УДК 681.3

С.Д. ПОГОРІЛИЙ, Ю.В. БОЙКО, Р.В. БІЛОУС

ФОРМУВАННЯ ТА АНАЛІЗ ПАРАЛЕЛЬНИХ СХЕМ АЛГОРИТМУ ДЕЙКСТРИ

Abstract: *Formalization of Deykstri's algorithm using mathematical means of the modified systems of algorithmic algebras. Strategies for creating a parallel algorithm are proposed and parallel regular charts of algorithm for different structures are received.*
Key words: *routing, Deykstri's algorithm, parallel regular chart of algorithm, scheme's equivalent transformations, paralleling, process*.

Анотація: *Виконано формалізацію алгоритму Дейкстри з використанням математичного апарату модифікованих систем алгоритмічних алгебр. Запропоновано стратегії розпаралелювання та одержано паралельні регулярні схеми алгоритму для різних архітектур обчислювальних систем.*
Ключові слова: *маршрутизація, алгоритм Дейкстри, паралельна регулярна схема алгоритму, еквівалентні перетворення схем, розпаралелювання, процес.*

Аннотация: *Выполнена формализация алгоритма Дейкстры с использованием математического аппарата модифицированных систем алгоритмических алгебр. Предложены стратегии распараллеливания и получены параллельные регулярные схемы алгоритма для различных архитектур вычислительных систем.*
Ключевые слова: *маршрутизация, алгоритм Дейкстры, параллельная регулярная схема алгоритма, эквивалентные преобразования схем, распараллеливание, процесс.*

1. Вступ

Маршрутизація (routing) є однією із класичних функціональних задач мереж, ефективність розв'язання якої визначає продуктивність мережі в цілому. Необхідність значного підвищення швидкодії комп'ютерних мереж, зменшення часу реакції висуває задачу створення нових протоколів маршрутизації на основі продуктивних алгоритмів. У сучасних мережах використовуються такі протоколи маршрутизації стеку TCP/IP, як OSPF (грунтується на алгоритмі Дейкстри) та RIP (грунтується на алгоритмі Беллмана-Форда). Через швидкий розвиток галузі мережевих технологій розмірності оброблюваних графів зростають, тому виникає необхідність покращання часових характеристик сучасних алгоритмів маршрутизації.

Одним із способів зменшення часу виконання є розпаралелювання алгоритму або його частин з метою подальшої реалізації на паралельній архітектурі. Перетворити існуючий алгоритм на паралельний можна за допомогою досвіду, інтуїції або математичного апарату еквівалентних перетворень. В роботі розглянуто формування регулярних схем алгоритмів (РСА) та отримання паралельних регулярних схем алгоритмів (ПРСА) для алгоритму Дейкстри і обґрунтування результатів за допомогою апарату еквівалентних перетворень, для виконання яких застосовано математичний апарат модифікованих систем алгоритмічних алгебр (САА-М) В.М. Глушкова.

В задачі про найкоротші шляхи задається зважений орієнтований граф $G = (V, E, \omega)$,

де V – множина вершин графа G;

E – множина його ребер;

$\omega : E \to R$ – вагова функція ребер графа G.

Розглянемо задачу пошуку найкоротшого шляху з однієї вершини [1, 2], в якій для заданого графа $G = (V, E, \omega)$ потрібно знайти найкоротший шлях, що починається у визначеній вихідній

ISSN 1028-9763. Математичні машини і системи, 2008, № 4

вершині $r \in V$ і закінчується в кожній з вершин $v \in V$. Зазначимо деякі алгоритми, що вирішують ці задачі.

Алгоритм Беллмана-Форда [1, 2] дозволяє вирішити задачу про найкоротший шлях із однієї вершини в загальному випадку, коли вага кожного з ребер може бути від'ємною. Для заданого зваженого орієнтованого графа алгоритм Беллмана-Форда повертає логічне значення, що вказує на те, чи присутній у графі цикл з від'ємною вагою, досяжний з витоку. Якщо такий цикл існує, в алгоритмі вказується, що рішення не існує. Якщо ж таких циклів немає, алгоритм видає найкоротші шляхи та їх вагу.

