ПАРАЛЕЛЬНИЙ MIMD-ОРІЄНТОВАНИЙ ТОПОЛОГІЧНИЙ АНАЛІЗАТОР МОДЕЛІ МЕРЕЖНОГО ДИНАМІЧНОГО ОБ'ЄКТА

Смагін О.М.

кафедра ЕОМ ДонНТУ,
smagin@cs.dgtu.donetsk.ua

Abstract

Smagin O. Parallel MIMD-oriented topological analyzer of mine ventilation network model. *The paper presents the research of topological analyser. The structure of topological analyzer is considered. Parallel algorithms of construction of tree and antitree, matrices A, S are discussed.*

Вступ

Мережні динамічні об'єкти (МДО) відносяться до класу складних динамічних систем [1]. Математичні моделі МДО будуються на основі формального опису, що включає дві складові: опис топології, який характеризує структуру та кількісну складність об'єкта; математичний опис процесів, що в кількісному та якісному факторах тісно пов'язаний з топологією об'єкта.

Задача керування мережними динамічними об'єктами різної фізичної природи відноситься до складної наукової проблеми, для вирішення якої останнім часом застосовуються паралельні обчислювальні системи MIMD-структури [2]. В рамках міжнародного наукового співробітництва ДонНТУ та Штутгартського університету ведеться розробка MIMD-моделей МДО і їх імплементація в розподіленому паралельному моделюючому середовищі (РПМС). Важливою складовою РПМС є топологічний аналізатор (ТА), який при побудові моделей МДО реальної розмірності виконує комп'ютерну підтримку роботи з топологіями, що зводиться до наступних функцій (рис. 1): кодування топології; побудова дерева й антидерева графа МДО; перекодування топології відповідно знайденому варіантові дерева й антидерева; побудова топологічних матриць інциденцій A і незалежних контурів S.

Алгоритми топологічного аналізу МДО для послідовних ЕОМ розглянуті в [1,3]. У статті пропонуються паралельні алгоритми побудови топологічного аналізатора, що забезпечують вхідні дані для функціонування MIMD-моделей мережних об'єктів. Практичне застосування MIMD-моделей МДО можливе при автоматизації шахтних вентиляційних мереж (ШВМ), газотранспортних систем та інших об'єктів.

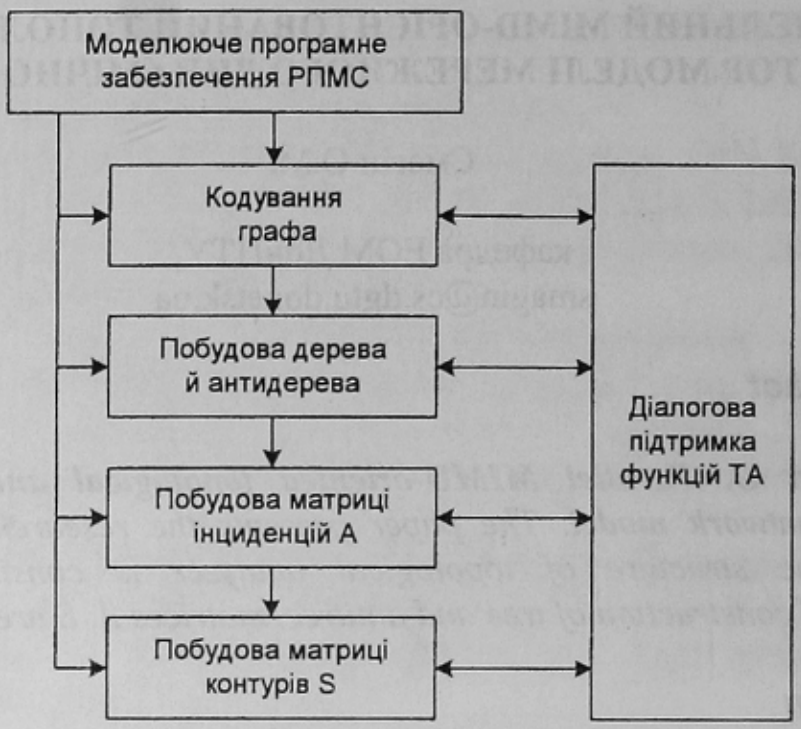

Рисунок 1 – Структура та функції топологічного аналізатора

1. Алгоритм побудови дерева й антидерева

МДО може бути представлений у вигляді графа $G(U,Q)$, що має $n=|U|$ вузлів і $m=|Q|$ гілок. Граф $G(U,Q)$ кодується таблицею *TABUR*, що має формат, зображений на рис. 2. Поля таблиці означають наступне:

- *AKJ* – номер початкового вузла *QI*-гілки, при цьому $J \in (1,2,...,n)$;
- *EKK*–номер кінцевого вузла *QI*-гілки, при цьому $K \in (1,2,...,n)$;
- *QI*–ідентифікатор гілки графа;
- *RI,KI,HI* – фізичні параметри гілки графа;
- *KOM* – опис важливих для предметної області ознак гілки у формі коментарів.

