УДК 004.4

С.Д. Погорілий, д. т.н., проф.,
С.О. Комісарук, студент
Київський національний університет імені Тараса Шевченка
sdp77@i.ua, svitlanakomisaruk@gmail.com

Формування та дослідження паралельних схем алгоритму Йена

Виконано формалізацію алгоритму Йена пошуку k найкоротших шляхів у зваженому неорієнтованому графі з використанням математичного апарату модифікованих систем алгоритмічних алгебр В.М. Глушкова. Запропоновано концепцію розпаралелювання алгоритму для архітектур зі спільною пам'яттю, що ґрунтується на мінімізації витрат на паралельну обробку та синхронізацію даних. Проведено трансформацію схеми алгоритму та одержано паралельну схему. Виконано експериментальне порівняння швидкодії послідовної та паралельної схем з використанням кластерних обчислень.

Ключові слова: маршрутизація, алгоритм Йена, граф, матриця суміжності, k найкоротших шляхів, система алгоритмічних алгебр, САА-М схема, розпаралелювання, потік, синхронізація, контрольна точка, асинхронна диз'юнкція, OpenMP-технологія.

Вступ

На сьогодні, системи та пристрої телекомунікації стрімко розвиваються. Одним із завдань їх проектування та адміністрування є налаштування маршрутизаторів, що здійснюють передачу інформації в пакетному режимі, у яких доцільно та актуально застосовувати задачі пошуку найкоротших шляхів у графах.

Протокол маршрутної інформації (RIP – Routing Information Protocol) – один із найпростіших протоколів дистанційно-векторної маршрутизації, який оперує транзитними ділянками в якості метрики маршрутизації. У сучасних мережевих середовищах RIP не є найкращим рішенням, оскільки його можливості поступають таким більш сучасним протоколам як EIGRP (Enhanced Interior Gateway Routing Protocol), OSPF (Open Shortest Path First) та ін. Проте перевагою протоколу RIP є простота конфігурування.

OSPF – протокол стану каналу (link-state). Кожен маршрутизатор приймає інформацію про стан каналу та на її основі будує повну таблицю маршрутизації. Цей протокол працює швидше, ніж RIP, оскільки використовує безпосередньо IP. EIGRP є вдосконаленим внутрішнім протоколом шлюзів, який добре масштабується і дозволяє забезпечити малий час конвергенції при мінімальному мережевому трафіку.

У мережах з великою кількістю альтернативних маршрутів спостерігається неконтрольоване розщеплення потоку між кожною парою джерело-одержувач уздовж множини знайдених маршрутів. Для усунення такої проблеми необхідна розробка алгоритмів пошуку оптимальних маршрутів, що дозволяють ввести обмеження на число паралельних маршрутів. Використовуючи модифіковану систему алгоритмічних алгебр В.М. Глушкова, сформуємо паралельні схеми алгоритму Йена, що використовуються для розрахунку оптимальних маршрутів, та проведемо дослідження їх програмних реалізацій.

Опис алгоритму

Одним із найефективніших методів пошуку k найкоротших простих шляхів є алгоритм Йена.

Алгоритм починає роботу зі знаходження найкоротшого шляху P^1 за допомогою алгоритму Дейкстри. Далі шукають наступний оптимальний шлях P^k для $k = 2$. Для цього знаходиться набір кращих відхилень від заданого шляху, використовуючи на кожному кроці алгоритм Дейкстри.

Далі шукається набір кращих відхилень від заданого шляху. Найкоротший серед знайдених кандидатів P_i^k і буде P^k. Далі переходять до пошуку P^k для $k = 3$ і т. д.

Введемо низку позначень:

N – розмірність графа;
G – зважений орієнтований граф, заданий у вигляді матриці суміжностей;
k – кількість найкоротших шляхів;
H – матриця суміжностей;
L – список k найкоротших шляхів з вершини s у вершину t;
P – список кандидатів на найкоротший шлях з s до t;
A – найкоротший маршрут шляху з s у t;
M – масив значень довжин шляхів;
W – матриця маршрутів;
f – номер першої порожньої матриці в L;

e – номер останньої заповненої матриці в *P;*
m – мінімальний з кандидатів P^i;
F – масив, у якому елемент F_i приймає значення *false*, якщо відстань тимчасова, *true* – постійна*;*
Dijkstr(G, N, F, M, s, t, V, W, A) – алгоритм Дейкстри пошуку найкоротшого шляху з вершини *s* у *t*;
addKandidate(L, P, f, e, N, F, M, s, t, V, W, A) – додавання в список *L* кандидатів ;
setShortest(L, P, f, N) – встановлення *k* найкоротших шляхів.