2. Алгоритм Дейкстри [1, 2] вирішує задачу пошуку найкоротшого шляху із однієї вершини у зваженому орієнтованому графі в тому випадку, коли ваги ребер невід'ємні.

Введемо позначення:

r – опорна вершина;

L – множина вершин графа, для яких обчислено найкоротший шлях від опорної вершини r;

n – кількість вершин графа;

v – поточна вершина графа.

Схема послідовного алгоритму Дейкстри може бути представлена у вигляді [1, 2].

Алгоритм 1.

1. Процедура Dijkstra_Single_Source_Shortest_Path (V, E, ω, r).
2. $L := \{r\}$.
3. Для всіх вершин $v \in (V - L)$.
4. Якщо ((r, v) існує) то $s[v] := \omega[r, v]$.
5. Якщо ((r, v) не існує) то $s[v] := \infty$.
6. **Поки** ($L \neq V$) виконувати.
7. Знайти вершину u, таку, що $s[u] := \min\{s[v] | v \in (V - L)\}$.
8. $L := L \bigcup \{u\}$.
9. **Для** усіх вершин $v \in (V - L)$.
10. $s[v] := \min\{s[v], s[u] + \omega[u, v]\}$.

Алгоритм використовує множину L, що містить вершини графа, для яких найкоротший шлях від вихідної вершини r уже знайдено. Він також використовує чергу з пріоритетами $Q = L/V$ та атрибутом $s[1..n]$ (де $n -$ кількість вершин графа G), в якому для кожної вершини $v \in V$ зберігається поточне значення найкоротшого шляху.

У ході кожної нової ітерації алгоритму до множини L додається нова вершина u, така, що $s[u] = \min\{s[v] | v \in (V - L)\}$. Після додавання вершини всі значення масиву $s[v]$ оновлюються,

якщо сума відстані до вершини u та ваги ребра з «бахроми» менша, ніж поточна відстань до вершини. Алгоритм закінчує роботу, коли $L = V$.

Час виконання алгоритму Дейкстри залежить від реалізації черги з пріоритетами [1]. Спочатку розглянемо випадок, коли черга з пріоритетами підтримується за рахунок того, що всі вершини пронумеровані від 1 до $|V|$. Атрибут $s[v]$ просто поміщається як елемент масиву з індексом v.

Кожна операція додавання вершини до множини L та операція переходу до наступної вершини займає час $O(1)$, а кожна операція пошуку мінімуму ($s[u] = \min\{s[v] | v \in (V - L)\}$) $O(V)$ (оскільки в ній виконується пошук по всьому масиву); в результаті повний час роботи алгоритму дорівнює $O(V^2 + E) = O(V^2)$.

Слід зазначити, що у випадку ненасиченого графа G черга з пріоритетами задається у вигляді двійкового дерева. Тоді операція пошуку $s[u] = \min\{s[v] | v \in (V - L)\}$ займає $O(\lg V)$, а таких операцій відбувається $|V|$, тобто сумарна складність алгоритму складає $O((V + E)\lg V)$. Ця складність еквівалентна $O(E \lg V)$, якщо всі вершини графа G є досяжними з опорної вершини r.

3. РСА Дейкстри

Виконаємо формалізацію послідовного алгоритму Дейкстри. Розглянемо зважений граф $G = (V, E, W)$, де W – вагова матриця суміжності графа G.

Введемо додаткові позначення:

$next(v)$ – перехід до наступної за порядком вершини графа;

$first$ – перша вершина у списку вершин графа;

$last$ – остання вершина у списку.

Ініціалізація алгоритму відбувається в два етапи:

– спочатку задається початкове значення множини L викликом функції $Init(L)$.

Функція $Init(L)$ виконує $L := \{r\}$;

– задаються початкові значення елементів масиву $s[v]$ викликом функції $Init(s[v])$.

Зазначена функція $Init(s[v])$ для всіх вершин $v \in (V - L)$ виконує $s[v] := \omega[r, v]$, якщо ребро (r, v) існує, або $s[v] := \infty$, якщо ребро (r, v) не існує.

Використовуючи схему послідовного алгоритму, описану вище (Алгоритм 1), побудуємо регулярну схему алгоритму, попередньо увівши деякі позначення:

Позначення для операторів:

A = $Init(L)$;

B = $Init(s[v])$;

C = $(v := first)$;

D = $next(v)$;

E = $(\min := s[u])$;

F = $(u := v)$;

G = $(L := L \cup \{u\})$;

H = $(s[v] := s[u] + \omega[u,v])$.

