УДК 004.3

С.Д. Погорілий, М.І. Трибрат, Ю.В. Бойко, Д.В. Афанасьєв

ПІДХОДИ ДО ПАРАЛЕЛІЗАЦІЇ АЛГОРИТМУ ЙЕНА ДЛЯ СИСТЕМ ІЗ СПІЛЬНОЮ ПАМ'ЯТТЮ

Побудовано регулярну схему алгоритму Йена з використанням апарату систем алгоритмічних алгебр, запропоновано підходи до паралельної реалізації алгоритму Йена для SMP–архітектур. Виконано формалізацію методу шаблонів шляхом формування паралельних схем з використанням математичного апарату модифікованих систем алгоритмічних алгебр. Реалізовано паралельний алгоритм Йена, наведено отримані експериментальні дані.

Вступ

Типові розв'язки задачі знаходження найкоротшого шляху в теорії графів дозволяють отримати один шлях для пари вершин заданого графу. Практичні застосування вимагають накладення на знайдені шляхи специфічних вимог, задоволення яких відбувається за рахунок введення ряду обмежень у відомі алгоритми пошуку Процедура модифікації алгоритмів призводить до значного збільшення обчислюваних потужностей.

Простішою виявляється задача вибору оптимального за заданими критеріями шляху з деякої множини доступних шляхів, тому актуальною є задача пошуку *K* – найкоротших між розглядуваними вершинами шляхів. Безпосередньо дана задача виникає при пошуку альтернативних шляхів проходження пакету між маршрутизаторами в комп'ютерній мережі. Її розв'язання дозволяє зменшити характерні часи збіжності в мережі.

Вперше проблема знаходження *K*–найкоротших шляхів була поставлена в 1959 році Хоффманом і Певлі. В подальшому вона досліджувалася Сакаровичем, Белманом, Калабою, Епштейном та ін. У роботі [1] проведено експериментальний аналіз ефективності чотирьох алгоритмів розв'язку розглядуваної задачі – Йена, Лаурера, Катоха і Хоффмана. Експериментальні залежності часу витраченого на пошук від кількості вузлів у графі (в якості графа було розглянуто топологію системи *European Optical Network* [2]) вказують на алгоритм Хоффмана як найбільш ефективний серед розглядуваних. Але у даному алгоритмі на знайдені шляхи не накладається умова простоти (ациклічності). Натомість, практичні застосування вимагають саме простих шляхів. Цим умовам задовольняють алгоритми, у основі яких лежить алгоритм Йена. В літературі під назвою алгоритм Йена зазвичай розуміють модифікацію цього алгоритму Лаулером. Складність даного алгоритму в найгіршому випадку для графа із додатними ребрами $O(Kn^3)$ [3].

На сьогодні поширеною є практика розпаралелювання класичних алгоритмічних задач, актуальність якої підживлюється напрямком розвитку комп'ютерної техніки у бік збільшення кількості ядер на одному кристалі.

Мета дослідження – аналіз існуючих та розробка нових формалізованих підходів до паралельної реалізації алгоритму Йена за рахунок їх опису з використанням математичного апарата модифікованих систем алгоритмічних алгебр В.М. Глушкова (САА-М) [4], а також експериментальна перевірка та аналіз ефективності паралельних реалізацій алгоритму Йена для двоядерних і чотириядерних архітектур.

1. Формалізація алгоритму

Для формального опису алгоритму введемо наступні позначення:

- $G = (X, R)$ – визначає заданий граф;
- $X = (x_1, \ldots, x_n)$ – скінчена множина елементів, що називаються вузлами ;
- $R = \{(x_i, x_j)\} \subseteq X \times X$ – множина, що задає бінарне відношення на X, елемент такої множини R називається ребром

© Погорілий С.Д., Трибрат М.І., Бойко Ю.В., Афанасьєв Д.В., 2011

$r_{i,j} = (x_i, x_j)$, кожному з яких поставлене у відповідність число $w_{i,j}$, що називається вагою ребра. При задані відношення R на X кожному елементу x_i зіставимо певну підмножину X , що позначимо її як Rx_i ;

- $p =< s, x_1^{'}, x_2^{'} \ldots, t >$ – послідовність вузлів така, що $\exists$ ребро $r_k = (x_k^{'}, x_{k+1}^{'}) \in R$ називається шляхом з s вузла в p. Шлях p буде простим, якщо $x_a^{'} \neq x_b^{'}, \forall \; x_a^{'}, x_b^{'} \in p$. Вузли x_i, x_j називаються початковим і кінцевими вузлами шляху p. P_{x_i,x_j} визначає множину шляхів з вузла x_i в x_j у графі G. Нехай x_a і x_b два вузли, що належать шляху $p \in P_{x_i,x_j}$, тоді кажуть $q \in P_{x_a,x_b}$ підшлях p, якщо він збігається з p від x_a до x_b. Позначатимемо такий шлях $sub_p(x_a, x_b)$. Кожному шляху ставиться у відповідність величина $w(p) = \sum_{(a,b)\in p} w_{a,b}$, яка носить назву вартості шляху, та величина $c(p)$, яка визначає кількість вузлів, що входять у шлях p ;
- $p <> q$ – об'єднання двох шляхів $p \in P_{x_i,x_j}$ і $q \in P_{x_j,x_l}$, шлях з x_i в x_l, сформований шляхом p та доповнений q.

