УДК 681.3

ФОРМУВАННЯ УЗАГАЛЬНЕНИХ ПАРАЛЕЛЬНИХ СХЕМ АЛГОРИТМУ ФЛОЙДА-УОРШАЛА

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

Виконано формалізацію алгоритму Флойда-Уоршала з використанням математичного апарату модифікованих систем алгоритмічних алгебр. Покроково створено низку схем, розглянуто їх особливості і можливі проблеми експериментальної реалізації. Створено узагальнену паралельну регулярну схему алгоритму, що враховує особливості як систем зі спільною пам'яттю, так із розподіленою.

ВСТУП

Ефективність функціонування сучасних комп'ютерних мереж значною мірою визначається успішним розв'язанням задачі маршрутизації. Для оцінки алгоритму маршрутизації використовують різноманітні критерії якості, такі, як коректність, ефективність, складність, стійкість, адаптивність, справедливість (щодо вузлів, які обслуговуються) тощо. Алгоритм Флойда-Уоршала є одним із методів вирішення задачі маршрутизації, який відповідає названим критеріям.

Із появою багатоядерних процесорів та парадигми паралельних кластерних обчислень з'явилась можливість оптимізувати роботу алгоритму за критерієм часу, використовуючи технології розпаралелювання. Конкретна реалізація паралельного алгоритму суттєво залежить від методів роботи з пам'яттю і може спиратись або на системи з розподіленою пам'яттю, або зі спільною. При застосуванні математичного апарату модифікованих систем алгоритмічних алгебр В.М. Глушкова (САА-М) [1, 2] вдається формалізувати логіку роботи алгоритму таким чином, щоб абстрагуватися від конкретної парадигми і розглядати їх як часткові випадки більш загальної абстрактної схеми.

У роботі з використанням САА-М виконано формування узагальнених паралельних схем алгоритму Флойда-Уоршала [3] для різних парадигм паралельного програмування з метою його застосування, в тому числі й у маршрутизаторах нового покоління.

ОПИС АЛГОРИТМУ

Алгоритм використовує поняття проміжної вершини — вершини, що належить простому шляху $\{v_1, v_2, v_3, ..., v_{N-1}, v_N\}$, проте відрізняється від v_1 та v_N. Нехай граф складається з n вузлів: $V = \{1,2,3,...,n\}$. Розглянемо деяку підмножину вузлів $\{1,2,3,...,k\}$ для певного k. Для довільної пари вершин i, j, що належать V, аналізуємо усі можливі шляхи від i до j, що мають

проміжні вершини з обраної підмножини. Нехай серед цих шляхів p — шлях з найменшою вагою. Використаємо взаємозв'язок між p та найкоротшим шляхом, що має проміжні вершини з підмножини $\{1,2,3,...,k-1\}$. Якщо k не проміжна вершина на шляху p, то усі проміжні вузли p належать до $\{1,2,3,...,k-1\}$. Отже, найкоротший шлях з i в j, що має проміжні вершини з $\{1,2,3,...,k\}$, збігається з найкоротшим шляхом між тими самими вершинами, проміжні вузли якого належать $\{1,2,3,...,k-1\}$. Якщо k — проміжна вершина p, то цей шлях можна розбити у такий спосіб: $i \overset{p_1}{\to} k \overset{p_2}{\to} j$, де p_1 — найкоротший шлях з i до k, всі проміжні вузли якого належать до $\{1,2,3,...,k\}$, але k не може бути проміжною вершиною даного шляху. Тому усі проміжні вузли p_1 належать $\{1,2,3,...,k-1\}$. Аналогічно і для p_2 [3]. Рекурсивне визначення ваги шляхів, що відповідає зробленому вище зауваженню, наступне:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} - \text{вага відповідного ребра при } k = 0; \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}), \ k \neq 0. \end{cases}$$

Елементи $d_{ij}^{(0)}$ складають матрицю D, яка є набором вхідних даних алгоритму. Далі введено рекурсивне визначення матриці передування, яка після виконання алгоритму повертає набір останніх проміжних вершин на найкоротшому шляху від i до j.

$$\pi_{ij}^{(0)} = \begin{cases} NIL \text{ при } i = j \text{ або } w_{ij} = \infty, \\ i \text{ в іншому випадку.} \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ при } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} \text{ в іншому випадку.} \end{cases}$$

Формування інформаційної множини $\overline{M}$ у багатьох випадках залежить від конкретної задачі та умов її виконання. За умови вважатимемо сам факт виконання алгоритму в системі з обмеженими ресурсами (часу, пам'яті тощо), а також логічну організацію (архітектуру) системи (дає вимогу створення САА-схем у її термінах (наприклад, схем синхронізації та обміну даними). З цієї причини інформаційна множина для схеми послідовного алгоритму та паралельної регулярної схеми (ПРС) можуть суттєво відрізнятися.

Інформаційна множина послідовного алгоритму. Робота алгоритму Флойда-Уоршала ґрунтується на обробці масивів даних, які є, в свою чергу, матрицею передування P ($P_{i,j}$ містить номер вершини, що передує вершині j на шляху від i до j) та матрицею ваги найкоротших шляхів D. Тому P і D належать $\overline{M}$. Знаходження необхідних значень цих матриць відбувається ітеративно. На кожному кроці знаходяться проміжні матриці P^k, D^k. До інформаційної множини також належать індекси i, j (призначені для проходження по P і D) та індекс k — параметр (крок) виконання алгоритму.

Оператори та умови. Розглянемо набір операторів і умов із САА-схеми алгоритму, спираючись на загальну думку (без деталізації проблем реалізації у системах з обмеженими ресурсами). Основні оператори алгоритму такі:

$$A_D \equiv D_{i,j}^k = D_{i,j}^{k-1}; \quad A_P \equiv P_{i,j}^k = P_{i,j}^{k-1};$$

$$B_D \equiv D_{i,j}^k = D_{i,k}^{k-1} + D_{k,j}^{k-1}; \quad B_p \equiv P_{i,j}^k = P_{k,j}^{k-1}.$$

З цього місця індекси біля операторів (типу D, P чи q) визначатимуть підмножину інформаційної множини, на якій оператор здійснює зміни.

Введемо допоміжні оператори, дія яких полягає у зміні (інкременті) параметрів циклічних процесів:

$\mathrm{inc}\,X$, де $X = i, j, k$ — оператори інкременту (збільшення на одиницю) відповідного параметра.

InitTemps — оператор для ініціалізаціїї тимчасових даних. Вигляд оператора не будемо конкретизувати. Він може бути і тотожним оператором E. Поява цього оператора пов'язана з можливою наявністю механізму зменшення об'ємів використовування пам'яті. Введемо умови:

α_X (де $X = i, j, k) \equiv X \in [1, n] \cup Z$ — умова виходу із циклу (альфа-умова);

$\beta_{XYZ} \equiv D_{YZ}^X \neq \infty$ $(Y, Z = i, j;\ X = k, k-1)$ — умова перевірки на нескінченність елементів матриці з вагою шляхів (бета-умова);

$$\gamma \equiv D_{i,j}^{k-1} < (D_{i,k}^{k-1} + D_{k,j}^{k-1})$$
$$\gamma_= \equiv D_{i,j}^{k-1} \le (D_{i,k}^{k-1} + D_{k,j}^{k-1})$$

— умови вибору найкоротшого шляху (гамма-умови).