Позначення для предикатів:

$\alpha = (v \neq last)$;

$\beta = (v \in (V - L))$;

$\gamma = (s[v] < \min)$;

$\delta = (L \neq V)$;

$\mu = (s[v] < s[u] + \omega[u,v])$.

Пошук вершини u, такої, що $s[u] := \min\{s[v] | v \in (V - L)\}$, можна представити у вигляді схеми

$$\underset{\alpha\,\beta\,\gamma}{(C)*\{\,(\,((F)*(E)))*(D\}}.$$

Внутрішній покроковий цикл останньої схеми можна представити у вигляді

$$\underset{\alpha\,\beta\,\mu}{(C)*\{\,(\,(H))*(D\}}.$$

Тоді схема алгоритму Дейкстри може бути записана у такому вигляді:

$$\underset{\delta}{A*B*\{}\underset{\alpha\,\beta\,\gamma}{C*\{\,(\,(F*E))*D\}}*\underset{\alpha\,\beta\,\mu}{C*\{\,(\,(H))*D\}\}}. \qquad (1)$$

Введемо позначення: $\beta \wedge \gamma = \sigma$ та $\beta \wedge \mu = \phi$.

Тоді схема (1) матиме вигляд

$$\underset{\delta}{A*B*\{}\underset{\alpha\,\sigma}{C*\{\,(\,F*E)*D\}}*\underset{\alpha\,\phi}{C*\{\,(H)*D\}\}}. \qquad (2)$$

Одержане співвідношення (2) є послідовною РСА Дейкстри.

4. Концепції розпаралелювання

Користуючись класифікацією Фліна [5, 7], виділяють 4 класи обчислювальних систем. В класі MIMD будемо розрізнювати:

● обчислювальні системи зі слабким зв'язком між процесорами, до яких відносять системи із розподіленою пам'яттю (кластерні архітектури);

● обчислювальні системи із сильним зв'язком (системи зі спільною пам'яттю).

Далі розглянуто обидві архітектури обчислювальних систем класу MIMD.

Алгоритм Дейкстри є ітераційним [3]. Кожна ітерація додає нову вершину до множини L. Оскільки значення $s[v]$ для вершини v може змінитись після кожної ітерації, то за одну ітерацію

неможливо включити більше ніж одну вершину до множини L. Це означає, що зовнішня $\alpha-$ ітерація не підлягає розпаралелюванню.

Нехай $p-$ кількість процесів, а $n-$ кількість вершин графа G. Множина V ділиться на p рівно потужних підмножин. Нехай V_i- підмножина вершин, що пов'язана з процесом P_i (де $i=0,1,...,p-1$). Кожна підмножина V_i містить n/p вершин, і обробка кожної з них проводиться різними процесами.

5. Системи з розподіленою пам'яттю

Кожен процес P_i зберігає частину масиву s, що відноситься до V_i (тобто процес P_i зберігає $s[v]$, такі, що $v \in V_i$). На рис. 1 зображено таке розбиття:

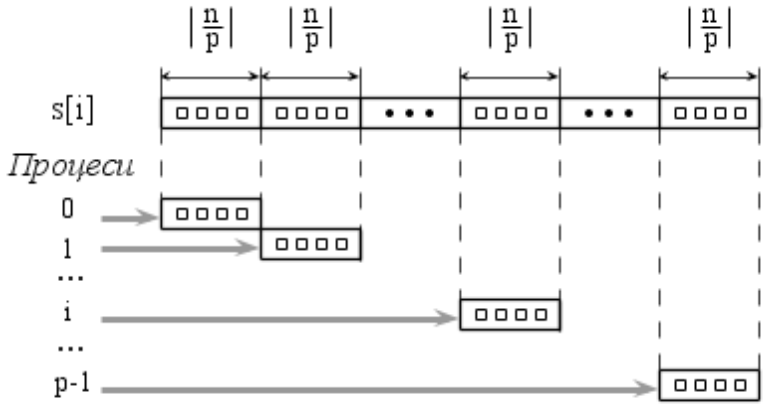

Рис. 1. Розбиття масиву s для систем із розподіленою пам'яттю

Кожен процес P_i обраховує $s[u]=\min\{s[v]|v\in(V-L)\cap V_i\}$ під час кожної $\alpha-$ітерації. Загальний мінімум одержується з усіх $s_i[u]$ за допомогою операції зведення даних з усіх процесів в один (all-to-one reduction) і зберігається в процесі P_0. Тепер процес P_0 містить нову вершину u, що буде включена до множини L. Процес P_0 посилає u усім процесам, використовуючи операцію розсилання даних (one-to-all broadcast). Процес P_i, що відповідає вершині u, позначає її як вершину, що належить до L. Тоді кожен процес обраховує нові значення $s[v]$ для відповідних вершин.

Процес, що відповідає за вершину v, повинен «знати» вагу ребра $\omega(u,v)$. Отже, кожен процес P_i повинен зберігати стовпці вагової матриці суміжності графа, які відповідають підмножині його вершин V_i. Це відповідає розбиттю матриці на блоки. Об'єм пам'яті, необхідний для збереження одного блока для кожного процесу, обчислюється як $O(n^2/p)$. Рис. 2 ілюструє розбиття даних між процесами:

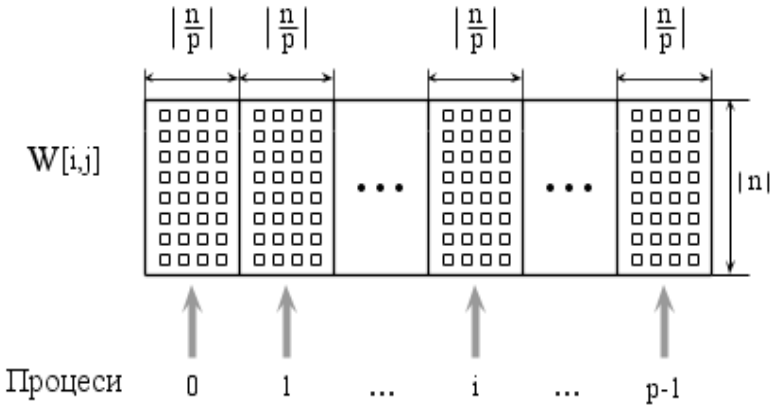

Рис. 2. Розбиття матриці суміжності графа для систем
із розподіленою пам’яттю

У такий спосіб обчислювальні затрати для розв’язання задачі розподіляються між усіма p процесами. Але в цьому випадку з’являються додаткові накладні витрати, пов’язані з обміном даними між процесами при кожній ітерації циклу задачі.

6. Системи зі спільною пам’яттю

Як і для випадку систем з розподіленою пам’яттю, кожен окремий процес P_i обраховує $s[u] = \min\{s[v] | v \in (V - L) \bigcap V_i\}$ під час кожної ітерації циклу та обраховує нові значення $s[v]$ для відповідних вершин.