Алгоритм Йена є девіаційним алгоритмом, який передбачає, що один з найкоротших шляхів уже знайдено за допомогою базового алгоритму пошуку найкоротшого шляху між заданими вершинами, який застосовується в найгіршому випадку *Kn* разів. Наступний шлях шукається на основі попередніх як відгалуження і має відрізнятися від попереднього хоча б одним ребром.

- $BSPA(G(X), x_1, x_2)$ – базовий алгоритм пошуку найкоротшого шляху між вершинами x_1, x_2 графа G;
- $d(p)$ – вузол девіації, тобто вузол батьківського для шляху p, з якого виходить дочірній шлях;
- $d_{-1}(p)$ – вузол в шляху p, який займає попередню позицію за відношенням до $d(p)$;
- $q(p)$ – кількість вузлів в шляху p спільна з батьківським;
- L – буфер шляхів – претендентів на звання найкоротшого шляху.

Визначимо впорядковану множину K – найкоротших шляхів між заданими початковою s і кінцевою t вершинами $P_K = \{p_{1,\ldots,} p_K\} \subseteq P_{st}$ шляхів таких, що :

- p_k простий шлях, $k \in \{1, \ldots, K\}$;
- $w(p_k) \le w(p_{k+1})$, $k \in \{1, \ldots, K-1\}$;
- p_k визначений раніше p_{k+1}.

Множина P_K, побудована в такий спосіб представляє собою дерево із коренем у вершині s (рис. 1).

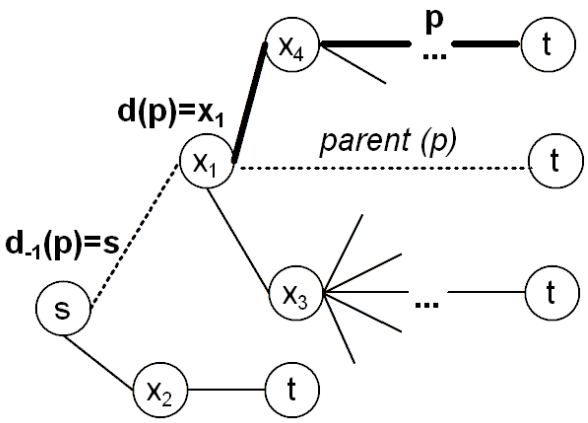


Рис. 1. Структура множини P_K

2. Формування схеми алгоритму

2.1. Загальна схема роботи алгоритму. Нехай відомий k–ий найкоротший шлях p_{k} з набору P_K, тоді дії алгоритму виглядатимуть наступним чином :

- перебір вершин x_i^k останнього знайденого шляху p_{k} включно до вузла девіації з метою пошуку дочірнього шляху, що збігається з p_{k} в частині $sub_{p_k}(s, x_i^k)$. За для унікальності від батьківського попередньо із множини R видаляється ребро (x_i^k, x_{i+1}^k), яке після відпрацювання на заданому кроці базового алгоритму знову відновлюється ;
- внесення знайдених шляхів у загальний буфер L кандидатів на $p_{\text{k+1}}$;
- вибір найкоротшого шляху з буфера й доповнення ним P_K.

З метою пошуку виключно ациклічних шляхів базовий алгоритм повинен працювати на k–му кроці з множиною $X = X \setminus \{s,..d_{-1}(p_k)\}$ (у цьому і полягає модифікація класичного алгоритму Йена Лаулером). Щоб забезпечити унікальність дочірнього шляху, множина R має бути модифікована

$$R = R \setminus \{(d(p_k),x_1),(d(p_k),x_2)...\};$$

$$x_1,x_2... | x_1 \subset p_1, x_2 \subset p_2,..p_1,p_2 \subset P_K,$$

що означає видалення з множини R ребер всіх шляхів, що отримані як відхилення від деякого спільного батьківського.

Детально алгоритм демонструє схему 1.

Схема 1:

Ініціалізація:

{

$BSPA(G(X),s,t);$ $\quad d(p) = s$

$L = \{p\};$ $\quad k = 0.$

}

Поки ($L \neq 0 \wedge \mathrm{k} \neq \mathrm{K}$)

{

$k=k+1$;

$\mathrm{p_k} = \min_{w(p \subset L)} L$;

$\mathrm{L} \to \mathrm{L} - \{\mathrm{p_k}\}$;

$\mathrm{X} \to X - p + \{t\}$;

Поки $((d(p_k),v^j) \in (p_j \subset P_K))$

$\mathrm{R} \to \mathrm{R} - \{(d(p_k),v^j)\}$;

$i = c(p_k) - 1;$

Поки ($\mathrm{x_i^k} \neq d_{-1}(p_k)$)

{

$\mathrm{X} \to \mathrm{X} + \{x_i^k\}$;

$\mathrm{p_L^k} = sub_{p_k}(s,x_i^k) <> BSPA(\mathrm{G(X)},x_i^k,t)$;

$\mathrm{L} \to \mathrm{L} + \{\mathrm{p_L^k}\}$;

}

$\mathrm{X} \to \mathrm{X} + \{sub_{p_k}(s,d(p_k))\}$;

Поки $((d(p_k),v^j) \in (p_j \subset P_K))$

$\mathrm{R} \to \mathrm{R} + \{(d(p_k),v^j)\}$;

}.

Важливою особливістю реалізації алгоритму є порядок перебору вершин шляху p_k: з метою підвищення ефективності виконання прохід по вершинами має відбуватися в оберненому порядку, що забезпечить оптимізацію витрат на видалення і повернення вершин у множину X [5]. Саме такий спосіб демонструє схема 1.