Однією з проблем реалізації подібних алгоритмів є необхідність моделювання ребра графа із нескінченною вагою. Виходом з цього становища є введення певної критичної величини метрики (найбільшого можливого значення для ваги). Таке введення було реалізовано ще в протоколі RIP [4], де нескінченність дорівнювала 16-ти. Проте існують й інші методи позбутися цієї проблеми (використання вказівників). Метод моделювання порожнього ребра ребром з великою вагою має ряд переваг та недоліків. Виявляється, що при застосуванні цього методу зникає необхідність перевірки бета-умови $\beta_{(k-1)ij}$ на кожному кроці внутрішньої альфа-ітерації без жодних ускладнень схем алгоритму. Крім того, можна взагалі позбутись від бета-умов, проте схеми алгоритму в цьому випадку потребуватимуть змін (введення додаткових операторів обробки вхідної та вихідної матриць алгоритму) для збереження можливості роботи алгоритму у великому діапазоні графів. У наступних схемах було використано моделювання порожнього ребра за допомогою певної критичної величини метрики.

Для алгоритму Флойда-Уоршала залишається актуальною проблема знаходження всіх можливих найкоротших шляхів. Схема, що розглядатиметься поспіль, знаходить лише один найкоротший шлях між двома вершинами. Який саме шлях буде обрано, залежить від вибору гамма-умови. Якщо обирати умову $\gamma_=$, то отримаємо найкоротший шлях із найменшою кількіс-

тю вузлів (схема (1)). Для умови γ кількість вузлів, навпаки, буде максимальною. За наявності великої кількості шляхів із найменшою вагою, всіх їх можна отримати, комбінуючи умови γ і $\gamma_=$ для різних кроків ітерації по k.

ФОРМУВАННЯ САА-СХЕМИ ПОСЛІДОВНОГО АЛГОРИТМУ

Створимо САА-схему алгоритму, використовуючи введені оператори і умови (з урахуванням зауважень 1 та 2). Найпростіші логічні міркування підводять до такої схеми [3]:

$$(k=1)*\underset{\alpha_k}{\{}\,\mathrm{InitTemps}*(i=1)*\underset{\alpha_i}{\{}(j=1)*\underset{\alpha_j}{\{}\quad(\quad[\quad(A_D\vee B_D)*$$

$$\underset{\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}\quad\gamma_=}{}\qquad *\ (\ A_P\vee B_P)]\vee[A_D*A_P])*\mathrm{inc}\,J\}*\mathrm{inc}\,I\}*\mathrm{inc}\,K\}. \qquad (1)$$

В ітераціях по i,j,k здійснюється послідовна обробка матриць P і D. Залежно від того, чи справедливі бета-умови нескінченності, обирається відповідна гілка обрахунку. При умові, що A і B не змінюють істинності гамма-умов (що вірно для даного кроку ітерації), справедлива формула:

$$\underset{\gamma_=}{(}\ A_D\vee B_D)*\underset{\gamma_=}{(}\ A_P\vee B_P)=\underset{\gamma_=}{(}[A_D\vee B_D]\vee[A_P\vee B_P]).$$

Тепер позбудемось від повторення складеного оператора A_D*A_P в початковій схемі алгоритму. Для цього скористаємось тим фактом, що оператор B_D*B_P виконується лише у випадку істинності умови $\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}$ і хибності гамма-умови. Тому справедлива така формула:

$$\underset{\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}\quad\gamma_=}{(}\qquad([A_D*A_P]\vee[B_D*B_P])\vee[A_D*A_P]\ \ )=$$

$$=\underset{\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}\wedge\overline{\gamma_=}}{(}\qquad[B_D*B_P]\vee[A_D*A_P]\ \ ).$$

Отже, отримуємо остаточну послідовну схему алгоритму:

$$(k=1)*\underset{\alpha_k}{\{}\,\mathrm{InitTemps}*(i=1)*\underset{\alpha_i}{\{}(j=1)*\underset{\alpha_j}{\{}\quad(\qquad[B_D*B_P]\vee$$

$$\underset{\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}\wedge\overline{\gamma_=}}{}\qquad\vee[A_D*A_P])*\mathrm{inc}\,J\}*\mathrm{inc}\,I\}*\mathrm{inc}\,K\}. \qquad (2)$$

Для спрощення введемо умову: $\delta\equiv\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}\wedge\overline{\gamma_=}$.

Якщо у внутрішній ітерації бета-умова, що не залежить від індексу j, дорівнюватиме нулю, то на кожній ітерації по j виконуватиметься друга гілка альфа-диз'юнкції. Проте схема (2) передбачає перевірку дельта-умови на кожному кроці ітерації, що значно сповільнить обчислювальний процес. Для уникнення цього, слід винести бета-умову за альфа-ітерацію по j:

$$\underset{\beta_{(k-1)ik}\wedge\beta_{(k-1)kj}\wedge\overline{\gamma_=}}{(}\qquad[B_D*B_P]\vee[A_D*A_P]\ \ )=$$

$$= \underset{\beta_{(k-1)ik}}{(} \quad \underset{\beta_{(k-1)kj} \wedge \overline{\gamma_=}}{(} \quad [B_D * B_P] \vee [A_D * A_P] \quad) \vee [A_D * A_P])$$

$$(2) = (j=1) * \underset{\alpha_J}{\{} \underset{\beta_{(k-1)ik}}{(} \quad \underset{\beta_{(k-1)kj} \wedge \overline{\gamma_=}}{(} [B_D * B_P] \vee [A_D * A_P]) \vee [A_D * A_P]) * \mathrm{inc}\, J\} =$$

$$= \underset{\beta_{(k-1)ik}}{(} [(j=1) * \underset{\alpha_J}{\{} \quad \underset{\beta_{(k-1)kj} \wedge \overline{\gamma_=}}{(} \quad [B_D * B_P] \vee [A_D * A_P] \quad) * \mathrm{inc}\, J\}] \vee$$

$$\vee [(j=1) * \underset{\alpha_J}{\{} [A_D * A_P] * \mathrm{inc}\, J\}]) = (k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * (i=1) *$$

$$* \underset{\alpha_i}{\{} \quad \underset{\beta_{(k-1)ik}}{(} \quad [(j=1) * \underset{\alpha_j}{\{} \quad \underset{\beta_{(k-1)kj} \wedge \overline{\gamma_=}}{(} \quad [B_D * B_P] \vee [A_D * A_P] \quad) * \mathrm{inc}\, J\}] \vee$$

$$\vee [(j=1) * \underset{\alpha_j}{\{} [A_D * A_P] * \mathrm{inc}\, J\}]) * \mathrm{inc}\, I\} * \mathrm{inc}\, K\}. \qquad (3)$$

ФОРМУВАННЯ КОНЦЕПЦІЇ РОЗПАРАЛЕЛЮВАННЯ ЗА ДАНИМИ

Ідея розпаралелювання полягає у розбитті інформаційної множини $\overline{M}$ на підмножини, що не перетинаються, з метою їх асинхронної обробки. Подібні міркування реалізовані у алгоритмі Туега [5], основою якого і є алгоритм Флойда-Уоршала. За цим алгоритмом кожний маршрутизатор X обробляє масив даних $D[X,Y]$, де Y — набір усіх інших вузлів мережі. Дані пересилаються між відповідними маршрутизаторами після зміни топології мережі чи виникнення певної проблеми. В загальному випадку кожна паралельна гілка є оператором , що працює лише на певній підмножині $\overline{M}$.