Роботу алгоритму можна розділити на 3 послідовні кроки:

1. Пошук 1-го найкоротшого шляху за допомогою алгоритму Дейкстри.

2. Пошук всіх відхилень P_i^k *(k-1)* найкоротшого шляху P_i^{k-1} для всіх *i* = 1, 2,…, q_{k-1}.

3. Переміщення найкоротшого шляху з P_i в L_i, позначивши L_{i+1} символом *f*. Якщо *f = (k-1)*, алгоритм закінчує роботу та *L* містить потрібний список *k* найкоротших шляхів.

Послідовна схема алгоритму

Формалізуємо алгоритм Йена шляхом формування схеми з використанням системи алгоритмічних алгебр Глушкова (САА). Фіксована САА являє собою двоосновну алгебраїчну систему, основами якої є множина операторів і множина умов. Операції САА поділяються на логічні та операторні. До логічних відносяться узагальнені булеві операції і операції лівого множення оператора на умову, а до операторних – основні конструкції структурного програмування (послідовне та циклічне виконання, тощо).

Кожен алгоритм може мати певну інтерпретацію у вигляді САА-схеми. Тобто, якщо визначити основи САА для конкретного алгоритму, можна представити цей алгоритм у вигляді схеми і проводити подальші трансформації й оптимізації вже не з алгоритмом, а з його САА-схемою.

Для скорочення запису формули алгоритму, використовуючи позначення, запишемо умови:

$$\alpha_1(L) = (L_{000} == 0);$$

$$\alpha_2(k, N, f, e) = ((f != k) \Lambda (e > -2) \Lambda (e < N \times N \times 2)).$$

Умова α_1 вказує на відсутність шляху L_{ijk} між вершинами *s* та *t*, а α_2 – на знаходження *(k-1)* найкоротшого шляху.

Додавання кандидатів у *P*:

$$\alpha_3(c, N, L, f) = ((c < N-1) \Lambda (L_{(f-1)0(c+1)} != 0)).$$

Булеві вирази, які визначають, чи повторюється виконання інструкцій всередині циклу:

$$\alpha_4(N, i) = (i < N \times N \times 2);$$

$$\alpha_5(N, i) = (i < N);$$

$$\alpha_6(N, j) = (j < N);$$

$$\alpha_7(N, i) = (i < N + 1);$$

$$\alpha_8(N, j) = (j < N + 1);$$

$$\alpha_{15}(N, k) = (k < N + 1);$$

$$\alpha_{16}(k, i) = (i < k).$$

Визначення мінімального із кандидатів:

$$\alpha_9(P, N, m, i) = (P_{i0N} < P_{m0N}).$$

Перевірка на збіг шляхів в *L*:

$$\alpha_{10}(L, f, N, D) = (L_{(f-1)0N} != D_N);$$

$$\alpha_{11}(L, f, d, N, D) = ((f - d) >= 0) \wedge$$
$$\wedge (L_{(f-d)0N} == D_N));$$

$$\alpha_{12}(L, f, d, D, i) = (L_{(f-d)0i} != D_i);$$

$$\alpha_{13}(r, c) = (r == c - 1).$$

Перепис в *L* з номером *f*:

$$\alpha_{14}(b) = (b == 1).$$

Сформуємо операнди:

$$A_1 = (m = 0);$$

$$A_2 = (m = i);$$

$$A_3 = (D_i = P_{m0i});$$

$$A_4 = (b = 0;\ d = 1);$$

$$A_5 = (b = 1);$$

$$A_6 = (r = 0);$$

$$A_7 = (b_1 = 0);$$

$$A_8 = (b_1 = 1);$$

$$A_9 = (r = r + b_1);$$

$$A_{10} = (L_{fij} = P_{mij});$$

$$A_{11} = (A_{ij} = 0);$$

$$A_{12} = ((P_{(e+1)ij} = L_{(f-1)ij});\ (A_{(i-1)j} = L_{(f-1)ij}));$$

$$A_{13} = ((P_{(e+1)L_{(f-1)0e}(L_{(f-1)0(e+1)}-1)} = 65535);$$
$$(P_{(e+1)L_{(f-1)0(e+1)}(L_{(f-1)0e}-1)} = 65535);$$
$$(A_{(L_{(f-1)0e}-1)(L_{(f-1)0(e+1)}-1)} = 65535);$$
$$(A_{(L_{(f-1)0(e+1)}-1)(L_{(f-1)0(e)}-1)} = 65535);$$

$(P_{(e+1)0N} = Dijkstr(H,N,F,M,s,t,V,W,A)));$

$A_{14} = (P_{(e+1)0i} = A_i);$

$A_{15} = (L_{00N} = Dijkstr(H,N,F,M,s,t,V,W,A));$

$A_{16} = (L_{00i} = A_i);$

$A_{17} = (L_{0ij} = G_{(i-1)j});$

$A_{18} = (output\ "There\ is\ no\ path");$

$A_{19} = ((e = -1);\ (f = 1));$

$A_{20} = ((t_1 = addKandidate(L,P,f,e,N,F,M,$
$s,t,V,W,A));\ (e = t_1);\ (t_3 = f);$
$(t_2 = setShortest(L,P,f,N));(f = t_2)).$

Ініціалізуємо масиви *P* та *L* з метою подальшої коректної роботи алгоритму:

$A_{21} = (L_{ijk} = 0; P_{ijk} = 0);$

$A_{22} = (P_{ijk} = 0);$

$A_{23} = (L_{i0N} = 65535; P_{i0N} = 65535);$

$A_{24} = (P_{i0N} = 65535).$

Оперуючи умовами та операндами, запишемо кожен із кроків алгоритму:

$$setShortest(L,P,f,N) = A_1 * \{ \underset{\alpha_4}{(} A_2 \underset{\alpha_9}{)} \times$$
$$\times(++i)\} * \{ \underset{\alpha_7}{A_3} \times(++i)\} * A_4 * \underset{\alpha_{10}}{(} A_5 \vee$$
$$\vee(A_6 * \{ \underset{\alpha_{11}}{A_7} * \{ \underset{\alpha_5}{(} A_8 \underset{\alpha_{12}}{)} \times(++i)\} * A_9 \times(++d)\} *$$
$$* \underset{\alpha_{13}}{(} A_5)))* \underset{\alpha_{14}}{(} \{ \underset{\alpha_7}{\{} \underset{\alpha_8}{A_{10}} \times(++j)\} \times(++i)\} \times$$
$$\times(++f));$$

$$addKandidate(L,P,f,e,N,F,M,s,t,V,$$
$$W,A) = \{ \underset{\alpha_3}{\{} \underset{\alpha_5}{\{} \underset{\alpha_6}{\{} A_{11} \times(++j)\} \times(++i)\} *$$
$$* \{ \underset{\alpha_7}{\{} A_{12} \times(++j)\} \times(++i)\}\} * A_{13} * \{ \underset{\alpha_5}{A_{14}} \times$$
$$\times(++i)\} \times(++e) \times(++c)\}.$$

Сформуємо послідовну регулярну схему (ПРС) алгоритму Йена:

$$Yen(G,N,A,F,M,s,t,V,W) = \quad (1)$$
$$= \{ \underset{\alpha_4}{\{} \underset{\alpha_8}{\{} \underset{\alpha_9}{\{} \underset{\alpha_{10}}{(} A_{21} \vee A_{22} \underset{\alpha_{10}}{)} \times(++k)\} \times(++j)\} \times$$
$$\times(++i)\} * \{ \underset{\alpha_4}{(} \underset{\alpha_{10}}{(} A_{23} \vee A_{24} \underset{\alpha_{10}}{)} \times(++i)\} * A_{15} *$$
$$* \{ \underset{\alpha_5}{A_{16}} \times(++i)\} * \{ \underset{\alpha_7}{\{} \underset{\alpha_6}{A_{17}} \times(++j)\} \times$$
$$\times(++i)\} * \underset{\alpha_1}{(} A_{18} \vee (A_{19} * \{ \underset{\alpha_2}{A_{20}} \})\}.$$