2.2. САА-схема алгоритму. Для формалізації алгоритму використовувався апарат САА. За своїми зображувальними можливостями він еквівалентний класичним алгоритмічним системам, таким як машини Тьюринга, рекурсивні функції, ланцюги Маркова та ін. Переваги САА – орієнтація на алгоритмічні задачі та розвинена система засобів формальних перетворень, які застосовуються до формального запису алгоритму – його САА схеми. Дана алгебра є багатоосновною, основами якої слугують множина операторів та множина умов з визначеними на даних множинах сигнатурою операцій.

Нехай:

- k – поточне значення потужності множини найкоротших шляхів P_K ;
- x – поточна вершина графа;
- p – поточний шлях;
- g – кількість спільних вузлів шляху з батьківським на даному кроці ітерації алгоритму.

Визначимо допоміжні оператори:

- $next(.)$ – наступний елемент у множині заданій як аргумент ;
- $previous(.)$ – попередній елемент у множині заданій як аргумент ;
- $first(.)$, $last(.)$ – перший та передостанній елементи у множині, заданій як аргумент;
- $parent(p)$ – шлях, батьківський за відношенням до шляху p, який міститься в множині P_K .

Визначимо множини операторів і предикатів у відповідності до Схеми 1 :

Оператори:

$S = \{\mathrm{p=BSPA(G,s,t)}, \mathrm{d(p)=s}, \mathrm{L=\{p\}}, \mathrm{k=0}\};$

$A = \{p := \min_c L, L := L - p, P_K := P_K + p\};$

$F = (x = first(p));$

$L = (x := last(p));$

$N = (x := next(p));$

$P = (x := previous(p))$;
$C = (X := X - x)$;
$I = (X := X + x)$;

$R = (R := R - (x, next(x)))$;
$E = (R := R + (x, next(x)))$;

$K = (L := L + sub_p(s, x) <> BSPA(G(X), x, t))$;

$G = (g := q(p))$;
$D = (p := parent(p))$;
$Q = (x := x_g)$.

Предикати:

$\alpha = (L \neq 0 \wedge k \neq K)$; $\gamma = (x \neq first(p))$;
$\chi = (q(p) = g)$;
$\beta = (x \neq d_{-1}(p))$;
$\varepsilon = (x \neq last(p))$.

Схема 2:

Видалення спільних ребер з множини R:

$$CR = G * \{D * Q * R\}_{\chi}. \quad (2.1)$$

Відтворення видалених спільних ребер в множині R :

$$\overline{CR}. \quad (2.2)$$

Початкове видалення вузлів шляху p з множини X :

$$CN = F*_{\varepsilon} \{C * N\}. \quad (2.3)$$

Відтворення вузлів шляху p, спільних з батьківським :

$$\overline{CN} = F*_{\beta} \{I * N\}. \quad (2.4)$$

Пошук дочірніх шляхів, отриманих як відхилення від батьківського:

$$DEV = L*_{\beta} \{I * R * K * E * P\}. \quad (2.5)$$

Формула (2.1) описує один із способів видалення ребер, спільних із попередньо визначеними найкоротшими шляхами, який базується на тому, що алгоритм Йена є девіаційним алгоритмом, а отже кожен наступний шлях отримується як відгалуження від попереднього.

Тоді остаточна регулярна схема алгоритму Йена виглядатиме :

$$YEN = S*_{\alpha} \{A * CR * CN * * DEV * \overline{CN} * \overline{CR}\}. \quad (2.6)$$

3. Аналіз методів розпаралелювання алгоритму Йена для систем із спільною пам'яттю

Описана в п.1 структура алгоритму Йена вказує на два можливі підходи до розпаралелювання алгоритму:

- паралельна реалізація базового алгоритму;
- реалізація паралельного пошуку дочірніх найкоротших шляхів для поточного шляху.

Кожен із даних методів має працювати із спільними для всіх паралельних гілок множинами X і R, що визначає архітектуру обчислювальної системи, що задовольняє вимогам алгоритму, як систему із спільною пам'яттю.

3.1. Паралельна реалізація базового алгоритму. Основне навантаження в алгоритмі Йена створює алгоритм пошуку найкоротшого шляху, тому доцільно при аналізі паралельної реалізації алгоритму розглянути паралельну реалізацію базового алгоритму (рис. 2). Одним з найбільш продуктивних алгоритмів, що реалізує базовий алгоритм є алгоритм Дейкстри, який розв'язує задачу пошуку найкоротших шляхів із заданої вершини у зваженому графі з невід'ємними ребрами до всіх інших вершин. Він знайшов широке застосування в мережевих технологіях: використовується в протоколі маршрутизації OSPF (*Open Shortest Path First*). В основі роботи алгоритму лежать операції з неспадною чергою з пріоритетами [6]. Складність цих операцій визначається реалізацією черги з пріорітетами. Можливі реалізації – лінійний масив, біноміальна піраміда, Фібоначеева піраміда. Реалізація при представленні у вигляді біноміальної піраміди має складність $O((m+n)\ln n)$, m – потужність множини R. Відповідна складність алгоритму Йена буде $O(Kn((m+n)\ln n))$. Алгоритм Йена відноситься до задач пошуку найкоротшого шляху між двома заданими вершинами водночас як алгоритм Дейкстри є алгорит-

мом, який дозволяє знайти найкоротші шляхи від заданої вершини до всіх інших вершин графа.