Розбиття інформаційної множини включає поділ матриць P і D на підматриці (з можливим введенням нових індексів до $\overline{M}$ для кожної паралельної гілки). Розглянемо спочатку розбиття по координаті j . Введемо нові умови в множину умов САА:

$$\alpha_{j_x} \equiv j \in [a_{x-1}, a_x), x \in [1,s] \cup N, a_x \in [1,n] \cup N, a_0 = 1, a_s = n+1.$$

Тоді справедлива така формула:

$$\overline{\alpha_j} = \overline{\alpha_{j_1}} \wedge \overline{\alpha_{j_2}} \wedge \overline{\alpha_{j_3}} \wedge \ldots \wedge \overline{\alpha_{j_{s-1}}} \wedge \overline{\alpha_{j_s}} = \bigwedge_{x=1}^{s} \overline{\alpha_{j_x}}.$$

Перейдемо від схеми (2) до синхронної обробки матриць:

$$(k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * (i=1) * \underset{\alpha_s}{\{} (j=1) * \underset{\alpha_j}{\{} X\} * \mathrm{inc}\, I\} * \mathrm{inc}\, K\}, \qquad (4)$$

$$[X - \text{складений оператор}: X = \underset{\delta}{(} B \vee A) * i\mathrm{inc}\, J].$$

Скористаємось тотожністю САА-М [2] $\underset{\alpha_1 \wedge \alpha_2}{\{} A\} = \underset{\alpha_1}{\{} A\} * \overline{\alpha_2} \vee \underset{\alpha_2}{\{} A\} * \overline{\alpha_1}$ з уточненням, що полягає у специфіці самих операторів і умов. Перед вико-

нанням будь-якого циклу в (4) спочатку відбувається ініціалізація індексу. Оператор A (X в схемі (4)) містить його інкремент. Отже, зважаючи на альфа-умови по j, які було введено раніше, лише одна з $\overline{\alpha_{j_x}}$ буде хибною за кожної перевірки значення $\overline{\alpha_j}$. Якщо $\overline{\alpha_j}$ не хибне, то і усі $\overline{\alpha_{j_x}} = 1$. Властивість випливає з неперетинності множин розбиття і означає виконання лише одної з синхронних гілок в певний момент часу. Розглядаючи виконання синхронної диз'юнкції в цілому, слід зазначити: при переході $\overline{\alpha_{j_{x-1}}} = 0 \to 1$ під час ітеративного процесу відбувається перехід $\overline{\alpha_{j_x}} = 1 \to 0$. Це призводить до миттєвого припинення виконання $(x-1)$ гілки і народження x-ї; $(x-1)$-а гілка дорівнюватиме невизначеному оператору [1], x-а — ні, і виконання диз'юнкції буде дією операторів з x-ої гілки:

$$= (k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) * \underset{\alpha_i}{\{} (j=1) * [(\underset{\alpha_{j1}}{\{} X\} * \underline{\overline{\alpha_{j_2}}} * \underline{\overline{\alpha_{j_3}}} * \ldots * \underline{\overline{\alpha_{j_s}}}) \vee$$

$$\vee (\underset{\alpha_{j2}}{\{} X\} * \underline{\overline{\alpha_{j_1}}} * \underline{\overline{\alpha_{j_3}}} * \ldots * \underline{\overline{\alpha_{j_s}}}) \vee \ldots$$

$$\ldots \vee (\underset{\alpha_{js}}{\{} X\} * \underline{\overline{\alpha_{j_1}}} * \underline{\overline{\alpha_{j_2}}} * \ldots * \underline{\overline{\alpha_{j_{s-1}}}})] * \text{inc}\, I\} * \text{inc}\, K\} =$$

Як уже зазначалося, при переході від одної гілки до іншої своє значення змінює лише наступна альфа-умова $\overline{\alpha_{j_x}}$ (попередня є $(x-1)$ гілкою). Тоді можна позбутися від надлишкових альфа-фільтрів в САА-схемі, що наведена вище. Також зауважимо, що дія оператора X в кожній гілці відбувається на підмножині інформаційної множини, яка не перетинається з будь-якою підмножиною інших гілок. Позначимо ці підмножини через $q_1, q_2, q_3 \cdots q_{s-1}, q_s$.

$$= (k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) * \underset{\alpha_i}{\{} (j=1) * [(\underset{\alpha_{j1}}{\{} X_{q_1}\} * \underline{\overline{\alpha_{j_2}}}) \vee$$

$$\vee (\underset{\alpha_{j2}}{\{} X_{q_2}\} * \underline{\overline{\alpha_{j_3}}}) \vee \ldots \vee (\underset{\alpha_{js}}{\{} X_{q_s}\})] * \text{inc}\, I\} * \text{inc}\, K\} =$$

$$= (k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) *$$

$$* \underset{\alpha_i}{\{} (j=1) * [\bigvee_{x=1}^{s-1} (\underset{\alpha_{jx}}{\{} X_{q_x}\} * \underline{\overline{\alpha_{j_{x+1}}}}) \vee \underset{\alpha_{js}}{\{} X_{q_s}\}] * \text{inc}\, I\} * \text{inc}\, K\}. \qquad (5)$$

Наступним кроком є зведення у часі виконання паралельних гілок САА-схеми (5). Розглянемо спочатку її часову діаграму (рис. 1).

Схема (5) потребує подальших удосконалень, оскільки експериментальна реалізація вимагає часу на створення кожної гілки, що призводить до сповільнення дії алгоритму.

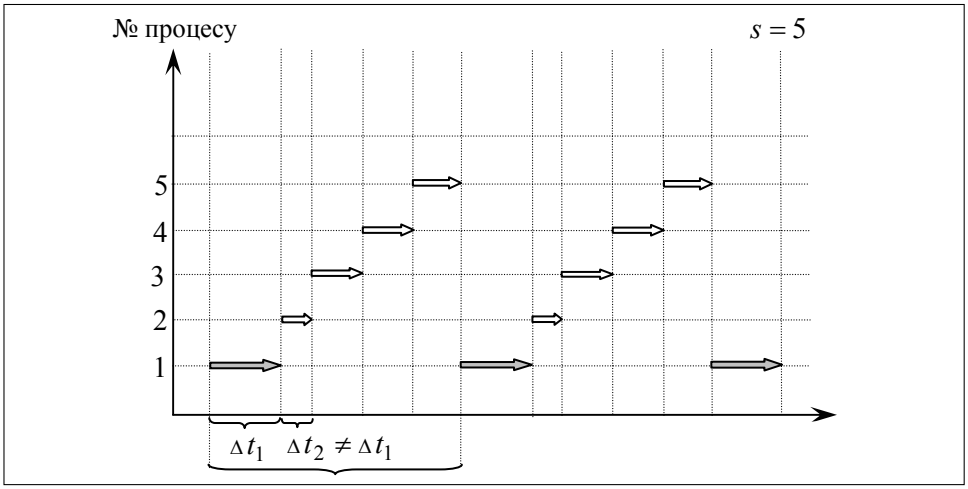

Рис. 1. Часова діаграма схеми (5)

СТВОРЕННЯ АСИНХРОННОЇ ПРС АЛГОРИТМУ

Асинхронне виконання процесів передбачає створення операторів та об'єктів синхронізації в інформаційній множині. Причому, гілки мають діяти незалежно і обмінюватися даними через оператори обміну. Це постає з рівноправності усіх гілок щодо обробки матриць інформаційної множини. Тобто, утворена схема має бути симетричною до гілок виконання. Як було вже зазначено, доцільніше, з метою економії часового ресурсу програми, використати дублювання деяких елементів $\overline{M}$ (наприклад, створити набори індексів для кожної гілки), аніж постійно передавати їх по мережі.