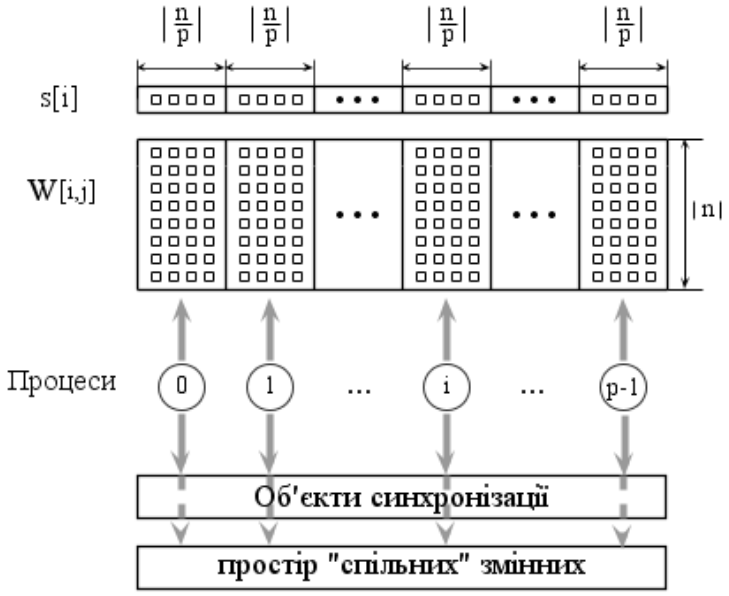

Рис. 3. Розбиття даних для систем зі спільною пам’яттю

Пошук загального мінімуму в масиві $s[v]$ у цьому випадку відбувається за допомогою об'єктів синхронізації, що залежать від конкретної програмної реалізації алгоритму. Розбиття за даними (масиву $s[v]$ та матриці суміжності графа) та за контролем над «спільними» змінними схематично зображено на рис. 3.

У такому випадку зникають накладні витрати на пересилання даних між процесами. Але додатковий час роботи програми витрачається на синхронізацію процесів та контроль над «спільними» змінними. «Спільними» будемо називати ті ресурси (дані), контроль над якими здійснюється паралельно усіма процесами одночасно. Ці ресурси відповідно потребують додаткового контролю коректності звертань та синхронізації процесів.

7. Формування паралельних схем алгоритму

Системи з розподіленою пам'яттю. Сформулюємо ПРСА Дейкстри за описаним вище підходом, користуючись апаратом еквівалентних трансформацій у відповідній алгебрі [4, 11].

Користуючись методом розпаралелювання за даними, що описаний вище, розглянемо співвідношення (2).

Використаємо формулу $(\underset{\alpha}{A}) = \underline{\alpha} A$ і позначимо фільтри. При цьому виконання алгоритму для гілок з істинними умовами продовжується, а для інших обривається:

$$A * B * \{\underset{\delta}{C} * \{(\underline{\sigma}(F * E)) * D\} * G * C * \{(\underline{\phi} H) * D\}\}_{\alpha}. \tag{3}$$

Предикат α може бути записаний у вигляді кон'юнкції в такому вигляді:

$$\alpha = \alpha_1 \wedge \alpha_2 \wedge .. \wedge \alpha_p \ ,$$

де $\alpha_i = (v_i \neq last_i)$, $last_i = first + (n/p) * i$, $p -$ кількість процесів.

Це дає змогу сформувати паралельні гілки алгоритму, користуючись наведеними вище концепціями розбиття даних.

Тоді схема (3) матиме вигляд

$$A * B * \{\underset{\delta}{C} * \{(\underline{\sigma}(F * E)) * D\} * G * C * \{(\underline{\phi} H) * D\}\}_{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_p}. \tag{4}$$

Введемо додаткові позначення:

$$K = (\underline{\phi} H) * D \ .$$

Оскільки умова циклу – кон'юнкція, то формулу (4) можна представити у вигляді паралельної схеми:

$$A * B * \{\underset{\delta}{C} * \{(\underline{\sigma}(F * E)) * D\} * G * C * \#\{K\}_{\alpha_1} \overset{\bullet}{\vee} \{K\}_{\alpha_2} \overset{\bullet}{\vee} ... \overset{\bullet}{\vee} \{K\}_{\alpha_p} \#\}. \tag{5}$$

Символами «#...#» позначається частина, оператори якої виконуються паралельно, а символами «$\overset{\bullet}{\vee}$» – оператори асинхронної диз'юнкції паралельних гілок алгоритму.

Зрозуміло, що в цій схемі значення $last_i$ та v_i є локальними для кожної гілки виконання. Скористаємось розподільчою властивістю паралельного виконання гілок алгоритму. Таке перетворення дозволяє сформувати незалежні паралельні гілки виконання алгоритму і таким чином зменшити накладні витрати на синхронізацію даних.

Використавши рівність $A*\#B\overset{\bullet}{\vee}C\#=\#A*B\overset{\bullet}{\vee}A*C\#$, одержимо:

$$A*B*\underset{\delta}{\{}C*\underset{\alpha}{\{}(\underline{\sigma}(F*E))*D\}*G*\#C*\underset{\alpha_1}{\{}K\}\overset{\bullet}{\vee}C*\underset{\alpha_2}{\{}K\}\overset{\bullet}{\vee}...\overset{\bullet}{\vee}C*\underset{\alpha_p}{\{}K\}\#\}. \qquad (6)$$

Тут $C=(v_i \coloneqq first_i)$, $first_i=(n/p)*(i-1)$.