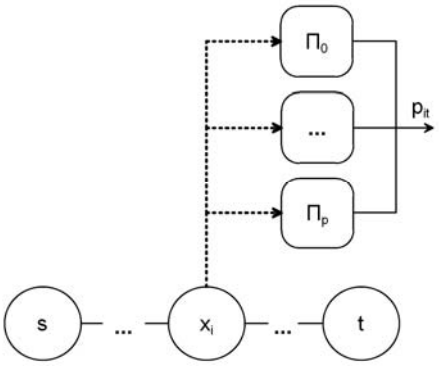


Рис. 2. Паралельний пошук найкоротшого шляху між x_i та t; $П_0 ... П_р$ – доступні паралельні процесори

Тому в контексті розглядуваної задачі умова в регулярній схемі алгоритму Дейкстри, описаний в [7] $\delta = (L \neq V)$ перейде в умову $\delta^{'} = (x \neq t)$, тобто пошук буде обриватися при досягненні кінцевої вершини.

Описаний у [7, 8] метод паралельної реалізації алгоритму Дейкстри базується на розбитті початкової черги на p паралельно оброблюваних черг, де величина p визначається кількістю паралельно запущених гілок виконання. Складність описаної у [7] паралельної реалізації алгоритму становить $O\left(\left(\frac{m}{p}+n\right)ln\frac{n}{p}\right)$, де $p-$ кількість паралельно виконуваних гілок. Відповідна складність алгоритму Йена – $O\left(Kn\left(\left(\frac{m}{p}+n\right)ln\frac{n}{p}\right)\right)$.

3.2. Паралельний пошук дочірніх найкоротших шляхів. Даний метод полягає в тому, щоб для k-го найкоротшого шляху асинхронно проводити пошук дочірніх. Даний підхід безпосередньо опирається на структуру алгоритму Йена, тому саме така методика може бути названа паралельною реалізацією алгоритму Йена. Фактично підхід полягає у паралельному виконанні ітерацій циклу, що містить базовий алгоритм для кожної вершини, що входить у батьківський шлях (рис. 3). Розглянемо батьківський шлях p, який представляє собою вектор вершин послідовні переходи між якими і утворюють шлях.

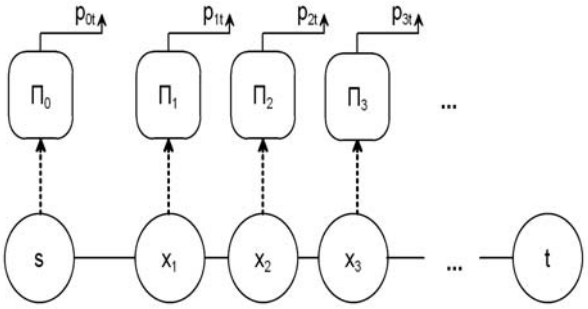


Рис. 3. Паралельний пошук дочірніх шляхів

Розглянемо два вузли, які мають у масиві p порядкові номери $i, j : i < j$. Для пошуку ациклічного дочірнього шляху, отриманого відхиленням у вершині j, необхідна відсутність у множині X вершин, наявних у шляху p під номерами $(1,\ldots j-1)$, тобто базовий алгоритм на даному кроці ітерації має працювати з множиною $X_j^{'} = X - \{sub_p(s, p_{j-1})\}$, аналогічна ситуація виникає і з i-м відхиленням, яке працює із множиною $X_i^{'}$, причому $X_i^{'} \setminus X_j^{'} \neq 0$.

Граф G представляється у вигляді квадратної матриці A: n x n, яка носить назву матриці суміжності, елементи якої можна задати у вигляді предиката $a_{i,j} = (x_j \in Rx_i)$, кожному з яких поставлено у відповідність числа $w_{i,j}$. У такому випадку *i*-й рядок відповідає за вихідні для ребра x_i, стовпець відповідно за вхідні. Фактично множина вершин X проектується на рядки і стовпці матриці A. Саме із таким представленням графа працює базовий алгоритм.

Видалення вузла із множини X досягається видаленням ребер із множини R, а відповідно стовпця із матриці A, які є вхідними за відношенням до нього.

Отже, на перший погляд здається, що ітерації не можуть працювати із однією

спільною множиною X, оскільки виконують модифікацію множини R : для кожної із них необхідно утворювати власну, несумісну із множиною іншої ітерації множину $X_j^{'}$, а відповідно із нею і множину $R_j^{'}$, що створюватиме високі накладні витрати на створення копій. Аналогічна проблема виникає і для спільних із батьківським шляхом ребер (x_i^k, x_{i+1}^k).

Рішення побудуємо за рахунок зміни порядку звертання до елементів матриці A. Процес видалення вузлів з матриці суміжності представлений на (рис. 4).

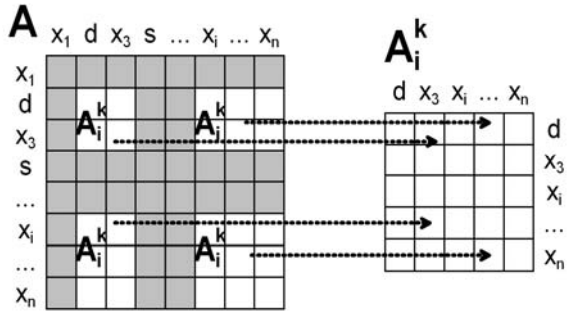


Рис. 4. Процедура видалення вузлів з початкової матриці суміжності для k- го батьківського шляху на i- ій ітерації

