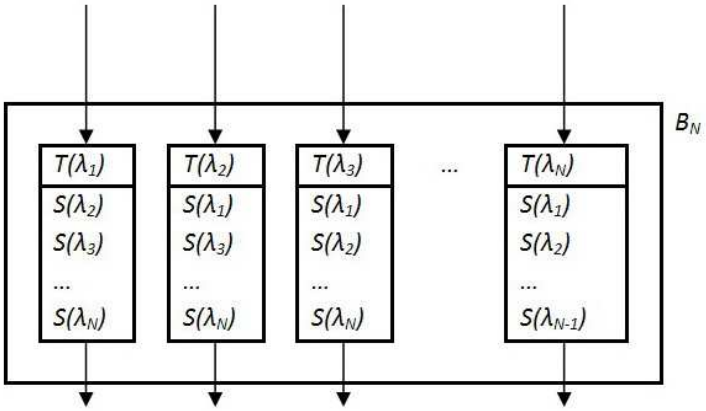


Рис. 5. Внутрішня організація бар'єра

Таким чином, кожен потік для проходження бар'єра повинен подолати $(N-1)$ точку синхронізації, перед цим попередньо розблокувавши свою власну. Схематично роботу бар'єра зображено на рис. 5. Коли будь-який з потоків потрапляє у бар'єр, він не зможе подолати його, поки кожен з решти потоків не розблокує власну контрольну точку.

Еквівалентні перетворення схеми алгоритму. Застосуємо апарат тотожніх перетворень САА-М до схеми (8) [8].

Спочатку розв'яжемо деяку узагальнену задачу. Нехай маємо α-ітерацію (вважатимемо, що на початку ітераційного процесу змінна i ініціалізується одиницею, а на кожному кроці циклу неявно інкрементується на одиницю):

$$I = \underset{i \notin \{1..N\}}{\{\quad} A\}, \tag{12}$$

де оператор A опрацьовує деяку частину вхідних даних. Введемо позначення:

$$\gamma_j = i \notin \{b_j^{k,N}..e_j^{k,N}\}, \tag{13}$$

$$\gamma = i \notin \{1..N\} = \bigwedge_{j=1}^{k} \gamma_j .$$

Таким чином, ми розбили інтервал $\{1..N\}$ на k приблизно однакових частин. Внесемо умову γ всередину ітерації у вигляді фільтра:

$$I = \{\underset{\gamma}{\overline{\gamma}} A\} = \{\underset{\gamma}{\overline{\gamma_1 \wedge \gamma_2 \wedge ... \wedge \gamma_k}} \times A\} = \{\underset{\gamma}{\overline{\gamma_1} \vee \overline{\gamma_2} \vee ... \vee \overline{\gamma_k}} \times A\}. \tag{14}$$

Скористаємося властивістю фільтрації для локально замкнутих умов γ_i для розділення фільтра, після чого винесемо оператор диз'юнкції за ітераційні дужки:

$$I = \{\underset{\gamma}{(\overline{\gamma_1} \vee \overline{\gamma_2} \vee ... \vee \overline{\gamma_k}) \times A}\} = \{\underset{\gamma}{\overline{\gamma_1} A \vee \overline{\gamma_2} A \vee ... \vee \overline{\gamma_k} A}\} = \bigvee_{l=1}^{k} \left(\{\underset{\gamma}{\overline{\gamma_l} A}\} \right). \tag{15}$$

Оскільки в кожний довільний момент часу може бути виконана лише одна з умов типу $i \in \{b_j^{k,N}..e_j^{k,N}\}$, то можна сказати, що $\forall i, j$:

$$\overline{\gamma_i} \wedge \overline{\gamma_j} = \delta_{ij} \wedge \overline{\gamma_i} . \tag{16}$$