Позначимо $M_i=C*\underset{\alpha_i}{\{}K\}$, тоді (6) можна подати у вигляді

$$A*B*\underset{\delta}{\{}C*\underset{\alpha}{\{}(\underline{\sigma}(F*E))*D\}*G*\#M_1\overset{\bullet}{\vee}M_2\overset{\bullet}{\vee}...\overset{\bullet}{\vee}M_p\}\#\}. \qquad (7)$$

Скориставшись таким же принципом, можна розподілити пошук найменшого шляху серед вершин $v\,|\,v\in(V-L)$.

Але в цьому випадку слід ввести додатковий оператор.

Позначимо $Q-$ оператор пошуку найкоротшого шляху серед результатів пошуку дочірніх процесів та розсилання результату всім дочірнім процесам.

Введемо позначення: $N_i=C*\underset{\alpha_i}{\{}\underline{\sigma}(F*E)*D\}$.

Тоді схему (7) можна записати так:

$$A*B*\underset{\delta}{\{}\#N_1\overset{\bullet}{\vee}N_2\overset{\bullet}{\vee}...\overset{\bullet}{\vee}N_p\#*Q*G*\#M_1\overset{\bullet}{\vee}M_2\overset{\bullet}{\vee}...\overset{\bullet}{\vee}M_p\#\}. \qquad (8)$$

Схема (8) являє собою ПРСА Дейкстри, представленою в САА-М, для систем із розподіленою пам'яттю.

Системи зі спільною пам'яттю. На першому кроці скористаємось аналогічним методом розпаралелення за даними для обчислення нових значень поточного найкоротшого шляху при кожній ітерації, що було використано для систем з розподіленою пам'яттю.

Використаємо формулу (7). Розглянемо тепер послідовність операторів пошуку найменшого шляху серед вершин $v\,|\,v\in(V-L)$:

$$C*\underset{\alpha}{\{}(\underline{\sigma}(F*E))*D\}. \qquad (9)$$

Оскільки наведена послідовність операторів працює з даними, що є «спільними» для усіх паралельних гілок алгоритму, необхідно ввести додаткові оператори синхронізації:

$L(Lock)-$ оператор, що блокує доступ до простору «спільних» змінних для усіх паралельних гілок, окрім поточної. Якщо ж даний ресурс вже заблоковано, то оператор очікує його звільнення;

$U(Unlock)-$ оператор, що знімає блокування простору «спільних» змінних .

З урахуванням наведених вище позначень формулу (9) можна записати у вигляді

$$C * \underset{\alpha}{\{}(\underline{\sigma}(L * F * E * U)) * D\}.$$

Тоді ПРСА для систем зі спільною пам'яттю матиме вигляд

$$A * B * \underset{\delta}{\{} C * \underset{\alpha}{\{}(\underline{\sigma}(L * F * E * U)) * D\} * G * \# M_1 \overset{\bullet}{\vee} M_2 \overset{\bullet}{\vee} ... \overset{\bullet}{\vee} M_p \}\#\}. \qquad (10)$$

Введемо позначення:

$$S_i = C * \underset{\alpha_i}{\{} \underline{\sigma}(L * F * E * U) * D\}.$$

Аналогічно, як і у схемі для систем з розподіленою пам'яттю, схему (10) подамо у вигляді

$$A * B * \underset{\delta}{\{}\# S_1 \overset{\bullet}{\vee} S_2 \overset{\bullet}{\vee} ... \overset{\bullet}{\vee} S_p \#* G * \# M_1 \overset{\bullet}{\vee} M_2 \overset{\bullet}{\vee} ... \overset{\bullet}{\vee} M_p \#\}.$$

У такий спосіб одержано 2 схеми для дослідження та реалізації алгоритму Дейкстри для різних MIMD архітектур обчислювальної системи.

8. Оцінка паралельних схем алгоритму

Для аналізу паралельних схем алгоритму введемо деякі функціонали для позначення ефективності та підвищення швидкодії їх реалізацій [8, 11].

$W(p)-$ число елементарних операцій, що виконують p процесів. Ця величина відноситься до обчислювальних затрат.