	EKK			KI		

Рисунок 2 – Формат початкової таблиці *TABUR*

MIMD-орієнтований алгоритм побудови дерева й атидерева базується на виконанні дій з таблицями. Завдання полягає в одержанні з початкової таблиці кодування графа *TABUR* таблиці дерева *BAUMTAB* і антидерева *ANTIBAUMTAB*, що складають таблицю *TABURXY*, перекодованої відповідно знайденому варіантові дерева й антидерева.

Розглянемо етапи алгоритму (рис. 3):

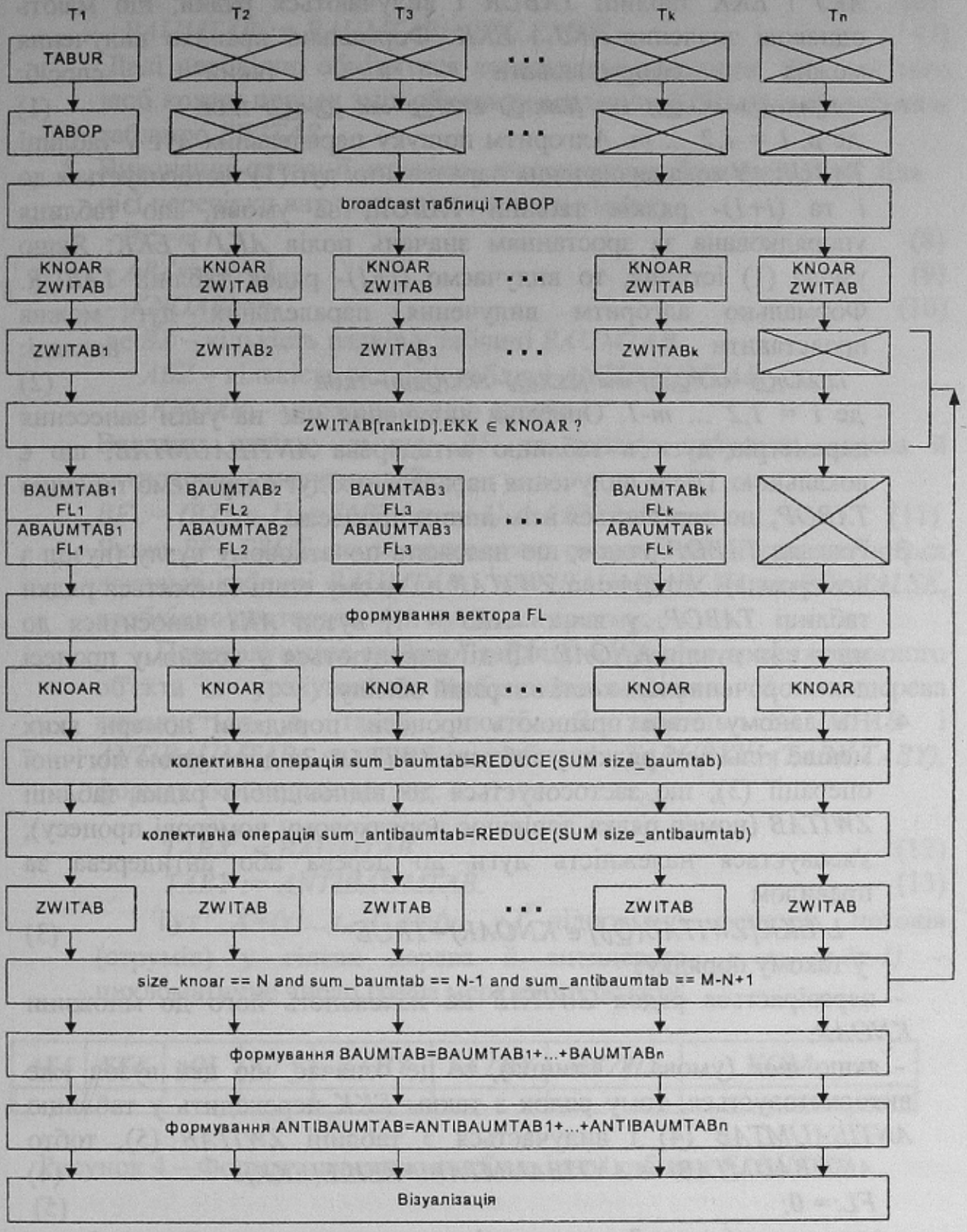

Рисунок 3 – Схема паралельного алгоритму визначення дерева

1. Введення й ініціалізація початкової таблиці кодування графа *TABUR*, операція виконується в процесі T_1.