Перейдемо до асинхронної обробки P і D із збереженням часової схеми (рис. 1). Для цього введемо синхронізацію у вигляді контрольних точок і циклів очікування. Спочатку перетворимо схему (5) шляхом внесення оператора $j = 1$ до першої гілки диз'юнкції:

$$(k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) * \underset{\alpha_i}{\{}[\bigvee_{x=2}^{s-1} (\underset{\alpha_{jx}}{\{} X_{q_x}\} * \overline{\underline{\alpha_{j_{x+1}}}}) \vee$$

$$\vee ((j=1) * \underset{\alpha_{j1}}{\{} X_{q_1}\} * \overline{\underline{\alpha_{j_2}}}) \vee \underset{\alpha_{js}}{\{} X_{q_s}\}] * \text{inc}\, I\} * \text{inc}\, K\}.$$

До цього часу ресурс j був спільним для усіх асинхронних гілок. Але означення асинхронної диз'юнкції вимагає, щоб q-множини, на яких виконуються оператори її гілок, були неперетинні. Тому введемо сукупність індексів, замість j в інформаційній множині. Змінюються наступні умови і оператори:

$$j \to j_x;\ \alpha_{j_x} \equiv j_x \in [a_{x-1}, a_x),\ \ x \in [1,s] \cup N,\ \ a_x \in [1,n] \cup N,\ \ a_0 = 1,\ \ a_s = n+1;$$

$$\text{inc}\, J \to \text{inc}\, J_x;\ \ q_x \to q_x \cup \{j_x\}.$$

Крім того, виникає необхідність у залученні нових операторів. Щоб зберегти поведінку виконання схеми (5), необхідно ввести: $j_x = j_{x-1}$ для $x \neq 1$. Тоді отримуємо таку схему:

$$(k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) * \underset{\alpha_i}{\{} [\bigvee_{x=2}^{s-1} ((j_x = j_{x-1}) * \underset{\alpha_{jx}}{\{} X_{q_x}\} * \overline{\underline{\alpha_{j_{x+1}}}}) \vee$$

$$\vee ((j_1 = 1) * \underset{\alpha_{j1}}{\{} X_{q_1}\} * \overline{\underline{\alpha_{j_2}}}) \vee ((j_s = j_{s-1}) * \underset{\alpha_{js}}{\{} X_{q_s}\})] * \text{inc}\, I\} * \text{inc}\, K\}. \quad (6)$$

Оператори синхронізації мають бути не прив'язані до конкретної реалізації. Їх дія полягає в очікуванні одних гілок обрахункового процесу деякої події в інших. Такі оператори потребують введення додаткових умов, що стають істинними у випадку виконання необхідної події (крім того можлива поява в інформаційній множині об'єктів синхронізації, стан яких і генерує подію). Оператори синхронізації в загальному випадку є складеними операторами і можуть бути представлені як композиції операторів встановлення події (умови), очікування, скидання умови. Аналогом останніх можуть виступати оператори контрольної точки T, циклу очікування S та оператор R (Reset), що протилежний по дії до T. R, S, T можуть бути рознесені по САА-схемі із збереженням загальної синхронізації гілок. Як умови очікування введемо нові умови контрольних точок: τ_x-умови, що встановлюються в одиницю при дії оператора $T(\tau_x)$ і скидаються у нуль при дії $R(\tau_x)$.

При використанні асинхронної диз'юнкції на момент переходу до нової гілки виконується присвоювання індексу цієї гілки значення індексу попередньої на момент виходу із неї. Оскільки інформаційні підмножини цих гілок неперетинні, то для виконання оператора необхідно передати значення індексу шляхом обміну між гілками. Проте можна скористатися тим фактом, що значення відповідного індексу при переході на нову гілку вже відоме (це ліва межа відповідного відрізку, на які було розбито розмірність j): $j_x = a_{x-1}$. Тому, враховуючи специфіку роботи синхронної і асинхронної диз'юнкції, справедлива формула:

$$[\bigvee_{x=2}^{s-1} ((j_x = j_{x-1}) * \underset{\alpha_{jx}}{\{} X_{q_x}\} * \overline{\underline{\alpha_{j_{x+1}}}}) \vee ((j_1 = 1) * \underset{\alpha_{j1}}{\{} X_{q_1}\} * \overline{\underline{\alpha_{j_2}}}) \vee$$

$$\vee ((j_s = j_{s-1}) * \underset{\alpha_{js}}{\{} X_{q_s}\})] =$$

$$= [\dot{\bigvee_{x=2}^{s}} (S(\tau_{x-1}) * R(\tau_{x-1}) * (j_x = a_{x-1}) * \underset{\alpha_{jx}}{\{} X_{q_x}\} * T(\tau_x) * S(\tau_s)) \dot{\vee}$$

$$\dot{\vee} ((j_1 = a_0) * \underset{\alpha_{j1}}{\{} X_{q_1}\} * T(\tau_1) * S(\tau_s) * R(\tau_s))], \quad (7)$$

де S — оператор циклу очікування істинності відповідної умови. Ця схема справедлива лише для систем виконання зі спільною пам'яттю, оскільки не передбачає обміну даними між гілками під час свого виконання.

У синхронній диз'юнкції схем (6) і (7) виконання гілок відбувається послідовно. В асинхронній диз'юнкції схеми (7) наступна гілка очікує виконання умови, що встановлюється попередньою. Крім того, кожна гілка очікує виконання останньої (s -ї) після відповідної альфа-ітерації:

$$(k=1)*\underset{\alpha_k}{\{}\,\mathrm{InitTemps}*(i=1)*\underset{\alpha_i}{\{}[\dot{\bigvee_{x=2}^{s}}(S(\tau_{x-1})*R(\tau_{x-1})*(j_x=a_{x-1})*\underset{\alpha_{jx}}{\{}\,X_{q_x}\}*$$

$$*T(\tau_x)*S(\tau_s))\,\dot{\vee}\,((j_1=a_0)*\underset{\alpha_{j1}}{\{}\,X_{q_1}\}*T(\tau_1)*S(\tau_s)*R(\tau_s))]*\mathrm{inc}\,I\}*\mathrm{inc}\,K\}. \quad (8)$$

Тепер зведемо у часі процеси асинхронної диз'юнкції, перетворивши модель синхронізації. Не всі паралельні гілки рівноправні (це випливає з того, що до народження гілок існував їхній «батьківський» процес). Має існувати мінімум одна гілка, оператори якої б не відповідали порядку операторів чи взагалі операторам інших (порушення симетрії гілок). Вона називається основним чи *серверним процесом*. Оскільки в нашому випадку виділено тільки перший і останній процеси, то оберемо як серверний процес саме перший. Ідея полягає у тому, що він має керувати синхронною роботою інших гілок. При роботі на системі з розподіленими ресурсами є можливість створення симетричної схеми без виділення процесу. Положення операторів T в схемі (8) визначає послідовний характер виконання схеми. Переміщення кожного з операторів T на початок виконання кожного процесу (розміщення T після R) призведе до паралельного виконання гілок. Гілка 1 запускає другий процес на паралельне виконання, гілка 2 запускає третю і т.п. Але оскільки всі процеси з другого по останній є рівноправними, то слід привести схему (8) до симетричної щодо цих процесів. Для того, щоб зберегти правильність виконання наступних схем, введемо тимчасовий оператор TMP, що зупиняє виконання відповідної гілки, поки інші процеси не почнуть його виконувати:

$$(k=1)*\underset{\alpha_k}{\{}\,\mathrm{InitTemps}*(i=1)*\underset{\alpha_i}{\{}[\dot{\bigvee_{x=2}^{s}}(S(\tau_{x-1})*R(\tau_{x-1})*(j_x=a_{x-1})*$$

$$*\underset{\alpha_{jx}}{\{}\,X_{q_x}\}*T(\tau_x)*S(\tau_s)*TMP)\,\dot{\vee}$$

$$\dot{\vee}\,((j_1=a_0)*\underset{\alpha_{j1}}{\{}\,X_{q_1}\}*T(\tau_1)*S(\tau_s)*R(\tau_s)*TMP)]*\mathrm{inc}\,I\}*\mathrm{inc}\,K\}.$$