Принцип обробки даних

Сформулюємо деякі засади роботи із даними, які розміщенні у пам'яті. Набір вхідних даних для алгоритму зберігається у вигляді матриці суміжності графа *G*, яка є незмінною до кінця його роботи. Інформація про список кандидатів в найкоротший шлях та *k* найкоротших шляхів зберігається у вигляді двох трьохвимірних масивів *P* та *L* відповідно.

Оскільки всі інші найкоротші шляхи $A^i_{s...t}$ не повинні співпадати з першим $A^0_{s...t}$, який знайдено за допомогою алгоритму Дейкстри в графі *G* , то вони не містять як мінімум одне із ребер першого шляху. Тому видаляємо по одному ребру із $A^0_{s...t}$ і знаходимо шляхи в отриманих графах, які задані матрицями суміжностей *H*. Знайдені шляхи з поміткою про те, яке ребро було видалено, додаємо в список кандидатів *P*. Із нього вибираємо найкоротший та переносимо в *L*.

Обробку даних алгоритму зображено на рисунках 1(а, б, в, г, д):

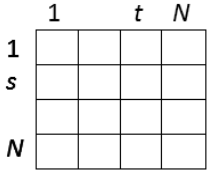
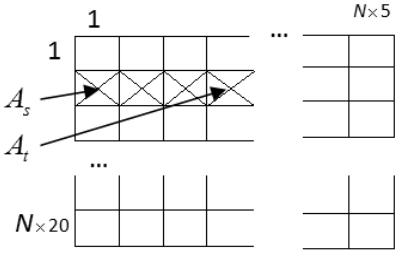
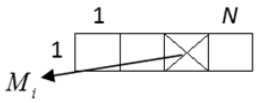


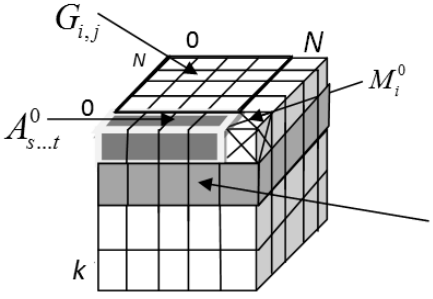
Рисунок 1а – Матриця суміжності G



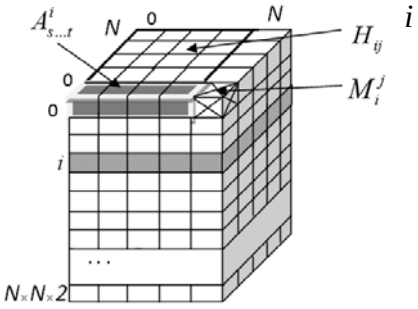
б – Матриця маршрутів W



в – Масив довжин шляхів M



г – Масив L найкоротших шляхів



д – Масив кандидатів P

Таким чином, спочатку L в 1-му шарі (i = 0) присвоюємо маршрут найкоротшого шляху $A^0_{s...t}$ у комірці з координатами (i = 0, j = 0, k = 0…N-1), довжину шляху (i = 0, j = 0, k = N) та матрицю суміжності G (i = 0, j = 1…N, k = 1…N). Потім у наступні шари із P переписуємо шляхи в L, поки i не буде дорівнювати k. Після виконання цієї умови робота алгоритму закінчується.

Формування паралельної схеми алгоритму Йена

Розбиття матриці даних. Розглянемо розпаралелювання алгоритму. Нехай потрібно розділити масив на M приблизно рівних частин прямокутної форми, як зображено на рисунку 2.

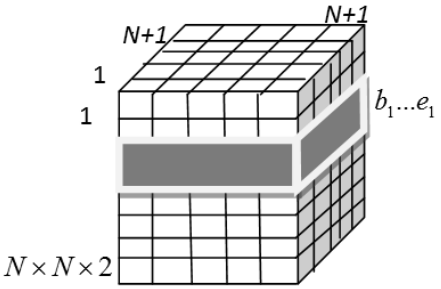


Рисунок 2 – Розбиття матриці даних

Візьмемо, наприклад, масив кандидатів.