2. Вилучення паралельних дуг. У процесі T_1 аналізуються стовпці *AKJ* і *EKK* таблиці *TABUR* і вилучаються рядки, що мають однакові значення *AKJ* і *EKK*. Формально правило вилучення можна сформулювати в такий спосіб:

$$L([AKJ(Q_i)=AKJ(Q_p)] \text{ and } [EKK(Q_i)=EKK(Q_p) \text{ and } [Q_i \neq Q_p])=TRUE, \quad (1)$$

де *p, i = 1,2 ... m*. Алгоритм пошуку паралельних дуг у таблиці *TABUR*: Умова визначення паралельних дуг (1) застосовується до *i* та *(i+1)*- рядків таблиці *TABUR*, за умови, що таблиця упорядкована за зростанням значень полів *AKJ* і *EKK*. Якщо умова (1) істинна, то вилучаємо *(i+1)*- рядок таблиці *TABUR*. Формально алгоритм вилучення паралельних дуг можна представити у вигляді:

$$L([AKJ(Q_i)=AKJ(Q_{i+1})] \text{ and } [EKK(Q_i)=EKK(Q_{i+1})])=TRUE, \quad (2)$$

де *i = 1,2 ... m-1*. Операція вилучення має на увазі занесення параметрів дуги в таблицю антидерева *ANTIBAUMTAB*, що є локальною. Після вилучення паралельних дуг одержуємо таблицю *TABOP*, що передається всім іншим процесам.

3. Розгляд *TABOP*-рядків, що належать початковому вузлу (вузол з номером 1). У таблицю *ZWITAB* на цьому етапі заносяться рядки таблиці *TABOP*, у яких *AKJ* = 1, вузол *AK1* заноситься до множини вузлів *KNOAR*. Ці дії виконуються у кожному процесі для скорочення кількості операцій обміну.
4. На даному етапі працюють процеси, порядкові номери яких менше кількості рядків у таблиці *ZWITAB*. За допомогою логічної операції (3), що застосовується до відповідного рядка таблиці *ZWITAB* (номер рядка дорівнює порядковому номерові процесу), з'ясовується належність дуги до дерева або антидерева за правилом

$$L(EKK[ZWITAB(Q_i)] \in KNOAR)=TRUE \quad (3)$$

у такому порядку:

- перевіряється рядок *ZWITAB* на належність його до множини *KNOAR;*
- якщо *так* (умова 3 істинна), те це означає, що цей вузол уже використовується, тому рядок з таким *EKK* переходить у таблицю *ANTIBAUMTAB* (4) і вилучається з таблиці *ZWITAB* (5), тобто

$$ANTIBAUMTABi := ANTIBAUMTABi+ZEILE(EKK) \quad (4)$$

$$FL := 0; \quad (5)$$

- якщо *немає* (умова 3 помилкова), то це означає, що цей вузол і відповідний рядок таблиці *ZWITAB* повинні бути додані до *KNOAR* (6) і *BAUMTAB* (7) відповідно. Виникає неоднозначність додавання того самого вузла в різні локальні таблиці, тому потрібно взяти за правило, що операції (6) і (7) виконуються в процесі з меншим порядковим номером, інакше виконуються

операції (4) і (5). Формально це можна виразити так:

$FL := EKK$ (6)

$BAUMTABi := BAUMTABi + ZEILE(EKK)$ (7)

Далі необхідно обмінятися локальними змінними *FL*, для того щоб кожен процес зміг обновити множину *KNOAR* і сформувати таблицю *ZWITAB*.

5. Виконання операції перевірки завершення побудови дерева. Для цієї перевірки використовуються три умови:

- $BZ=n-1$ (8)
- $ABZ=m-n+1$ (9)
- $|KNOAR|=n$ (10)

де *BZ* – кількість рядків у таблиці *BAUMTAB*

ABZ – кількість рядків у таблиці *ANTIBAUMTAB*

|KNOAR| - множина вузлів дерева

Введемо логічну змінну *BF* як ознаку побудови дерева й антидерева і визначимо її:

$BF := (BZ=n-1) \cap (ABZ=m-n+1) \cap (|KNOAR|=n)$ (11)

Якщо *BF=TRUE*, то за допомогою операції обміну поєднуються локальні таблиці *BAUMTAB* і *ANTIBAUMTAB*. Якщо *BF=FALSE*, необхідно повторити 4-й пункт алгоритму.