Перенесемо оператор $T(\tau_x)$ на початок композиції операторів гілки. Він змінює лише стан умов τ_x і жодним чином не впливає на інші елементи множини $\overline{M}$. Крім того, оператори композиції, які не пов'язані із синхронізацією, не впливають на істинність τ_x, а TMP оперує з іншим набором синхронізуючих умов. Тому справедлива наступна схема:

$$(k=1)*\underset{\alpha_k}{\{}\,\mathrm{InitTemps}*(i=1)*\underset{\alpha_i}{\{}[\dot{\bigvee_{x=2}^{s}}(S(\tau_{x-1})*R(\tau_{x-1})*T(\tau_x)*$$

$$*(j_x=a_{x-1})*\underset{\alpha_{jx}}{\{}\,X_{q_x}\}*S(\tau_s)*TMP)\,\dot{\vee}$$

$$\dot{\vee}\,(T(\tau_1)*S(\tau_s)*R(\tau_s)*(j_1=a_0)*\underset{\alpha_{j1}}{\{}X_{q_1}\}*TMP)]*\mathrm{inc}\,I\}*\mathrm{inc}\,K\}. \quad (9)$$

Отриманий механізм синхронізації ще не забезпечує необхідної зупинки гілок виконання, яку надає оператор TMP. Із схеми бачимо, що певна гілка почне виконання лише тоді, коли попередня досягне відповідного оператора T (це не стосується першого процесу). Як і раніше, серверний процес запускає на виконання усі інші і чекає відгуку останнього. Процеси вводять по ланцюгу: перший запускає другий, другий — третій тощо. Після запуску відповідної гілки, остання скидає свою умову очікування. Це необхідно для багаторазового використання такої моделі синхронізації. Проте, для бажаного очікування гілок іншим даної схеми не достатньо. Так, якщо виконання першої та другої гілок набагато випереджає третю, це призводить до проходження другого процесу у новий цикл ітерації з можливим використанням неправильних даних (позбавитись цього недоліку можна, подвоївши утворений складений оператор $S*R*T(T*S*R)$ у тілі схеми). Введемо оператор:

$\mathrm{Sync}_x(\vec{\alpha})$ — оператор синхронізації, $\vec{\alpha}$ — вектор умов синхронізації.

Для схеми (9):

$$\mathrm{Sync}_x(\vec{\alpha})\equiv T(\tau_1)*S(\tau_s)*R(\tau_s)*T(\tau_1)*S(\tau_s)*R(\tau_s),\ \vec{\alpha}=(\tau_1,\tau_s)\ \text{при}\ x=1;$$

$$\mathrm{Sync}_x(\vec{\alpha})\equiv S(\tau_{x-1})*R(\tau_{x-1})*T(\tau_x)*S(\tau_{x-1})*R(\tau_{x-1})*T(\tau_x),\ \vec{\alpha}=(\tau_x,\tau_{x-1})$$

— в іншому випадку.

Такий оператор абстрагується від конкретної схеми синхронізації. Тепер із схеми можна прибрати оператор TMP, після чого отримаємо:

$$(k=1)*\underset{\alpha_k}{\{}\,\mathrm{InitTemps}*(i=1)*\underset{\alpha_i}{\{}[\overset{s}{\underset{x=1}{\dot{\vee}}}(\mathrm{Sync}_x(\vec{\alpha})*$$

$$*(j_x=a_{x-1})*\underset{\alpha_{jx}}{\{}X_{q_x}\})]*\mathrm{inc}\,I\}*\mathrm{inc}\,K\}. \quad (10)$$

Схема є симетричною щодо будь-якої гілки. У загальному випадку оператор Sync призводить до очікування відповідної гілки дії операторів Sync інших гілок. На рис. 2 зображена часова діаграма виконання процесів за схемою (10), що не враховує додаткові витрати часу, які зумовлені обмеженістю апаратних та програмних ресурсів системи (наприклад, мала кількість процесорів, втрата часу на ініціалізацію об'єктів синхронізації в інформаційній множині тощо).

Системи із спільною пам'яттю надають можливість використання тієї ж самої ділянки пам'яті у всіх потоках (гілках обчислень). У даному випадку оператори InitDE_x надають змогу використовувати створений канал в x-гілці, InitDE наразі створює канал обміну. Відповідно оператори DelDE, DelDE_x призначені для знищення каналу обміну. Оператор DE_x у цій моделі є тотожним оператором, оскільки використання відповідного елемента іншої гілки передбачає пряме звернення до нього. Щоб задовольнити означення асинхронної диз'юнкції, враховуємо те, що елемент іншої гілки вико-

ристовується тільки для читання. Тоді можна вважати, що процеси використовують не безпосередньо елементи інших гілок, а їхні копії, отримання яких можливе за рахунок моделі спільної памяті.

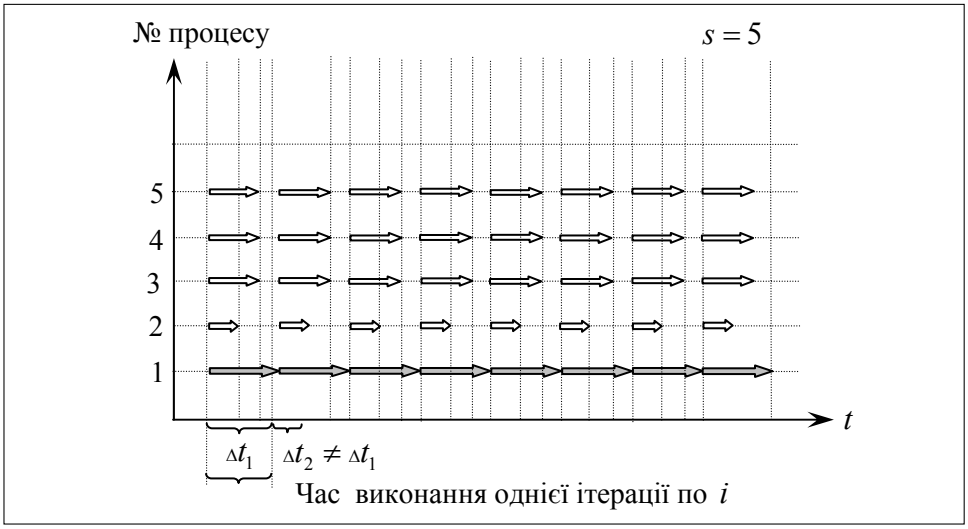

Рис. 2. Часова діаграма схеми (10)

Системи із розподіленою пам'яттю не надають можливості використовувати прямий доступ до елементів. Вони вимагають передачі даних від одних процесів до інших. Оператори ініціювання каналу і його знищення зберігають попередній зміст. Дія DE_x залежить від x і може бути як передачею даних у відповідні гілки, так і їх отриманням. Оператори InitDE і DelDE можуть бути тотожними операторами (наприклад, при використанні MPI [6]).