Позначимо перший рядок i-ї частини через b_1, а останній – e_1, та обчислимо їх за такими співвідношеннями:

$$b_i^{N\times N\times 2,M} = (i-1)\times \left] M \middle/ N\times N\times 2 \right[+1,$$

$$e_i^{N\times N\times 2,M} = i\times \left] M \middle/ N\times N\times 2 \right[.$$

Таким чином, можна отримувати межі областей, які оброблятимуться одним із паралельних потоків.

Синхронізація потоків. З метою орієнтації САА на формалізацію паралельних обчислень, використовується модифікована систему алгоритмічних алгебр (САА-М) з розширеною сигнатурою операцій.

Синхронізація паралельних гілок здійснюється за допомогою контрольних точок і залежних від них синхронізаторів.

Контрольні точки являють собою фіксовані позиції між операторами. З кожною контрольною точкою T асоціюється умова u, яка хибна до тих пір, поки процес обчислень не досягнув точки T: істинна з моменту досягнення даної точки. Умова u називається умовою синхронізації, що асоціюється з точкою T, яка позначається $T(u)$.

Затримка обчислень у гілках реалізується за допомогою синхронізаторів. Синхронізатор визначається за допомогою циклу S(u) = {[u] E}, де u – логічна функція, яка залежить від умов синхронізації, зв'язаних з визначеними контрольними точками схеми. Синхронізатор $S(u)$, встановлений в деякому місці ПРС, здійснює затримку обчислень в даному місці з моменту, коли умова синхронізації u стане істинна.

Представлення алгоритмів в САА-М, які специфікують асинхронні операторні взаємодії, називаються паралельними регулярними схемами (ПРС).

Узагальнену точку синхронізації кількох паралельний потоків, що є, фактично, бар'єром, який затримує їх виконання, можна представити формулою:

$$B_N(i) = T(u_i)\times S(u_1)\times ...\times S(u_{i-1})\times S(u_{i+1})\times ...\times$$

$$\times S(u_N) = T(u_i)\times \prod_{\substack{j=1\\ j\neq i}}^{N} S(u_j),$$

де T(u) – контрольна точка;
S(u) – точка синхронізації;
N – кількість потоків;
i – номер потоку, яких синхронізується.

Механізм реалізації синхронізації зображено на рис. 3.

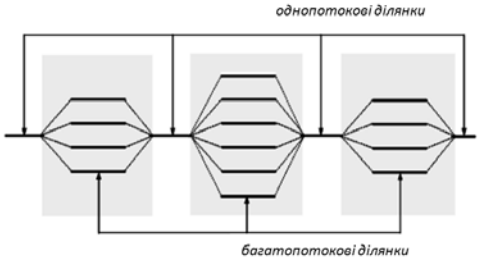


Рисунок 3 – Синхронізіція потоків

Таким чином, при ініціалізації набору паралельних процесів у програмі встановлюється контрольна точка, у якій очікується завершення всіх породжених паралельних процесів, але поки вони не завершать свою роботу, програма не може продовжувати працювати.

Еквівалентні перетворення схеми алгоритму. Застосуємо апарат тотожних перетворень САА-М до схеми (1).

До конструкцій САА-М, орієнтованих на асинхронні паралельні обчислення, відноситься асинхронна диз'юнкція $A \dot{\vee} B$ – бінарна операція, яка полягає в паралельному виконанні операторів A і B на різних підструктурах операційної