Перекодування вихідної таблиці кодування графа мережного об'єкта з урахуванням побудованого дерева й антидерева виконується в такий спосіб. З таблиць *BAUMTAB* і *ANTIBAUMTAB* складається таблиця *TABURXY=(TABX,TABY)*, формат якої подано на рис. 4:

$TABX := BAUMTAB,$ (12)

$TABY := ANTIBAUMTAB.$ (13)

Тут $X=(x_1 \ldots x_{n-1})^T, Y=(y_1 \ldots y_\gamma)^T$–відповідно вектори потоків (струмів) у гілках дерева й антидерева, $\gamma = m-(n-1)$ – цикломатичне число графа мережного об'єкта.

AKJ	*EKK*	*QI*	X / Y	*X / RY*	*KX / KY*	*HX / HY*	KOM

Рисунок 4 – Формат перекодованої вихідної таблиці *TABURXY*

2. Алгоритм побудови матриці інциденцій

Паралельний алгоритм генерування матриці інциденцій $A=(A_X \; A_Y)$ (рис.5) полягає в перетворенні таблиці *TABURXY* і представлений на рис. 6.

Генерування рядків матриці *A* виконується за наступними правилами:

$$Ax_{jk} = \begin{cases} -1, \text{ if TABURXY[k].AKJ} = j; \\ +1, \text{if TABURXY[k].EKK} = j; \\ 0, \text{ if KNOAR(TABURXY[k])} \neq j \end{cases} \quad (14)$$

де j = 1, 2,..,N, k = 1, 2, ... , N-1;

$$Ay_{jk} = \begin{cases} -1, \text{ if TABURXY[k].AKJ} = j; \\ +1, \text{if TABURXY[k].EKK} = j; \\ 0, \text{ if KNOAR(TABURXY[k])} \neq j \end{cases} \quad (15)$$

де j = 1, 2,..,N , k = N-1,...,M; Генерування кожного рядка може виконуватися незалежно від інших рядків, тому необхідно *n-1* процесів, у кожнім з яких буде заповнюватися рядок матриці A_X і A_Y за правилами (14) і (15).

Вузли	Гілки											
	X_1	X_2	...	...	...	X_{n-1}	Y_1	Y_2	...	...	...	Y_{m-n+1}
$AK(EK)_1$												
...												
...												
$AK(EK)_{n-1}$												