$T(p)-$ час виконання програми на $p-$ процесорній системі; зрозуміло, що $T(1) = W(1)$ та $T(p) \le W(p)$.

$S(p)-$ підвищення швидкодії алгоритму $S(p) = \dfrac{T(1)}{T(p)}$.

$E(p)-$ ефективність алгоритму $E(p) = \dfrac{T(1)}{pT(p)}$.

Оцінка функції складності алгоритму дозволяє визначити, як швидко зростають затрати ресурсів на виконання алгоритму при зростанні об'єму даних. В асимптотичному аналізі алгоритмів використовуються позначення, прийняті в математичному асимптотичному аналізі.

Системи з розподіленою пам'яттю. Обрахунок даних одним процесом, що включає знаходження мінімуму та обрахунок нових значень $s[v]$ під час кожної ітерації, займає $O(n^2/p)$ часу. Зв'язок між процесорами, що відбувається при кожній ітерації, займає час, необхідний для виконання двох операцій: all-to- one reduction та one-to-all broadcast. Затрати на ці операції залежать від конкретної реалізації паралельної обчислювальної установки. Для $p-$процесорної MP (Massage Passing) – архітектури ці затрати визначаються як $O(n(t_s+t_t)\log p)$, де t_s (startup time) – час, необхідний для обробки повідомлення процесами, що посилають та отримують його. Він включає час на підготовку повідомлення (додавання заголовку, корекція помилок) та час на його маршрутизацію. Ця затримка формується лише раз при передачі одного повідомлення.

t_t (transfer time) – час, необхідний для передачі одиниці інформації («слова») через канал обміну.

Отже для такої паралельної схеми алгоритму сумарний час його роботи складає T_p, що визначається за формулою

$$T_p = O\left(\frac{n^2}{p}\right) + O(n(t_s+t_t)\log p).$$

Тобто підвищення швидкодії та ефективність паралельного алгоритму в порівнянні з послідовним складають відповідно

$$S = \frac{O(n^2)}{O\left(\frac{n^2}{p}\right) + O(n(t_s+t_t)\log p)},$$

$$E = \frac{1}{1+O(p(t_s+t_t)\log p/n)}.$$

Системи зі спільною пам'яттю. Як і у випадку з системами з розподіленою пам'яттю, обробка даних одним процесом, що включає знаходження мінімуму та обрахунок нових значень $s[v]$ під час кожної ітерації, займає $O(n^2/p)$ часу. Оскільки дані розділяють спільну пам'ять, додаткові затрати часу T_s займає лише оператор синхронізації процесів. Час на його виконання залежить від конкретної реалізації обчислювальної установки. Отже

$$T_p = O\left(\frac{n^2}{p}\right) + O(nT_s(n,p)).$$

У найпростішому випадку виключне звертання до простору «спільних» змінних відбувається за рахунок певного об'єкта синхронізації, звертання до якого відбувається усіма процесами при кожній ітерації циклу. Таким чином, сумарні затрати на синхронізацію можна обчислити як $O(n\cdot p)$.

Отже для такої паралельної схеми алгоритму сумарний час роботи визначається за формулою

$$T_p = O\left(\frac{n^2}{p}\right) + O(n \cdot p).$$

Підвищення швидкодії та ефективність паралельного алгоритму в порівнянні з послідовним складають відповідно

$$S = \frac{O(n^2)}{O\left(\frac{n^2}{p}\right) + O(n \cdot p)}, \quad E = \frac{1}{1 + O(p^2 / n)}.$$

Зобразимо схематично залежності швидкодії від кількості обчислювальних вузлів та розмірності вхідних даних для систем з розподіленою пам'яттю (рис. 4) та систем зі спільною пам'яттю (рис. 5).