Видалення вузлів з матриці суміжності на i-му кроці ітерації еквівалентно утворенню нової квадратної матриці суміжності A_i^k, що складається із всіх вузлів, що не належать шляху $sub_{p_k}(s, x_{i-1}^k)$. Розмірність такої матриці буде $m_i^k = n - c(sub_{p_k}(s, x_{i-1}^k))$. Дану матрицю можна формально виділити з матриці A, шляхом створення шаблону звернення до елементів $\overline{sub_{p_k}(s, x_{i-1}^k)}$ (доповнення множини $sub_{p_k}(s, x_{i-1}^k)$ до X), який по суті є множиною вершин X_i^k. Звернення до ребра матриці A_i^k – r_{ab}^{ik} відбуватиметься як $r_{ab}^{ik} = r_{X_k^i[a] X_k^i[b]}$. Утворимо загальний шаблон, який можна використовувати для побудови необхідного індивідуального шаблону для певного кроку ітерації. Структура шаблону має вигляд $T^k = \{X \setminus p_k, p_k\}$ і допускає вільний порядок входження елементів множини $X \setminus p_k$. Множина p_k має зберігати порядок заданий порядком вершин у вихідному шляху, це дозволяє i-ій працювати з множиною X_i^k, а $i+1$ ітерація з множиною $X_{i+1}^k = X_i^k + \{x_{i+1}^k\}$. Приклад шаблону показано на (рис. 5).

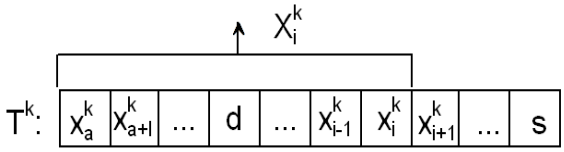


Рис. 5. Утворення шаблону для i-ї ітерації k-го шляху

Алгоритм Йена вимагає видалення на i-у кроці ітерації ребра (x_i^k, x_{i-1}^k) для забезпечення унікальності дочірнього та уникнення зациклювань. Тут також наявний зв'язок за даними між ітераціями. Але він може бути просто ліквідований. Алгоритм Дейкстри для i-й ітерації шукатиме шлях між від x_i^k до d і у даному шляху має бути відсутнім ребро (x_i^k, x_{i-1}^k). Алгоритм Дейкстри на початковому етапі присвоює всім вершинам у графі значення досяжності з точки x_i^k рівним ∞, а далі перебираючи ребра, які йдуть до цих вершин проводить процедуру релаксації [4], яка в кінцевому випадку полягає в зменшенні величини досяжності. Якщо на першій ітерації алгоритму Дейкстри розглянути лише вершини шаблону X_i^k за виключенням x_{i-1}^k, то в такий спосіб можна буде проігнорувати ребро (x_i^k, x_{i-11}^k) всередині ітерації.

У такий спосіб вдається незалежно користуватися єдиною матрицею A для кожного паралельного процесора, пов'язаного із відповідними ітераціями і тим самим створити умови для розробки паралельної схеми алгоритму. Складність реалізації становитиме $O\left(K\frac{n}{p}(m + n\ln n)\right)$.

4. САА-М схема алгоритму Йена для SMP - архітектури

САА-М схема алгоритму Йена формуватиметься на основі формули (2.4). Додамо нові елементи до інформаційної множини:

- T – загальний шаблон для поточного батьківського шляху.
- IT – індивідуальний шаблон для поточної ітерації пошуку дочірнього шляху.

Додамо до множини операторів оператори:

$$M = (T = \{p / X\} \cup p), \qquad (4.1)$$

$$IM = (IT = Form(T, x)), \qquad (4.2)$$

де операція $Form(T, x)$ формує індивідуальний шаблон для поточної ітерації, яка визначається поточним елементом x.

$$K^{/} = (\mathrm{L} := \mathrm{L} + sub_p(s, x) <> BSPA(\mathrm{G}(\mathrm{IT}), x, t))$$

Тоді можна записати:

$$M * DEV = M * A*_{\beta} \{IM * K^{/} * P\} \quad (4.3)$$

Задля реалізації статичного розподілу ітерації між паралельними процесорами введемо для кожного із них величини, що ідентифікують індекси елементів в масиві p :

$$first_i(p) = p_{c(p)} - p_{i*n},$$
$$last_i(p) = p_{c(p)} - p_{(i+1)*n} + 1, \qquad (4.4)$$

де i – номер паралельної гілки,

$n = \dfrac{c(p) - q(p)}{num}$ – загальна вузлів найкоротші шляхи між якими необхідно буде знайти, i – номер паралельної гілки, num – – кількість доступних паралельних гілок.

Введемо деяку умову $\beta_i = (x_i \neq last_i(p))$, де x_i визначають поточний елемент для i-ї паралельної гілки. Тоді предикат β може бути переписаний як $\beta = \beta_1 \wedge \beta_2 \wedge \ldots \wedge \beta_i \wedge \ldots \wedge \beta_p$.