Рисунок 5 – Формат таблиці *A*

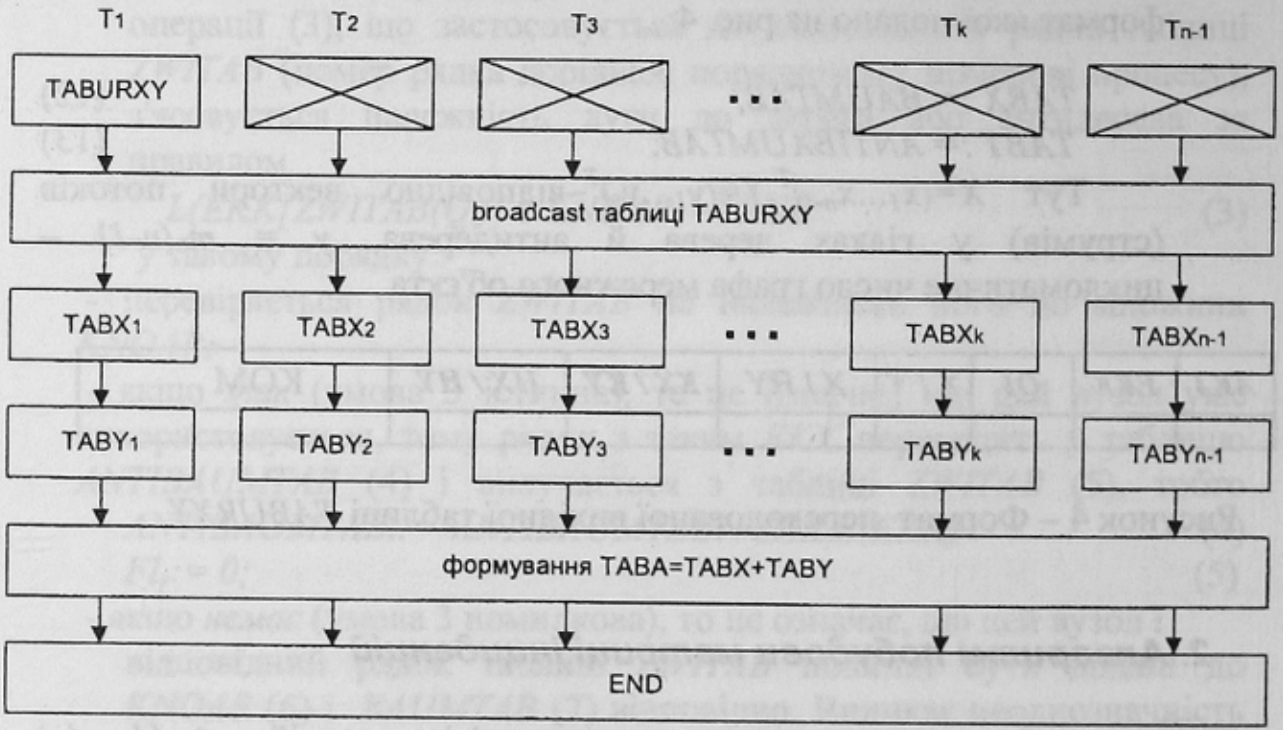

Рисунок 6 – Схема паралельного алгоритму визначення матриці *A*

3. Алгоритм побудови матриці незалежних контурів

Алгоритм генерування матриці незалежних контурів S полягає у виділенні всіх можливих замкнутих контурів з визначеною орієнтацією. Результат генерування:

$$S = (S_X \; S_Y), \tag{16}$$

де S_X і S_Y – частини матриць, структуровані щодо дерева й антидерева.

Відповідно таблиці *TABURXY* матриця S представляється в такий спосіб (рис. 7):

Контури	Гілки											
	X_1	X_2	...	...	...	X_{n-1}	Y_1	Y_2	...	...	...	Y_γ
1												
...												
...												
γ												

Рисунок 7 – Формат таблиці S

Структура алгоритму представлена на рис. 8.

Алгоритм генерування матриці S містить наступні кроки:

1. З *TABY* формуються контурні назви з рядками $Y_1 \div Y_{m-n+1}$ (m – кількість гілок, n – кількість вузлів), що є базами незалежних контурів.
2. Прохід по дереву в прямому напрямку, починаючи з кінцевого *EKP*-вузла бази контуру. У випадку відсутності гілки в таблиці дерева, для продовження контуру пошук гілки здійснюється в таблиці антидерева.

Пошук гілок на даному кроці продовжується доти, поки кінцевий *EKP*-вузол гілки не буде дорівнювати базовому вузлу дерева.

Якщо деякий *AKK*-вузел містить більш однієї вихідної гілки, то ідентифікуються всі гілки, що інцидентні вузлу *AKK*. Необхідно побудувати всі можливі контури і потім здійснити вибір одного з них за двома критеріями:

- мінімум гілок антидерева;
- максимум довжини контуру.

Вибір за критерієм максимуму довжини контуру здійснюється лише тоді, коли виникають два контури з однаковою кількістю гілок антидерева.

Знайдені гілки для контуру заносяться в таблицю *M(J)*.

3. Прохід по дереву в зворотному напрямку від початкового *AKJ*-вузла контурної назви до збігу цього вузла з базовим вузлом дерева. При цьому пошук гілки дерева для контуру виконується шляхом порівняння кінцевого *EKP*-вузла з початковим *AKJ*-вузлом у таблиці дерева. Якщо вони рівні, то гілка знайдена і вона заноситься в контур.

4. Виконується перевірка на завершення побудови контурів ($J > m\text{-}n+1$). Якщо всі контурні назви використані, то виконується перехід на 5-й крок, інакше перехід на крок 2.

5. По контурах $M(J)$, $J=1, 2, ..., m\text{-}n+1$ будується матриця контурів S за наступними правилами:

а) Розглядаються проміжні вектори $MS(J)$, що виходять з $M(J)$ як стовпці X, Y.

b) З елементів масиву $MS(j, k)$ формуємо елементи матриць S_X, S_Y:

$$S_{Xj,k} = \begin{cases} 1, \text{ if } MS(j,k) = Xk, 1 \le k \le n-1 \\ 0, \text{ if } MS(j,k) = 0, 1 \le k \le n-1 \end{cases} \quad (17)$$

$$S_{Yj,p} = \begin{cases} 1, \text{ if } MS(j,k) = Yp, k-(n-1) \le p \le m-(n-1), k > n-1 \\ 0, \text{ if } MS(j,k) = 0, k-(n-1) \le p \le m-(n-1), k > n-1 \end{cases} \quad (18)$$

c) Із підматриць S_X, S_Y утвориться матриця S:

$$S = (S_X S_Y) \quad (19)$$

$$S_{j,k} = \begin{cases} Sx_{j,k} \text{ для } 1 \le j \le m-n+1, 1 \le k \le n-1 \\ SY_{j,p} \text{ для } 1 \le j \le m-n+1, n-1 \le k \le m-n+1, p = k-(n-1) \end{cases} \quad (20)$$