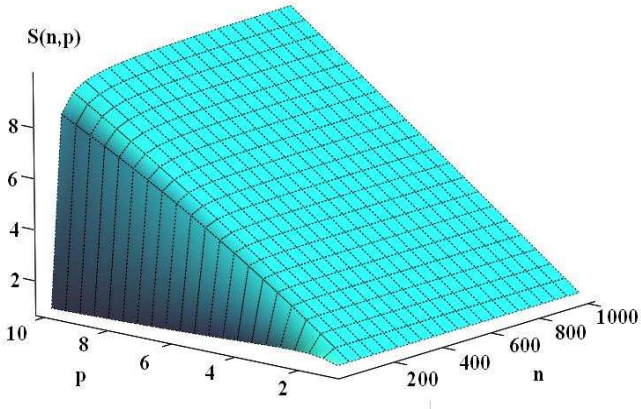

Рис. 4. Підвищення швидкодії для систем з розподіленою пам'яттю

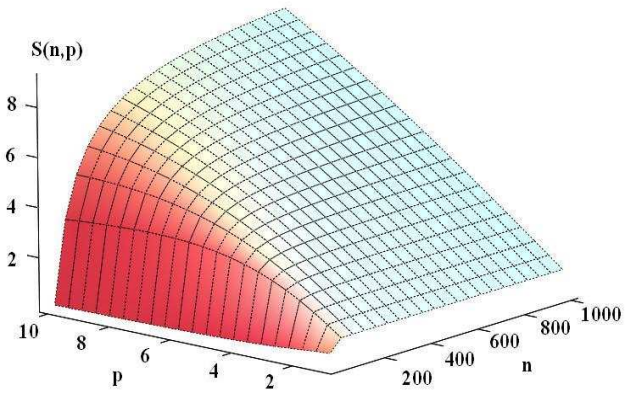

Рис. 5. Підвищення швидкодії для систем зі спільною пам'яттю

З графіків видно, що швидкодія алгоритму зростає зі збільшенням кількості процесорів для обох архітектур. Слід зазначити, що з отриманих графіків випливає доцільність використання спільної пам'яті для графів малої розмірності при невеликій кількості процесорів, тоді як модель розподіленої пам'яті виявляється значно ефективнішою для невеликих масивів оброблюваних даних, коли кількість вузлів зростає. Для графів великої розмірності обидва підходи однаково ефективні, що пояснюється відсутністю будь-яких обмежень в теоретичних моделях, які були використані.

9. Висновки

1. У статті виконано формалізацію послідовного алгоритму Дейкстри з використанням математичного апарату САА-М і отримано його РСА .

2. Запропоновано концепції розпаралелювання алгоритму для двох різних архітектур обчислювальних систем: розподілених систем та систем зі спільною пам'яттю.

3. Отримано ПРСА за допомогою еквівалентних трансформацій в САА-М.

4. Користуючись асимптотичним аналізом алгоритмів, оцінено ефективність та швидкодію ПРСА Дейкстри.

5. Отримані ПРСА є підґрунтям для створення мережевих маршрутизаторів нового покоління.

СПИСОК ЛІТЕРАТУРИ

1. Седжвик Р. Фундаментальные алгоритмы на C++. Алгоритмы на графах / Пер. с англ. – СПб.: ООО «ДиаСофтЮП», 2002. – 496 с.
2. Кормен Т. и др. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: Центр непрер. математического образования, 2000. – 960 с.
3. Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar «Introduction to Parallel Computing» Addison Wesley 2003 ISBN- 0-201-64865-2. – 856 p.
4. Погорілий С.Д., Калита Д.М. Оптимізація алгоритмів маршрутизації з використанням систем алгоритмічних алгебр // УСиМ. – 2000. – № 4. – С. 20 – 30.
5. Богачёв К.Ю. Основы параллельного программирования. – М.: БИНОМ. Лаборатория знаний, 2003. – 342 с.
6. 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.
7. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – Санкт-Петербург: БХВ-Петербург, 2002. – 608 с.
8. Parhami, Behrooz Introduction to Parallel Processing: Algorithms and Architectures. – Kluwer Academic Publishers, 2002.
9. Parallel and Distributed Programming Using C++ Cameron Hughes, Tracey Hughes Addison Wesley 2003 ISBN: 0-13-101376-9. – 720 p.
10. Погорілий С.Д. та інш. Про підвищення швидкодії алгоритмів формування мінімального вкриваю чого дерева / С.Д. Погорілий, О.О. Камардіна, Ю.С. Кордаш // Математичні машини і системи. – 2005. – № 4. – С. 30 – 38.
11. Погорілий С.Д. Програмне конструювання. – К.: ВПЦ “Київський університет”, 2005. – 438 с.

Стаття надійшла до редакції 27.03.2008