структури. Застосуємо її до послідовної регулярної схеми (1). Це дозволить розбити цикли на паралельні гілки, у яких n – кількість потоків:

$$setShortest(L,P,f,N) = A_1 * \{ \underset{\alpha_4}{} \overset{2n}{\bullet} \underset{a=1}{\vee} ((\underset{\alpha_9}{A_2})\times \quad (2)$$

$$\times(++i)\}) * \{ \underset{\alpha_7}{} \overset{2n}{\bullet} \underset{a=1}{\vee} (A_3 \times(++i)\}) * A_4 *$$

$$* (\underset{\alpha_{10}}{A_5} \vee (A_6 * \{ \underset{\alpha_{11}}{A_7} * \{ \underset{\alpha_5\ \alpha_{12}}{(} A_8)\times(++i)\} * A_9 \times$$

$$\times(++d)\} * (\underset{\alpha_{13}}{A_5}))) * (\underset{\alpha_{14}}{} \overset{2n}{\bullet} \underset{i=1}{\vee} (\{ \{ (\underset{\alpha_7\ \alpha_8}{A_{10}} \times$$

$$\times(++j)\}\times(++i)\}) \times(++f));$$

$$addKandidate(L,P,f,e,N,F,M,s,t,V, \quad (3)$$

$$W,A) = \{ \underset{\alpha_3}{} \overset{2n}{\bullet} \underset{a=1}{\vee} (\{ \{ \underset{\alpha_5\ \alpha_6}{A_{11}} \times(++j)\}\times(++i)\}) *$$

$$* \overset{2n}{\bullet} \underset{a=1}{\vee} (\{ \{ \underset{\alpha_7\ \ \alpha_6}{A_{12}} \times(++j)\}\times(++i)\}\}) * A_{13} *$$

$$* \{ \underset{\alpha_5}{A_{14}} \times(++i)\}\times(++e)\times(++c)\};$$

Підставляючи вирази (2) та (3) у (1), отримаємо:

$$Yen(G,N,A,F,M,s,t,V,W) = \overset{2n}{\bullet} \underset{a=1}{\vee} (\{ \{ \{ \quad (4)$$

$$(\underset{\alpha_{10}}{A_{21}} \vee A_{22})\times(++k)\}\times(++j)\}\times(++i)\}) *$$

$$* \overset{2n}{\bullet} \underset{a=1}{\vee} (\{ (\underset{\alpha_4\ \alpha_{10}}{} A_{23} \vee A_{24})\times(++i)\}) * A_{15} * \{ \underset{\alpha_5}{A_{16}} \times$$

$$\times(++i)\} * \{ \underset{\alpha_7\ \alpha_6}{} \{ \underset{}{A_{17}} \times(++j)\}\times(++i)\} *$$

$$* (\underset{\alpha_1}{A_{18}} \vee (A_{19} * \{ \underset{\alpha_2}{A_{20}}\})).$$

Отримана формула (4) – паралельна схема алгоритму Йена. Як бачимо, ітерації циклів розподіляються між потоками, і, як результат, виконані паралельно.

Аналіз програмних реалізацій схем алгоритму Йена

Технологія OpenMP. Одним із популярних засобів програмування для комп'ютерів із спільною пам'яттю, що базуються на традиційних мовах програмування і використанні спеціальних коментарів, є технологія OpenMP [11]. Її інтерфейс задуманий як стандарт для програмування на масштабованих SMP-системах (Symmetric Multiprocessing) у моделі спільної пам'яті. У стандарт OpenMP входять специфікації набору директив компілятора, допоміжних функцій і змінних середовища. OpenMP реалізує паралельне обчислення за допомогою багатопотоковості, у якій «головний» (master) потік створює набір «підлеглих» (slave) потоків, і задача розподіляється між ними. Вважається, що потоки виконуються паралельно на машині з декількома процесорами, причому кількість процесорів не обов'язково повинна співпадати з кількістю потоків.

Важливою особливістю технології OpenMP є можливість реалізації інкрементального програмування, тобто, поступового пошуку ділянок програми, які містять ресурс паралелізму. За допомогою механізмів, що надаються OpenMP, ділянки роблять паралельними. Такий підхід значно полегшує процес адаптації послідовних програм до паралельних комп'ютерів, а також налагодження і оптимізацію.

За допомогою OpenMP в послідовній версії алгоритму Йена було розподілено цикли, у яких ітерації виконуються тривалий час, між процесорними ресурсами комп'ютера. Було отримано час виконання алгоритму на кластері, що містить вузол із чотирма ядрами, при різній кількості потоків та вершин графа. Результати подано в таблиці 1 і представлено на графіку, де введено позначення:

S – кількість потоків;

N – кількість вершин графа;

t – час виконання алгоритму пошуку k найкоротших шляхів;

a – прискорення, що відповідає збільшенню кількості потоків.