Узгоджено схему (10) із схемою використання обміну даними. Для цього розмістимо оператор InitDE перед усіма альфа-ітераціями, DelDE — після усіх операторів, оскільки будь-яке інше розміщення буде призводити до додаткових витрат часу при кожній ініціалізації і знищенні каналу. InitDE_x і DelDE_x мають бути розташовані в паралельних гілках до та після використання процесу обміну даними. Не слід розташовувати їх в альфа-ітерації по j. Обмін даними DE_x виконується до їх використання і може розміщуватися як в циклічному процесі, так і поза ним (від цього залежить об'єм даних, що передаються. При розміщені DE_x в циклі, відбувається обмін малою кількістю даних (2 елементи на кожній ітерації)). Щоб уникнути багаторазового обміну, слід винести DE_x за межі альфа-ітерації. Введені оператори впливають лише на стан нововведених об'єктів — каналу та буферів. Отримуємо:

$$\mathrm{InitDE} * (k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * (i=1) * \underset{\alpha_i}{\{}[\dot{\bigvee_{x=1}^{s}}(\mathrm{InitDE}_x * \mathrm{Sync}_x(\vec{\alpha}) * \mathrm{DE}_x *$$

$$* (j_x = a_{x-1}) * \underset{\alpha_{jx}}{\{} X_{q_x}\} * \mathrm{DelDE}_x)] * \mathrm{inc}\, I\} * \mathrm{inc}\, K\} * \mathrm{DelDE}\,.$$

Виконаємо оптимізацію за індексом i аналогічно:

$$Y_M \equiv [\dot{\bigvee_{x=1}^{s}}(\mathrm{InitDE}_x * \mathrm{Sync}_x(\vec{\alpha}) * \mathrm{DE}_x * (j_x = a_{x-1}) * \underset{\alpha_{jx}}{\{} X_{q_x}\} * \mathrm{DelDE}_x)] * \mathrm{inc}\, I .$$

Як і для j, розіб'ємо відрізок $[1,n]$ значень індексу на p інтервалів, що не перетинаються:

$$\alpha_{i_x} \equiv i \in [b_{x-1}, b_x),\ x \in [1,p] \cup N,\ b_x \in [1,n] \cup N,\ b_0 = 1,\ b_p = n+1 .$$

Перейдемо до схеми (11) із синхронною диз'юнкцією, використавши співвідношення САА-М: $\underset{\alpha_1 \wedge \alpha_2}{\{\_} A\} = \underset{\alpha_1}{\{} A\} * \overline{\underline{\alpha_2}} \vee \underset{\alpha_2}{\{} A\} * \overline{\underline{\alpha_1}}$.

$$\mathrm{InitDE} * (k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * (i=1) * [\dot{\bigvee_{x=1}^{p-1}}(\underset{\alpha_{ix}}{\{} Y_{l_x}\} * \overline{\underline{\alpha_{i_{x+1}}}}) \vee$$

$$\vee \underset{\alpha_{ip}}{\{} Y_{l_p}\}] * \mathrm{inc}\, K\} * \mathrm{DelDE} . \qquad (11)$$

У схемі (11) $l_1, l_2, l_3, ..., l_{p-1}, l_p$ — позначення підмножин виконання оператора Y.

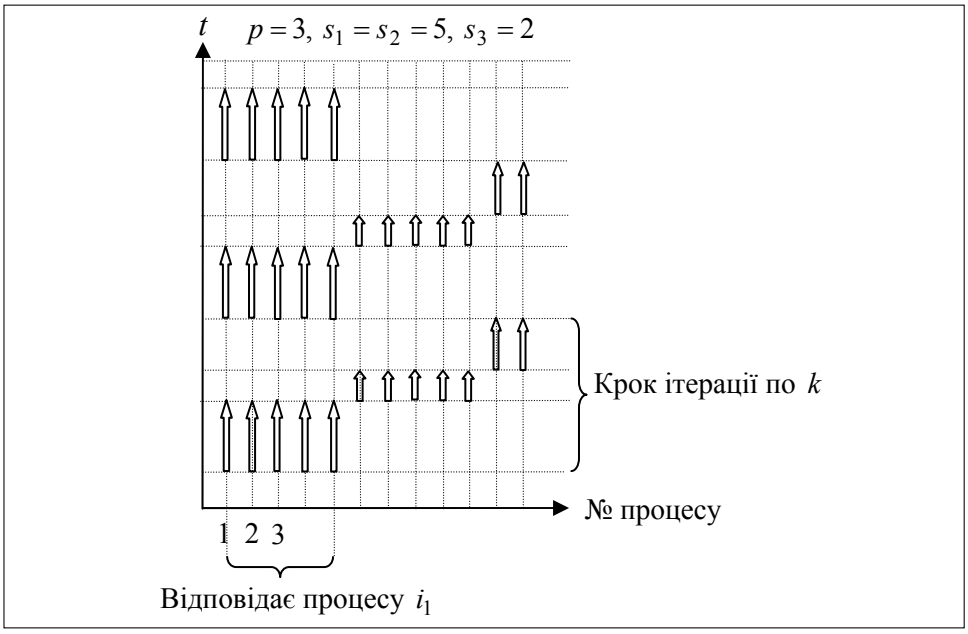

Рис. 3. Часова діаграма виконання схеми (11)

Рис. 3 ілюструє виконання схеми (11) для відповідних значень s, p. Паралельні процеси (стрілки на рис. 3) є гілками, що були утворені при оптимізації за індексом j. Їхня структура у збільшеному вигляді показана на рис. 2. Ресурс i — спільний для усіх гілок.

При формуванні формули (8) вигляд X не був конкретизований. Тому цю послідовність міркувань можна використати щодо оператора Y. Слід

ввести набір індексів $i_1, i_2, i_3, \ldots, i_{p-1}, i_p$ замість одного індексу і аналогічно змінити відповідні оператори, умови, множини.

Крім того, тимчасово вводиться новий набір умов синхронізації, циклів очікування, контрольних точок і операторів скидання умов. Отже, отримуємо аналог формули (10):

$$\text{InitDE} * (k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) *$$

$$* [\overset{p}{\underset{x=1}{\dot{\vee}}} (\text{Sync}_x^i(\vec{\alpha}) * (i_x = b_{x-1}) * \underset{\alpha_{ix}}{\{} Y_{l_x}\})] * \text{inc}\, K\} * \text{DelDE}. \qquad (12)$$

У схемі (12) введено набір нових операторів синхронізації, які за своєю дією аналогічні попереднім, проте можуть бути реалізовані по-іншому (про що свідчить верхній індекс у цих операторах). На відміну від схеми (10), формула не потребує введення операторів обміну, оскільки останні присутні в Y.