Як бачимо з (16), істинність будь-якої з умов $\overline{\gamma_i}$ виключає можливість істинності будь-якої $\overline{\gamma_j}$ при $i \neq j$, що фактично є умовою локальної замкнутості. Тому можемо замінити умову ітерації для кожної з гілок на відповідний фільтр:

$$I = \bigvee_{l=1}^{k} \left(\{ A\} \right). \tag{17}$$

Таким чином, цикл було розбито на k паралельних гілок за допомогою синхронної диз'юнкції. Зауважимо, що кожна гілка працює лише зі своєю частиною інформаційної множини, недоступною для інших. За означенням це власне і є асинхронна диз'юнкція:

$$I = \dot{\bigvee}_{l=1}^{k} \left(\{ A\} \right). \tag{18}$$

Введемо об'єкт для синхронізації гілок:

$$\dot{\bigvee}_{l=1}^{k} \left(\{ A\} \right) = \dot{\bigvee}_{l=1}^{k} \left(\{ A\} \times (T(\lambda_l) \times S(\lambda_l) \times ... \times S(\lambda_{l-1}) \times S(\lambda_{l+1}) . \times .. \times S(\lambda_l)) \right) = \dot{\bigvee}_{l=1}^{k} \left(\{ A\} \times B_N(l) \right). \tag{19}$$

Формула (19) являє собою асинхронну α-ітерацію обробки даних, яка виконується N гілками, які, у свою чергу, синхронізуються після виконання операцій. Застосуємо її до послідовної регулярної схеми (8). Це дозволить розбити кожен із трьох внутрішніх циклів на паралельні гілки:

$$SubDanc1(m) = \dot{\bigvee}_{l=1}^{2N} \left(\{ \underset{j \notin \{b_l..e_l\}}{} \{ \underset{i \in \{1..m\}}{} (\underset{\alpha_1}{} (\underset{\beta_1}{} setWay(m,i,j) \vee E) \vee E) \times (++i)\} \times (++j)\} \times B_{2N}(l) \right) =$$
$$= \dot{\bigvee}_{l=1}^{2N} \left(PSub1(m,2N,l) \times B_{2N}(l) \right). \tag{20}$$

$$SubDanc2(m) = \dot{\bigvee}_{l=1}^{2N} \left(\{ \underset{i \notin \{b_l..e_l\}}{} \{ \underset{j \in \{1..m\}}{} (\underset{\alpha_2}{} (\underset{\beta_2}{} setWay(i,m,j) \vee E) \vee E) \times (++j)\} \times (++i)\} \times B_{2N}(l) \right) =$$
$$= \dot{\bigvee}_{l=1}^{2N} \left(PSub2(m,2N,l) \times B_{2N}(l) \right). \tag{21}$$

$$SubDanc3(m) = \dot{\bigvee}_{l=1}^{2N} \left(\{ \underset{i \notin \{b_l..e_l\}}{} \{ \underset{j \in \{1..m\}}{} (\underset{\alpha_3}{} (\underset{\beta_3}{} setWay(i,j,m) \vee E) \vee E) \times (++j)\} \times (++i)\} \times B_{2N}(l) \right) =$$
$$= \dot{\bigvee}_{l=1}^{2N} \left(PSub3(m,2N,l) \times B_{2N}(l) \right). \tag{22}$$

Підставляючи вирази (20), (21) та (22) у (8), отримаємо

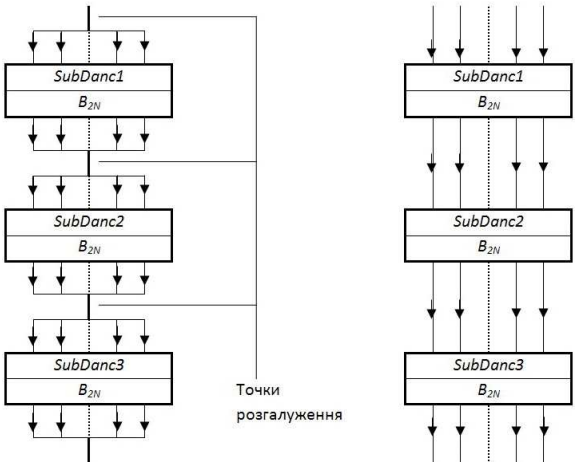
$$Dancig = \underset{m>N}{} \{ \left(\dot{\bigvee}_{l=1}^{2N} \left(PSub1(m,2N,l) \times B_{2N}(l) \right) \right) \times \left(\dot{\bigvee}_{l=1}^{2N} \left(PSub2(m,2N,l) \times B_{2N}(l) \right) \right) \times$$
$$\times \left(\dot{\bigvee}_{l=1}^{2N} \left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m)\}. \tag{23}$$