Як видно з експериментальних даних, при N = 20..60 тривалість виконання алгоритму майже однакова для різної кількості потоків, тому що витрачається час на створення паралельних регіонів. При збільшені вершин графа виграш у швидкості, що забезпечується паралельною областю, перевищує затримку на створення групи потоків.

При реалізації алгоритму на чотирьох потоках отримано максимальне прискорення, а на восьми потоках час виконання алгоритму збільшується. Це зумовлено присутністю додаткових витрат на синхронізацію потоків.

Обмін даними в OpenMP відбувається через загальні змінні, що призводить до необхідності розмежовування одночасного доступу різних потоків до загальних даних. Для цього й передбачені різні засоби синхронізації. Але при цьому потрібно враховувати, що використання синхронізації зменшує час виконання програми.

Графічна залежність часу виконання алгоритму t від кількості вершин N і кількості потоків S показана на рис.4.

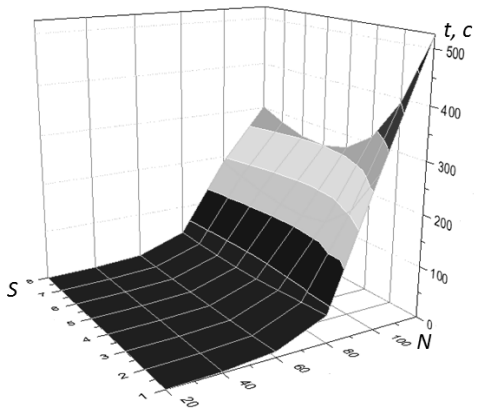


Рисунок 4 – Залежність часу виконання алгоритму t від кількості вершин N та кількості потоків S

Таблиця 1. Експериментальні дані, які отримано з кластера для вузла з чотирма ядрами.

N	S	t, c	a
20	1	0,1569	-
20	2	0,1323	1,1860
20	4	0,1023	1,5337
20	8	0,1149	1,3655
40	1	2,3845	-
40	2	2,1325	1,1181
40	4	1,9705	1,2101
40	8	2,0588	1,1582
60	1	10,9364	-
60	2	8,9633	1,2201
60	4	7,1257	1,5348
60	8	8,0156	1,3542
80	1	52,4387	-
80	2	48,3487	1,0846
80	4	39,1435	1,3396
80	8	41,1271	1,2750
100	1	292,6534	-
100	2	203,1320	1,4407
100	4	150,6231	1,9429

Продовження таблиці 1

100	8	188,5125	1,5524
120	1	522,3417	-
120	2	393,1284	1,3287
120	4	275,1162	1,8986
120	8	302,5632	1,7264

Отже, оптимізація алгоритму Йена за допомогою технології OpenMP, а також правильний вибір планування виконання потоків та організація їх взаємодії з контролем використання ресурсів показали хороший результат, і дозволили збільшити швидкодію програми в цілому.

Висновки

В роботі проаналізовано підхід до розв'язку задачі пошуку оптимальних маршрутів у комп'ютерних мережах, який ґрунтується на алгоритмі Йена. Виконано формалізацію алгоритму з використанням математичного апарату САА-М і отримано послідовну регулярну схему. Виконано трансформацію послідовної регулярної схеми за допомогою САА-М в паралельну регулярну схему.

Математичний апарат модифікованих алгоритмічних алгебр (САА-М) В.М. Глушкова допускає представлення алгоритму в аналітичному вигляді, а саме, САА-М-схем. У свою чергу, САА-М-схеми можуть бути формально модифіковані шляхом застосування до них еквівалентних перетворень, що дає зручну і потужну основу для створення відповідного програмного забезпечення.

Для дослідження паралельної схеми алгоритму застосовано парадигму, що базується на технології OpenMP, завдяки чому підвищується продуктивність алгоритму Йена при певному збільшенні кількості потоків та вершин графа.