Введемо допоміжний оператор:

$$CC_i = A*_{\beta_i} \{IM * K^{/} * P\} \qquad (4.5)$$

Тоді (3.3) можна представити як сукупність асинхронних диз'юнкції:

$$M * DEV = M * CC_1 \vee \ldots CC_i \vee \ldots CC_{num} \quad (4.6)$$

Формула (4.6) ілюструє паралельну частину САА-М алгоритму Йена. Тоді повна схема алгоритму виглядатиме :

$$\mathrm{YEN} = \mathrm{S}*_{\alpha} \{\mathrm{A}*\mathrm{CR}*\mathrm{M}*CC_1 \vee$$
$$\vee \ldots CC_i \vee \ldots CC_{num} * \overline{\mathrm{CR}}\}.$$

5. Технологія OpenMP

При роботі з *SMP* системами використовується так звана парадигма програмування з пам'яттю, що розділяється (*shared memory paradigm*), яка базується на природному використанні паралельних потоків, які мають спільну область пам'яті під код та дані. *POSIX* – інтерфейс для створення потоків – *Pthreads* [9] підтримується майже на всіх *UNIX* – системах (схожий за функціональними можливостями інтерфейс підтримують і *Windows* платформи – *Windows Threads)*, але такий інтерфейс більш трудомісткий для практичного паралельного програмування оскільки має дуже низький рівень програмування. Високорівневою надстройкою над *Pthreads* (та аналогічними бібліотеками) для *SMP* платформ, що масштабуються, на сьогоднішній день де-факто стала технологія програмування *OpenMP* (*Open Multi-Processing*)[10], що активно підтримується компанією *Intel*. У даній технології використовується модель програмування близька до *Pthreads*: динамічне породження потоків, спільні та розділювані дані, механізм "замків" для синхронізації. Для написання програм, що підтримують стандарт *OpenMP* зі специфікаціями набору директив компілятора, службових функцій і змінних середовища, використовуються спеціальні компілятори.

Основні переваги технології *OpenMP:*

- розробник не створює нову паралельну програму, а додає в текст послідовної програми директиви OpenMP;
- передбачається, що OpenMP – програма на однопроцесорній платформі може бути використана у якості послідовної программ. Директиви компілятора ігноруватимуться
- гнучкість за рахунок можливості використання низькорівневих механізмів.

Абстрактність інтерфейсу OpenMP дозволяє використовувати для алгоритмів природні поняття паралельного виконання операцій і дозволяє з легкістю провести формальний аналіз.

6. Експериментальні результати

З метою підтвердження ефективності роботи паралельних реалізацій алгоритму його було застосовано до щільно заповненого графа з щільністю 50% ($m \approx n \times n$). Складність у такому випадку буде для достатньо великих розмірностей – $O(Kn^3)$, а прогнозована складність паралельної версії $O\left(K\frac{n^3}{p}\right)$, p – кількість доступних паралельних процесорів.

Результати були отримані для випадкових графів розмірностей:
n = {50, 100, 250, 500, 1000, 1500,2500, 3500, 4500, 5500, 6500, 7500};
для кількості шляхів:
K = {10, 20, 50, 100, 200, 300, 400, 500};

Для того, щоб забезпечити однорідність розглядуваних графів, використовувалася одна матриця максимальної розмірності на основі якої формувалися підматриці меншої розмірності у спосіб, що показано на рис. 6.

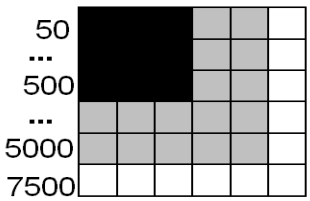


Рис. 6. Методика побудови підматриць

Часи виконання послідовної версії програми в залежності від кількості вузлів у графі й K показано на рис. 7 та 8.

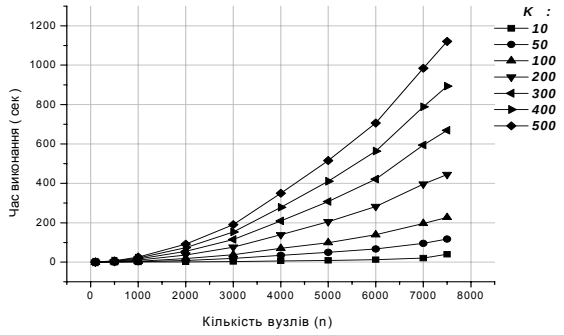


Рис. 7. Часи виконання послідовної версії алгоритму в залежності від кількості вузлів у матрицях для різних К

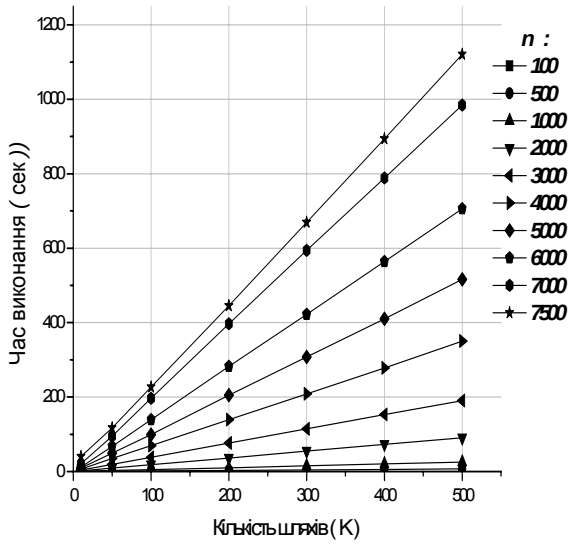


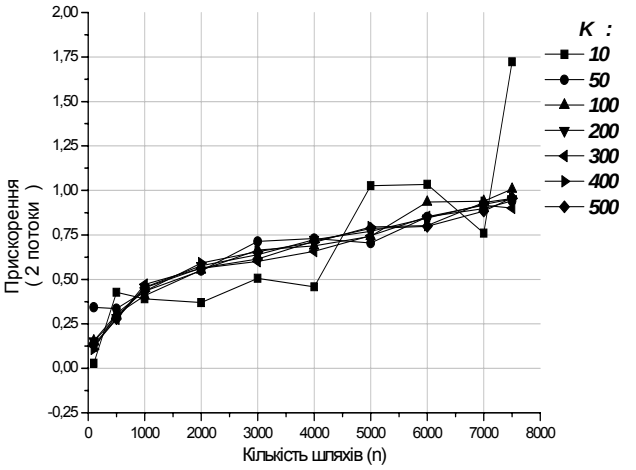
Рис. 8. Часи виконання послідовної версії алгоритму в залежності від кількості шуканих шляхів для матриць різних розмірностей

З графіків видно збігаюче з теорією лінійний закон зростання складності від кількості шуканих шляхів, та поліноміальний закон складності в залежності від розмірності матриці.