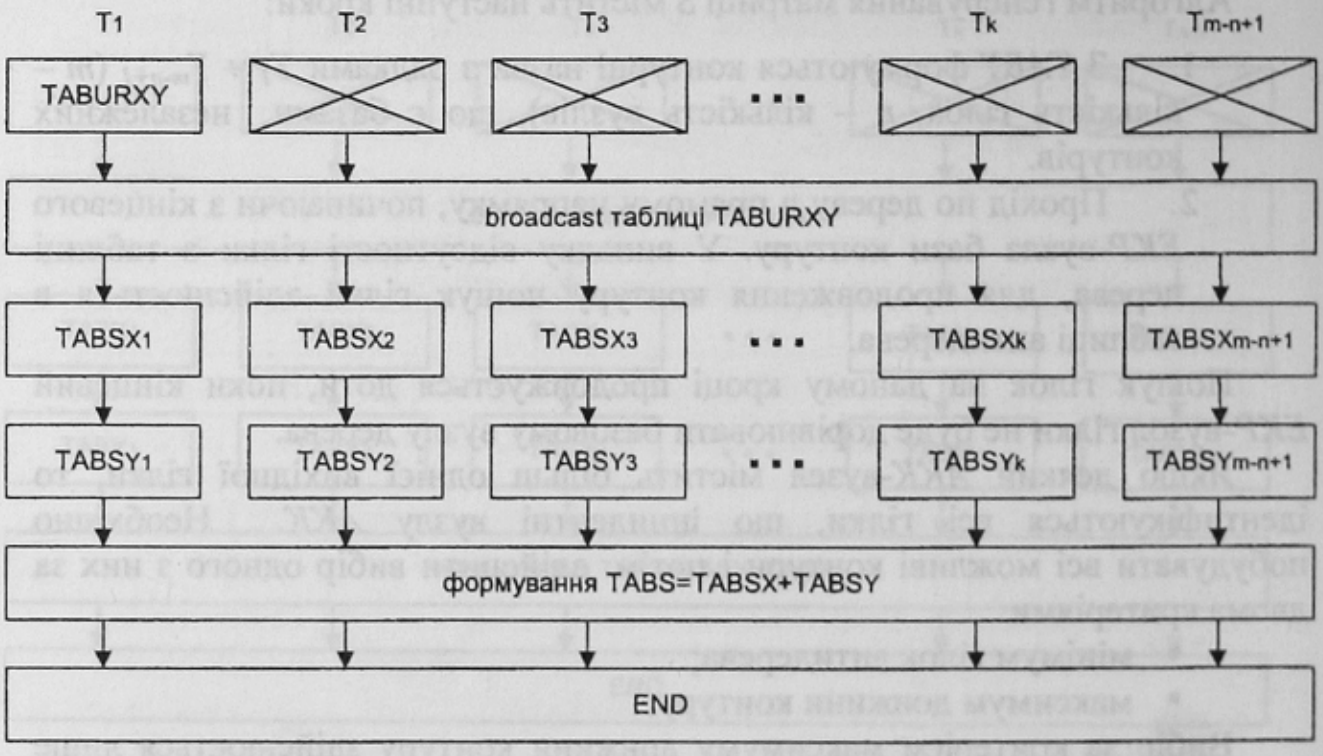

Рисунок 8 - Схема паралельного алгоритму визначення матриці S

4. Програмна реалізація топологічного аналізатора

Паралельні алгоритми визначення дерева й антидерева, матриці інциденцій *A*, матриці контурів *S* реалізовано MPI-програмою. В основній частині програми декларуються необхідні змінні і ініціалізується *MPI* з використанням функції, як параметри виступають змінні оточення програми. За допомогою *MPI*-функцій визначаються загальна кількість процесів, виділених програмі (*colProcesses*) і ідентифікаційний номер кожного процесу (від *0* до *colProcesses-1*). У процесі з номером *0* виконується підготовчий етап, що включає операції відкриття файлу з таблицею *TABUR*, зчитування значень *M* і *N* (кількість гілок і вузлів). Виділення пам'яті і заповнення таблиці *TABUR* виконується динамічно в залежності від отриманих значень *M* і *N*, після чого звільняється дескриптор файлу *TABUR*. Для динамічного виділення пам'яті під таблиці *TABOP, BAUMTAB, ANTIBAUMTAB, ZWITAB, KNOAR* в усі процеси необхідно передати значення *M* і *N*. Передачу кількості гілок і вузлів виконує функція *MPI_Bcast(...)*. Для зменшення кількості обмінів даними таблиця *TABUR* не передається іншим процесам, а в *0-м* процесі відбувається видалення рядків, що кодують паралельні гілки графа, у результаті одержуємо таблицю без закодованих паралельних дуг *TABOP*. Після проведення підготовчих етапів кожен процес має все необхідне для безпосереднього визначення дерева та антидерева. Відповідно до алгоритму спочатку організовується нескінченний цикл, вихід з якого відбувається після задоволення трьом умовам (3.8-3.10). На кожнім кроці в загальному випадку активно працюють не всі процеси, кількість працюючих процесів визначається розмірами таблиці *ZWITAB* на *i-м* кроці алгоритму. Отже, кожен процес обробляє один рядок з таблиці *ZWITAB*. Якщо вузол з номером *EKK* уже занесений у масив *KNOAR*, то даний вузол уже ввійшов у граф дерева і тому гілку необхідно занести в локальну, для даного процеса, таблицю *ANTIBAUMTAB* і установити ознаку занесення гілки в антидерево (*flag = 0*), інакше потрібно перевірити на допустимість занесення цієї гілки в таблицю *BAUMTAB* у даному процесі. Для цього перевіряється *EKK* гілок тих процесів, у яких номера менше ніж у поточного. Якщо збіг не знайдений, то даний процес може заносити дану гілку у таблицю *BAUMTAB* і встановлювати ознаку занесення гілки в дерево *flag=EKK*, інакше занесення цього рядка виконується в іншому процесі і встановлюється *flag=0*, щоб уникнути повторного занесення одного рядка в локальні таблиці *BAUMTAB*. В зв'язку з тим, що різні процеси закінчать цей етап у різні проміжки часу, тому їх необхідно синхронізувати, для цього застосовуємо функцію синхронізації. Алгоритм має на увазі наявність масиву *KNOAR* у кожному процесі, але після першого кроку необхідно модифікувати зміст масиву *KNOAR*, для цього від кожного процесу одержуємо ознаку *flag*, що містить *0*, якщо вузол уже