Список використаної літератури

1. Кристофидес Н. Теория графов. Алгоритмический подход / Н. Кристофидес. – М.: Мир, 1978. – 432 с.
2. Седжвик Р. Фундаментальные алгоритмы на С++: в ч. Ч. 5: Алгоритмы на графах / Р. Седжвик. – К.: «DiaSoft», 2002. – 484 с.
3. Майника Э. Алгоритмы оптимизации на сетях и графах / Э. Майника. – М.: Мир, 1981. – 324 с.
4. Eppstein D. Finding the k shortest paths / D. Eppstein // 35th IEEE Symp. Foundations of Comp. Sci. – Santa Fe, 1994. – P. 154-165. http://www.ics.uci.edu/~eppstein/pubs/Epp-TR-94-26.pdf
5. Yen J.Y. Finding the K shortest loopless path in a network / J.Y. Yen // Management Science. – 1971. – 17:712-716.
6. Martins E.Q.V. The K shortest paths problem / E.Q.V. Martins, M.M.B. Pascoal and J.L.E. Santos // CISUC. – 1998. http://www.mat.uc.pt/~marta/Publicacoes/k_paths.ps.gz
7. Многоуровневое структурное проектирование программ / Е.Л. Ющенко, Г.Е. Цейтлин, В.П. Грицай и др. – М., 1989. – 254 с.
8. Погорілий С.Д. Оптимізація алгоритмів маршрутизації з використанням систем алгоритмічних алгебр / С.Д. Погорілий, Д.М. Калита // УсиМ. – 2000. – №4. – С. 20-30.

9. Погорілий С.Д. Програмне конструювання: підручник / С.Д. Погорілий; під ред. академіка АПН України О.В. Третяка. – К.: ВПЦ. Київський університет, 2005. – 438 с.

10. Богачёв К.Ю. Основы параллельного программирования / К.Ю. Богачёв. – М.: «Бином», 2003. – 289 с.

11. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: учебное пособие / А.С. Антонов. – М.: Изд-во МГУ, 2009. – 77 с.

12. Kyiv National Taras Schevchenko University high-performance computing cluster / Y.V. Boyko, O.O. Vystoropsky, T.V. Nychyporuk et al. // Third international young scientists` conference on Applied Physics. – 2003. – June 18-20. – P. 180-181.

Надійшла до редколегії 20.03.2012

С.Д. ПОГОРЕЛЫЙ, С.О. КОМИССАРУК
Киевский национальный университет имени Тараса Шевченко

S.D. POGORILY, S.O. KOMISARUK
Taras Shevchenko National University Of Kyiv

ФОРМИРОВАНИЕ И ИССЛЕДОВАНИЕ ПАРАЛЛЕЛЬНЫХ СХЕМ АЛГОРИТМА ЙЕНА

Выполнена формализация алгоритма Йена поиска k кратчайших путей во взвешенном неориентированном графе с использованием математического аппарата модифицированных систем алгоритмических алгебр В.М. Глушкова. Предложена концепция распараллеливания алгоритма для архитектур с общей памятью, основанной на минимизации затрат на параллельную обработку и синхронизацию данных. Проведено преобразование схемы алгоритма и получена параллельная схема. Выполнено экспериментальное сравнение быстродействия последовательной и параллельной схем с использованием кластерных вычислений.

Ключевые слова: маршрутизация, алгоритм Йена, граф, матрица смежности, k кратчайших путей, система алгоритмических алгебр, САА-М схема, распараллеливание, поток, синхронизация, контрольная точка, асинхронная дизъюнкция, OpenMP-технология.

FORMALIZATION OF PARALLEL SCHEMES OFYEN'S ALGORITHM

Formalization of Yen's algorithm for the k shortest path find in weighted undirected graph using mathematical means of V. M. Glushkov modified system of algorithmic algebras is performed. Conceptions of paralleling algorithm for architectures with shared memory, which are based on minimization of loss on parallel data proceeding and synchronization are proposed. Transformation scheme of the algorithm is performed, parallel scheme are obtained. Experimental searching performance gain for parallel and consecutive schemes was carried out by using cluster computation.

Keywords: routing, Yen`s algorithm, graph, adjacency matrix, the k shortest path, systems of algorithmic algebras, SAA-M scheme, paralleling, thread, synchronization, checkpoint, asynchronous disjunctions, OpenMP technology.