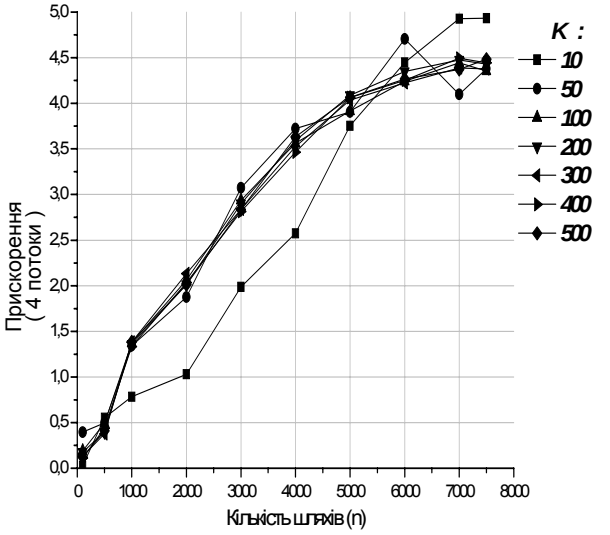
Паралельні реалізації тестувалися на вузлі кластеру КНУ ім. Т. Шевченка [11]. Аналіз проводився шляхом порівняння результатів послідовної та паралельної версій. Залежності отриманих прискорень від розмірностей матриць для різних K показано на рис. 9, а, б, та рис. 10, а, б.

Порівняння графіків прискорення для двох паралельних реалізацій дозволяє стверджувати, що у версії з паралелізацією базового алгоритму вихід на максимальне прискорення відбувається повільно і досягається лише на матрицях великих розмірностей водночас як у алгоритмі з масками прискорення мало залежить від розмірності матриці суміжності. Це пояснюється тим, що у алгоритмі Дейкстри час витрачається на створення та впорядкування p черг [8], які необхідні для незалежної роботи паралельних процесорів, водночас як для алгоритму з масками всі підготовчі (створення масок, видалення спільних з попередніми шляхами ребер) операції виконуються до початку ітерацій.

Паралельна реалізація вимагає додаткових витрат часу на синхронізацію потоків між собою. Звернення до засобів синхронізації вимагає використання системних викликів, що вимагають значного часу. Кількість синхронізуючих елементів напряму залежить від рівня вкладеності блоків коду, що підлягають розпаралелюванню.



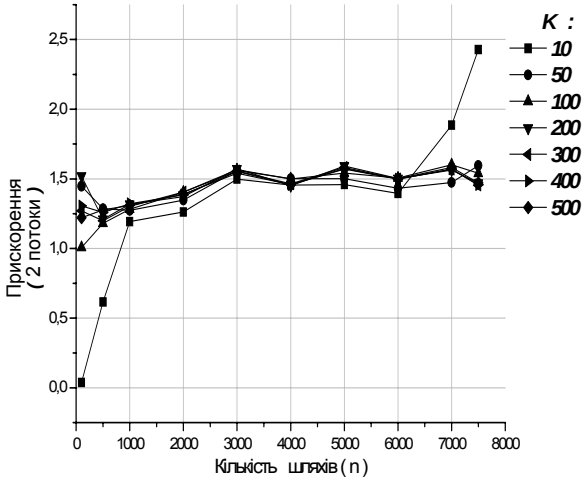
а



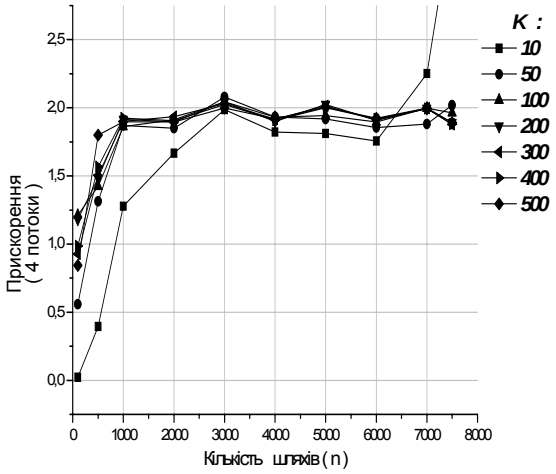
б

Рис. 9. Прискорення отримані для версій з використанням паралельної реалізації алгоритму Дейкстри:
а – для двох потоків;
б – для чотирьох потоків

Так підхід з використанням масок є більш високорівневим, оскільки проводить напряму паралелізацію ітерацій водночас як паралельна реалізація базового алгоритму виконується всередині окремої ітерації для внутрішніх ітерацій алгоритму Дейкстри.