є в *KNOAR*, інакше містить номер вузла, який необхідно додати в локальний масив *KNOAR*. Для одержання від кожного процесу ознаки *flag* і передачі всім процесам використовуємо функцію колективного обміну. Після одержання даних кожен процесор додає ненульові елементи масиву *local_knoar* до локального масиву *KNOAR*. Для визначення, чи побудовано дерево і антидерево, необхідно мати наступні дані: кількість елементів у масиві *KNOAR*, кількість рядків у таблицях *BAUMTAB* і *ANTIBAUMTAB*. Кожен процес має тільки розміри своїх локальних таблиць, тому за допомогою колективних функцій обміну одночасно організовується обмін даними між процесами і виконання арифметичної операції підсумовування, тому що потрібна тільки сума розмірів локальних таблиць. Формування таблиці *ZWITAB* відбувається в кожнім процесі, виконується заповнення таблиці *ZWITAB* тими рядками таблиці *TABOP*, у яких *AKJ* дорівнюють елементу масиву *local_knoar[j]*. Після визначення локальних таблиць *BAUMTAB* і *ANTIBUMTAB* їх необхідно передати в *0-й* процес для організації запису у файл для подальшої обробки. Так як розміри таблиць у загальному випадку різні, то на початку потрібно організувати прийом розмірів таблиць у *0-му* процесорі і за допомогою отриманих даних підготувати прийом самих таблиць і запис результатів у файли. Після цього можна переходити до визначення матриці інциденцій *A*. У процесі з номером *0* виконується підготовчий етап, що включає операції відкриття файлу з таблицею *TABURXY*, зчитування значень *M* і *N*. Виділення пам'яті і заповнення таблиці *TABURXY* виконується динамічно в залежності від отриманих значень *M* і *N*, після чого звільняється дескриптор файлу *TABURXY*. Треба відзначити, що кожен процес заповнює тільки один рядок таблиць *TABX, TABY*, тому пам'ять виділяється тільки для одного рядка. У *0-му* процесі пам'ять виділяється цілком, тому що дані від усіх процесів повинні бути передані в головний процес для подальшого компонування і запису у файл. Після ініціалізації таблиці *TABURXY* від *0-го* процесу вона передається всім процесам за допомогою функції колективного обміну. Відповідно до алгоритму в кожнім процесі, кількість яких повинна бути *N-1*, організується послідовність циклів по гілках дерева і антидерева таблиці *TABURXY* для заповнення рядків таблиць *TABX, TABY*. Кожен процес заповнює рядок, номер якого дорівнює порядковому номеру процесу, отже, для кожної гілки дерева потрібно перевірити *AKJ* і *EKK* на рівність з *rankID;* якщо *rankID* дорівнює *AKJ*, то гілка виходить з вузла і у таблицю *TABX* необхідно занести *–1*, інакше якщо *rankID* дорівнює *EKK*, то гілка входить у вузол і в таблицю *TABX* заноситься *+1*, інакше гілка у цьому вузлі не присутня і в таблицю *TABX* заноситься *0*. Заповнення таблиці *TABY* здійснюється в такий же спосіб, відмінність у тім, що розглядаються гілки антидерева таблиці *TABURXY* і заповнюється таблиця *TABY*. Після заповнення рядків таблиць *TABX, TABY* у процесах виконується збір даних у *0-м* процесі. У процесі з номером *0*,