Схема (12), після розкриття складної дії Y, перетвориться на схему (13):

$$\text{InitDE} * (k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * (i=1) * [\overset{p}{\underset{y=1}{\dot{\vee}}} (\text{Sync}_y^i(\vec{\alpha}) * (i_y = b_{y-1}) *$$

$$* \underset{\alpha_{iy}}{\{} [\overset{s}{\underset{x=1}{\dot{\vee}}} (\text{InitDE}_{x,y} * \text{Sync}_{x,y}(\vec{\alpha}) * \text{DE}_{x,y} * (j_{x,y} = a_{x-1,y}) *$$

$$* \underset{\alpha_{jxy}}{\{} X_{q_{xy}}\} * \text{DelDE}_{x,y})] * \text{inc}\, I_y\})] * \text{inc}\, K\} * \text{DelDE}. \qquad (13)$$

Оператори у внутрішніх гілках асинхронної диз'юнкції набувають додаткового індексу, що з'явився за рахунок розбиття по i. Цей факт свідчить, що для кожного зовнішнього процесу (y-процесу) може існувати власні синхронізація, оператори обміну даними і своє розбиття за індексом j: набір індексів j, введених до інформаційної множини ($j_{x,y}$). Отже, кількість можливих варіантів розбиття за індексом j у схемі дорівнює кількості областей розбиття розмірності i. Оператор X для кожного x, y має однакову реалізацію, проте виконуються на різних підмножинах інформаційної множини. Справедлива наступна САА-тотожність [1, с. 143]:

$$\underset{\alpha}{\{} A * T(\Delta_1) * S(\Delta_2) \dot{\vee} B * T(\Delta_2) * S(\Delta_1)\} =$$

$$= \underset{\alpha}{\{} A * T(\Delta_1) * S(\Delta_2)\} \dot{\vee} \underset{\alpha}{\{} B * T(\Delta_2) * S(\Delta_1)\}, \qquad (14)$$

де Δ_1, Δ_2 — локально замкнуті в ітераціях умови.

Введена вище формула (14) враховує скидання (окрім останньої ітерації) умов Δ_1, Δ_2 після дії відповідних операторів очікування. Поведінка отриманих в роботі схем дещо відрізняється від класичних за рахунок наявності R — оператора скидання умов. Це одночасно надає можливість прак-

тичного використання схем в багатьох системах, оскільки дії S і R у певних випадках можна об'єднати. Тому використаємо формулу (14) з урахуванням більш загальної моделі синхронізації. При винесенні диз'юнкції за межі ітерації слід враховувати збільшення кількості ітераторів в інформаційній множині. Кожна гілка x містить підгілку y, що має власний індекс $i_{x,y}$. Як і для випадку утворення ітераторів j, має місце модифікування відповідних умов альфа-ітерацій і операторів інкремента:

$$\underset{\alpha}{\{} A * \mathrm{Sync}_1(\vec{\alpha}) \dot{\vee} B * \mathrm{Sync}_2(\vec{\alpha})\} = \underset{\alpha}{\{} A * \mathrm{Sync}_1(\vec{\alpha})\} \dot{\vee} \underset{\alpha}{\{} B * \mathrm{Sync}_2(\vec{\alpha})\}.$$

$$\mathrm{InitDE} * (k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * (i=1) * [\overset{p}{\underset{y=1}{\dot{\vee}}} [\overset{s}{\underset{x=1}{\dot{\vee}}} (\mathrm{Sync}_y^i(\vec{\alpha}) * (i_y = b_{y-1}) *$$

$$* \underset{\alpha_{iy}}{\{} \mathrm{InitDE}_{x,y} * \mathrm{Sync}_{x,y}(\vec{\alpha}) * \mathrm{DE}_{x,y} * (j_{x,y} = a_{x-1,y}) *$$

$$* \underset{\alpha_{jxy}}{\{} X_{q_{xy}}\} *_{x,y} * \mathrm{inc}\, I_{y,x}\})]] * \mathrm{inc}\, K\} * \mathrm{DelDE}. \qquad (15)$$

Остання схема (15) включає також у гілки x-диз'юнкції оператори синхронізації та ініціалізації, що отримують при цьому інший індекс. Для синхронізації це означає лише те , що її модель залежить від конкретної робочої гілки, яких тепер $s_1 + s_2 + ... + s_p$ (всі процеси рівноправні).

У формулі (15) гілка зупиняється двічі: вперше — після моменту створення на новому кроці за індексом k; вдруге — при кожному вході в цикл за індексом i. Остання зупинка є надлишковою, оскільки алгоритм вимагає, щоб процеси входили синхронно лише в кожну нову ітерацію за k. Тому, позбавившись від операторів $\mathrm{Sync}_{x,y}(\vec{\alpha})$, досягнемо покращення часових параметрів алгоритму, оскільки час на створення надлишкових елементів синхронізації буде дорівнювати нулю:

$$\mathrm{InitDE} * (k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * [\overset{p}{\underset{y=1}{\dot{\vee}}} [\overset{s_y}{\underset{x=1}{\dot{\vee}}} (\mathrm{Sync}_{y,x}^i(\vec{\alpha}) * (i_{y,x} = b_{y-1,x}) *$$

$$* \underset{\alpha_{iyx}}{\{} \mathrm{InitDE}_{x,y} * DE_{x,y} * (j_{x,y} = a_{x-1,y}) * \underset{\alpha_{jxy}}{\{} X_{q_{xy}}\} *$$

$$* \mathrm{DelDE}_{x,y} * \mathrm{inc}\, I_{y,x}\})]] * \mathrm{inc}\, K\} * \mathrm{DelDE}.$$

Витрати часу виникають за рахунок постійної ініціалізації каналу зв'язку за входження гілки в цикл за локальним індексом *i*. Це питання вирішується винесенням операторів ініціалізації і знищення каналу за межі циклу. Власне оператор обміну DE може бути теж винесений за альфа-ітерацію, але він тоді набуває іншого змісту. DE обмінюється вже не окремими векторами, а цілими матрицями даних:

$$\mathrm{InitDE} * (k=1) * \underset{\alpha_k}{\{} \mathrm{InitTemps} * [\overset{p}{\underset{y=1}{\dot{\vee}}} [\overset{s_y}{\underset{x=1}{\dot{\vee}}} (\mathrm{Sync}_{y,x}^i(\vec{\alpha}) * (i_{y,x} = b_{y-1,x}) *$$

$$* \text{InitDE}_{x,y} * DE_{x,y} * \underset{\alpha_{iyx}}{\{} (j_{x,y} = a_{x-1,y}) * \underset{\alpha_{jxy}}{\{} X_{q_{xy}}\} * \text{inc}\, I_{y,x}\} *$$

$$* \text{DelDE}_{x,y})]] * \text{inc}\, K\} * \text{DelDE}. \qquad (16)$$

Наступним кроком пришвидшення алгоритму є внесення оператора ініціалізації тимчасових даних і альфа-ітерації по k до асинхронних диз'юнкцій. Це призводиить до розширення інформаційної множини відповідним набором індексів $k_{x,y}$ і проміжних змінних алгоритму. Кожна гілка тепер матиме повний власний набір ресурсів і дані передаються лише за рахунок операторів обміну. Саме таку схему можна реалізувати в системах із розподіленою пам'яттю (попередні схеми мали неявний спільний ресурс — параметр циклу k і тимчасові змінні):

$$\text{InitDE} * [\overset{p}{\underset{y=1}{\dot{\vee}}} [\overset{s_y}{\underset{x=1}{\dot{\vee}}} ((k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * \text{Sync}^i_{y,x}(\vec{\alpha}) * (i_{y,x} = b_{y-1,x}) *$$

$$* \text{InitDE}_{x,y} * \text{DE}_{x,y} * \underset{\alpha_{iyx}}{\{} (j_{x,y} = a_{x-1,y}) * \underset{\alpha_{jxy}}{\{} X_{q_{xy}}\} * \text{inc}\, I_{y,x}\} *$$

$$* \text{DelDE}_{x,y} * \text{inc}\, K\})]] * \text{DelDE}.$$

Останній крок до створення остаточної ПРС алгоритму — використання тотожності: $(A_1 \dot{\vee} A_2) \dot{\vee} (A_3 \dot{\vee} A_4) = A_1 \dot{\vee} A_2 \dot{\vee} A_3 \dot{\vee} A_4$.