а



б

Рис. 10. Прискорення отримані для версій з використанням шаблонів: а – для двох потоків;
б – для чотирьох потоків

В останньому випадку засобів синхронізації буде більше, а отже, більшим буде програш за часом. При цьому чим

більш низькорівнева паралелізація тим рівномірніше навантаження на паралельні процесори і відповідно менший час очікування завершення операцій всіма процесорами. Тому в даному сенсі метод шаблонів програє реалізації з паралельним алгоритмом Дейкстри.

Структура запропонованого паралельного пошуку дочірніх шляхів також вимагає накладних витрат на етапі виконання ітерації, які полягають у використанні адресації матриці суміжності за шаблоном, тобто непряме звертання до елементів матриці, що вимагає накладних витрат.

З рис. 9, 10 видно, що в проведених експериментах максимальні прискорення склали:

- для двох потоків методу шаблонів – 1.6;
- для чотирьох потоків методу базового алгоритму – 4.4 рази.

Можна стверджувати, що алгоритм Йена допускає ефективну паралелізацію для SMP – систем, але вимагає диференційованого підходу до застосування різних схем паралелізації.

Висновки

1. Побудовано регулярну схему алгоритму Йена ($O(Kn^3)$) з використанням апарату САА, яка є вихідною для формування паралельних версій.

2. Запропоновано два підходи до паралельної реалізації алгоритму Йена для SMP – архітектур. Перший – паралельна реалізація базового алгоритму пошуку найкоротшого шляху. Для другого розроблено метод шаблонів, який дозволяє провести асинхронний пошук дочірніх шляхів для заданого батьківського.

3. Проведено оцінки складностей запропонованих рішень. Перший підхід дає теоретичну складність $O\left(Kn\left(\left(\frac{m}{p}+n\right)ln\frac{n}{p}\right)\right)$, другий – $O\left(K\frac{n}{p}\left(m+n\,ln\,n\right)\right)$.

4. Виконано формалізацію запропонованого методу шаблонів шляхом формування паралельних схем з використанням математичного апарата САА–М.

5. Реалізовано паралельный алгоритм Йена, використовуючи метод шаблонів і метод розпраралелювання базового алгоритму, у якості якого було взято реалізацію алгоритму Дейкстри із складністю $O(m+n\,ln\,n)$.

6. Отримані експериментальні дані вказують на суттєвий приріст у швидкодії паралельних реалізацій у порівнянні з послідовною версією.

7. Паралельні реалізації алгоритму вимагають диференційованого підходу у застосуванні до архітектур із різною кількістю ядер. На двопроцесорній системі ефективнішим виявляється застосування методу шаблонів, а на чотирьохпроцесорній – метод паралельного базового алгоритму.

8. В обох паралельних схемах присутні накладні витрати, які призводять до відхилень прискорень від теоретично очікуваних.

1. *Brander A.W., Sinclair M.C.* A comparative study of k-shortest path algorithms// In Proc. of 11th UK Performance Engineering Workshop. – 1995.
2. *O'Mahony M.J, Sinclair M.C., Mikac B.* Ultra-high capacity optical transmission network. European research project COST 239 // Information, Telecommunications, Autonata. – 1993. – 12. – P 33–45.
3. *Кристофидес Н. Теория графов.* – М.: Мир, 1978. – 432 с.
4. *Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П. и др*. Многоуровневое структурное проектирование программ. Теоретические основы, интсрументарий. – М.: Финансы и статистика, 1989. – 342 с.
5. *Martins E., Pascoal M.* A new implementation of Yen's ranking loopless paths algorithm. – 2000. – 13 с.
6. *Кормен Т., Лейзерсон Х., Чарльз И., Штайн Л.* Алгоритмы: построение и анализ, 2-е издание : Пер. с англ. – М.: «Вильямс», 2005. – 1296 с.
7. *Погорілий С.Д., Бойко Ю.В., Білоус Р.В.* Формування та аналіз паралельних схем алгоритму Дейкстри // Математичні машини і системи. – 2008. – № 4. – С. 62 – 72.
8. *Ефимов С.С.* Обзор методов распараллеливания алгоритмов решения некоторых за-

дач вычислительной дискретной математики // Математические структуры и моделирование. – М.: Мир, 2007. – № 17. – С. 72–93.
9. *POSIX* Threads Programming [Електронний ресурс] / Blaise Barney, Lawrence Livermo-?e National Laboratory – Режим доступу https://computing.llnl.gov/tutorials/pthreads/
10. *The* OpenMP API specification for parallel programming [Електронний ресурс] – Режим доступу http://openmp.org/
11. *Марьяновский В.А., Погорелый С.Д., Бойко Ю.В, и др.* Программное обеспечение UAClaster // Управляющиесистемы и машины. – 2009. – № 5. – С. 76–80.

Отримано 08.12.2010

Об авторах:

Погорілий Сергій Дем'янович,
доктор технічних наук,
професор кафедри комп'ютерної інженерії радіофізичного факультету,

Бойко Юрій Володимирович,
кандидат фізико-математичних наук,
начальник інформаційно-обчислювального центру, зав. кафедри комп'ютерної інженерії радіофізичного факультету,

Трібрат Михайло Ігорович,
аспірант кафедри комп'ютерної інженерії радіофізичного факультету,

Афанасьєв Дмитро Володимирович,
студент магістратури
кафедри комп'ютерної інженерії
радіофізичного факультету.

Місце роботи авторів:

Київський національний університет
імені Тараса Шевченка,
01601, Київ-601,
вул. Володимирська, 60.
Тел.: (044)2590519,
e-mail: sdp@univ.kiev.ua,
mike3b@univ.kiev.ua,
boyko@univ.kiev.ua,
afon_rff@univ.kiev.ua