Отримана формула (23) – паралельна схема алгоритму Данцига, яка представлена на рис. 3. Як бачимо, всередині зовнішньої α-ітерації для виконання кожного з трьох послідовних кроків відбувається розщеплення на паралельні гілки. Для покращання ефективності доцільно винести асинхронну диз'юнкцію за межі ітераційного процесу:

$$Dancig = \{ \underset{m>N}{} \dot{\bigvee}_{l=1}^{2N} \left(PSub1(m,2N,l) \times B_{2N}(l) \times PSub2(m,2N,l) \times B_{2N}(l) \times PSub3(m,2N,l) \times B_{2N}(l) \right) \times$$

$$\times (++m)\} = \dot{\bigvee}_{l=1}^{2N} \{ \underset{m>N}{} PSub1(m,2N,l) \times B_{2N}(l) \times PSub2(m,2N,l) \times B_{2N}(l) \times PSub3(m,2N,l) \times \quad (24)$$

$$\times B_{2N}(l) \times (++m)\}.$$



а) б)

Рис. 6. Варіанти розпаралелювання алгоритму

Схеми (23) та (24) можна інтерпретувати, як зображено на рис. 6а та рис. 6б, де умовно зображено одну ітерацію багатопоточної обробки даних для кожного випадку.

Використаємо (23) для отримання схеми, зображеної на рис. 4.

Поділимо робочі потоки на дві групи: перша буде виконувати крок 1 алгоритму, а друга – крок 2. Введемо умову ω, істинність якої означає належність до першої групи, хибність – до другої. Запишемо модифіковану схему (23):

$$Dancig = \{ \underset{m>N}{} \left(\dot{\bigvee}_{l=1}^{N} \left(PSub1(m,N,l) \times B_N^{(1)}(l) \right) \right) \times \left(\dot{\bigvee}_{l=1}^{N} \left(PSub2(m,N,l) \times B_N^{(2)}(l) \right) \right) \times$$

$$\times \left(\dot{\bigvee}_{l=1}^{2N} \left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m)\}. \quad (25)$$

Введемо фіктивні гілки для вирівнювання кількості потоків та зведемо перші два кроки до спільної диз'юнкції:

$$Dancig = \{ \underset{m>N}{} \left(\dot{\bigvee}_{l=1}^{2N} \left(\underset{\omega}{} (PSub1(m,N,l) \times B_N^{(1)}(l) \vee E) \right) \right) \times \left(\dot{\bigvee}_{l=1}^{2N} \left(\underset{\omega}{} (PSub2(m,N,l) \times B_N^{(2)}(l) \vee E) \right) \right) \times$$

$$\times \left(\dot{\bigvee}_{l=1}^{2N} \left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m)\} =$$

$$= \{ \underset{m>N}{} \left(\dot{\bigvee}_{l=1}^{2N} \left(\underset{\omega}{} (PSub1(m,N,l) \times B_N^{(1)}(l) \vee E) \times \underset{\omega}{} (PSub2(m,N,l) \times B_N^{(2)}(l) \vee E) \right) \right) \times \quad (26)$$

$$\times \left(\dot{\bigvee}_{l=1}^{2N} \left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m)\}.$$

Перейдемо від α-диз'юнкцій до фільтрів, а потім скористаємося умовою дистрибутивності синхронної диз'юнкції:

$$Dancig = \{\underset{m>N}{} \left(\dot{\bigvee_{l=1}^{2N}} \left((\underline{\omega} PSub1(m,N,l) \times B_N^{(1)}(l) \vee \overline{\underline{\omega}} E) \times (\overline{\underline{\omega}} PSub2(m,N,l) \times B_N^{(2)}(l) \vee \omega E) \right) \right) \times$$
$$\times \left(\dot{\bigvee_{l=1}^{2N}} \left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m) \} =$$
$$= \{\underset{m>N}{} \left(\dot{\bigvee_{l=1}^{2N}} \left((\underline{\omega} PSub1(m,N,l) \times B_N^{(1)}(l) \vee \overline{\underline{\omega}} PSub2(m,N,l) \times B_N^{(2)}(l)) \right) \right) \times \quad (27)$$
$$\times \left(\dot{\bigvee_{l=1}^{2N}} \left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m) \}.$$