Тоді маємо:

$$\text{InitDE} * [\overset{p}{\underset{y=1}{\dot{\vee}}} \overset{s_y}{\underset{x=1}{\dot{\vee}}} ((k=1) * \underset{\alpha_k}{\{} \text{InitTemps} * \text{Sync}^i_{y,x}(\vec{\alpha}) * (i_{y,x} = b_{y-1,x}) *$$

$$* \text{InitDE}_{x,y} * \text{DE}_{x,y} * \underset{\alpha_{iyx}}{\{} (j_{x,y} = a_{x-1,y}) * \underset{\alpha_{jxy}}{\{} X_{q_{xy}}\} * \text{inc}\, I_{y,x}\} * *$$

$$\text{DelDE}_{x,y} * \text{inc}\, K\})] * \text{DelDE} \qquad (17)$$

Для систем із розподіленою пам'яттю InitDE, DelDE можуть бути «одиничними» операторами (наприклад, MPI). Для систем із спільною пам'яттю ці оператори створюють необхідну область пам'яті спільного користування.

Часова діаграма роботи алгоритму зображена на рис. 4. Аналіз схеми (17) свідчить, що немає сенсу утворювати процеси із суттєво різним навантаженням. За наявності такої конфігурації час виконання алгоритму визначатиметься часом обробки цими гілками великих масивів даних. Проте, інколи цей випадок використовують для фонового виконання операцій. Тобто, гілки з низьким навантаженням можуть виконувати не основний алгоритм, а певні інші дії (підготовка даних для обміну, тощо).

Процес оптимізації зводиться нанівець при розбитті основної матриці D таким чином, що існуватиме одна велика сукупність даних для одної гілки і малі набори для інших.

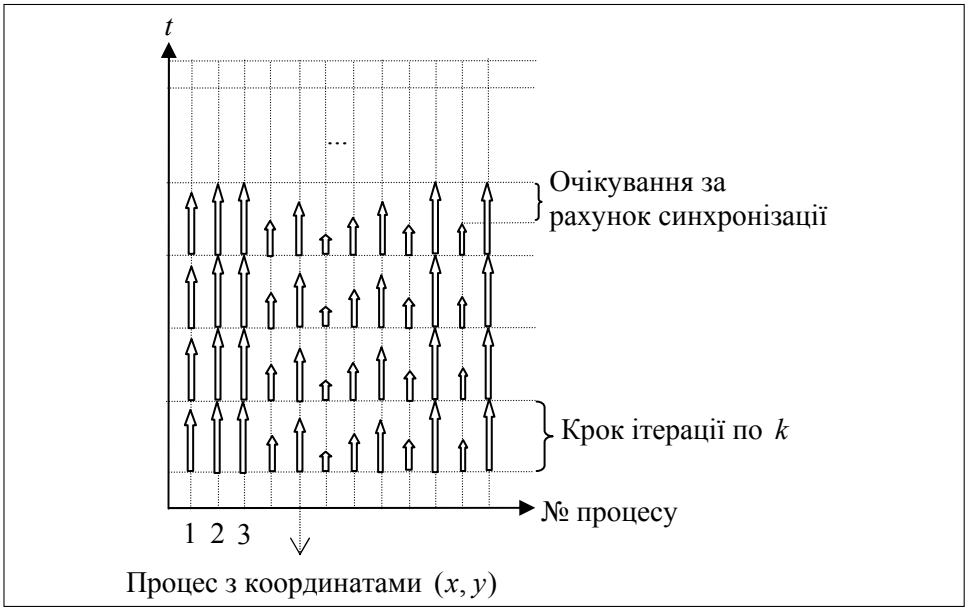

Рис. 4. Часова діаграма роботи ПРС алгоритму (17)

Рівномірного розподілу навантаження можна досягти через схеми розбиття, зображені на рис. 5 (*nproces* — це кількість процесів).

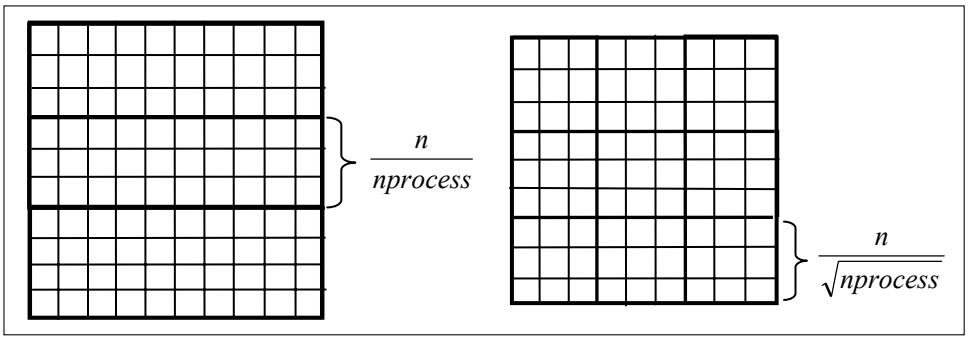

Рис. 5. Схема розбиття вхідного масиву

Ці розбиття можуть містити процеси з меншим навантаженням, порівняно з іншими гілками. Але кількість даних у них буде близька до кількості даних у інших. Тому ці схеми найкраще збалансовані [7].

ВИСНОВКИ

У роботі з використанням математичного апарату САА-М виконано формалізацію алгоритму Флойда-Уоршала. Покроково створено низку схем цього алгоритму (від послідовної до паралельних); розглянуто їх особливості і можливі проблеми експериментальної реалізації. Використання САА-М В.М. Глушкова надає можливість формалізувати паралельну схему алгоритму, не прив'язуючись до архітектури конкретної системи, на якій вона виконуватиметься. Саме тому створена ПРС є загальним випадком як для

архітектури систем зі спільною пам'яттю, так і з розподіленою. Експериментальна реалізація схеми (17) вимагає конкретизації взаємодії між гілками.

Перетворення САА-схем — потужний механізм оптимізації паралельних алгоритмів, який вимагає врахування специфіки поставленої задачі. Схема (17) теоретично дає пришвидшення алгоритму в $\sum_{y=1}^{p} s_y$ разів, але при експериментальній реалізації останнє обмежується фізичними характеристиками системи (кількістю вузлів обчислення, швидкістю обміну даними між ними та витратами часу на створення об'єктів синхронізації).

ЛІТЕРАТУРА

1. *Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К.* Многоуровневое структурное проектирование программ. — М.: Финансы и статистика, 1989. — 192 с.
2. *Погорілий С.Д., Камардіна О.О.* Дослідження та створення інструментальних засобів автоматизованої трансформації схем алгоритмів // Проблемы программирования. — 2004. — № 1–2. —С. 417–421.
3. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы, построение и анализ. — М.: МЦНМО, 2000. — С. 719–725.
4. *Столлингс В.* Современные компьютерные сети. — СПб.: Питер, 2003. — С. 479–486.
5. *Захаров В.А.* Курс лекцій: розподілені алгоритми. — http://mathcyb.cs.msu.su/courses/distr_alg.html.
6. *Богачев К.Ю.* Основы параллельного программирования. — М.: БИНОМ. Лаборатория знаний, 2003. — С. 232–292.
7. *Погорілий С.Д., Камардіна О.О., Бавикін О.І.* Про підхід до розпаралелювання алгоритму Флойда-Уоршала // Математичні машини і системи. — 2005. — Вып. 3. — С. 91–101.

Поступила 01.06.2009