Замінимо фільтри знову на α-диз'юнкцію й об'єднаємо бар'єри $B_N^{(1)}$ та $B_N^{(1)}$, а також винесемо оператор асинхронної диз'юнкції:

$$Dancig = \dot{\bigvee_{l=1}^{2N}} \{\underset{m>N}{} \left(\left(\left(\underset{\omega}{} PSub1(m,N,l) \vee PSub2(m,N,l) \right) \times B_{2N}(l) \right) \right) \times \quad (28)$$
$$\times \left(\left(PSub3(m,2N,l) \times B_{2N}(l) \right) \right) \times (++m) \}.$$

Отриману формулу (28) схематично зображено на рис. 7. Основною її перевагою є менша кількість синхронізацій у порівнянні із (23) та (24), що безперечно покращить ефективність роботи алгоритму.

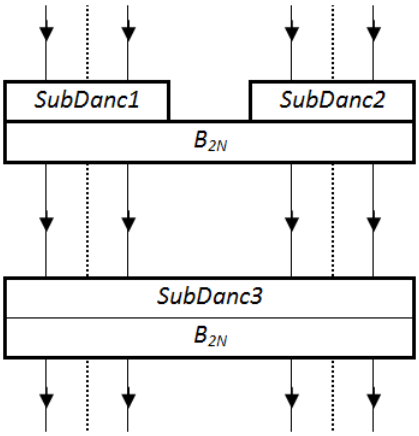


Рис. 7. Остаточна схема розпаралелювання

5. Висновки

У статті проаналізовано підхід до розв'язку задачі маршрутизації у комп'ютерних мережах, який ґрунтується на алгоритмі Данцига. Виконано формалізацію алгоритму з використанням математичного апарату САА-М і отримано послідовну регулярну схему (8).

Сформовано концепцію розпаралелювання алгоритму для систем зі спільною пам'яттю. Проаналізовано два підходи: в першому випадку зменшується навантаження на кожну з паралельних гілок, а в другому за рахунок зростання навантаження значно зменшуються витрати на синхронізацію гілок. Виконано трансформацію ПРС (8) за допомогою САА-М. Паралельні регулярні схеми (23), (24) представляють перший варіант, а (28) – другий.

Використання математичного апарату САА-М В.М. Глушкова дозволяє зосередитись на концепціях та методах трансформації паралельних схем алгоритмів, абстрагуючись від конкретних деталей програмної реалізації. При цьому отримані схеми є теоретичною базою для створення відповідного програмного забезпечення.

СПИСОК ЛІТЕРАТУРИ

1. Сик Дж., Ли Л., Ламсдэйн Э. C++ Boost Graph Library. Библиотека программиста / Пер. с англ. Р. Сузи. – СПб.: Питер, 2006. – 304 с.
2. Лісовець В., Цегелик Г. Метод М-паралельного послідовного пошуку записів у файлах баз даних і його ефективність // Вісник Львівського університету. Сер. прикл. матем. та інформ. – 2007. – Вип. 13. – С. 177 – 186.

3. Джонсон М. Харт Системное программирование под Windows / Пер. с англ. – 3-е изд. – М.: Издательский дом «Вильямс», 2005. – 592 с.
4. Майника Э. Алгоритмы оптимизации на сетях и графах. – М.: Мир, 1981. – 324 с.
5. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: Центр непрерывного математического образования, 2000. – 960 с.
6.Погорілий С.Д. Програмне конструювання: Підручник / За ред. О.В. Третяка. – 2-е вид. – К.: Видавничо-поліграфічний центр «Київський університет», 2005. – 483 с.
7. Погорілий С.Д., Калита Д.М. Оптимізація алгоритмів маршрутизації з використанням систем алгоритмічних алгебр // УсиМ. – 2000. – № 4. – С. 20 – 30.
8. Многоуровневое структурное проектирование программ. Теоретические основы, инструментарий / Е.Л. Ющенко, Г.Е. Цейтлин, В.П. Грицай и др. – М.: Финансы и статистика, 1989. – 342 с.
9. Богачёв К.Ю. Основы параллельного программирования. – М.: БИНОМ. Лаборатория знаний, 2003. – 342 с.

Стаття надійшла до редакції 28.05.2009