після прийому таблиць, повинне бути виконане формування загальної таблиці *A* и запис її у файл. Паралельно у всіх процесах відбувається звільнення пам'яті, зайнятої під таблиці *TABURXY, TABX, TABY*. Визначення матриці контурів *S* виконується наступним чином. Для виділення пам'яті під таблицю *TABS* в усі процеси необхідно передати значення *M* і *N*. Треба відзначити, що кожен процес заповнює тільки один рядок таблиці *TABS*, тому пам'ять виділяється тільки для одного рядка. У *0-му* процесі пам'ять виділяється цілком, тому що дані від усіх процесів повинні бути передані в головний процес для подальшого компонування і запису у файл. Після ініціалізації таблиці *TABURXY* у *0-му* процесі, вона передається всім процесам. Після проведення підготовчих етапів кожен процес має всі необхідні дані для визначення матриці *S*. Відповідно до алгоритму в кожнім процесі визначається один рядок таблиці *S*, кількість процесів відповідає кількості контурів, тобто *M-N+1*. Заповнення таблиці здійснюється в два етапи. На першому етапі відбувається визначення послідовності вузлів до базового вузла в прямому порядку, починаючи з *EKK* даної гілки антидерева. Для цього застосовується функція, яка рекурсивно визначає всі шляхи до базового вузла. Вибір оптимального шляху, за обраними критеріями, здійснюється динамічно, тобто як тільки знайдено поточний шлях, виконується перевірка з вже отриманим оптимальної шляхом, і в залежності від порівняння береться за оптимальний один з них. На другому етапі відбувається визначення послідовності вузлів до базового вузла в зворотному порядку, починаючи з *AKJ* даної гілки антидерева. Заповнення таблиці *TABS* відбувається за допомогою функції, яка знаходить у таблиці *TABURXY* номер гілки за початковим *AKJ* і кінцевому *EKK* вузлах. По отриманому номеру гілки ми заповнюємо таблицю *TABS*. Після заповнення рядків таблиць *TABS* у процесах виконується збір даних у *0-м* процесі, на прийомній стороні таблиця *TABS* має максимальний розмір і тому виділяти буферну пам'ять не треба. У процесі з номером *0* після прийому таблиці повинен бути виконаний запис її у файл. Паралельно у всіх процесах відбувається звільнення пам'яті, зайнятої під таблиці. Після звільнення ресурсів системи можна виконувати деінсталяцію *MPI*.

MPI-програма топологічного аналізатора відлагоджена на MIMD-системі NEC Штутгартського університету та на кластері ПЕОМ кафедри ЕОМ ДонНТУ.

Висновки

Як показали програмні реалізації й експериментальні дослідження, паралельний ТА забезпечує побудову топологічних характеристик (дерево, антидерево, матриця інциденцій і контурів, вектори і матриці параметрів) шахтних вентиляційних мереж та інших МДО реальної складності.

Запропонований аналізатор використовується в моделюючому сервісному центрі ШВМ як складова частина розподіленого паралельного моделюючого середовища [4].

Література

1. Абрамов Ф.А., Фельдман Л.П., Святный В.А. Моделирование динамических процессов рудничной аэрологии. Киев, Наукова думка, 1981г. 283с.
2. Святний В.А. Проблеми паралельного моделювання складних динамічних систем.- Наукові праці ДонДТУ, серія ІКОТ, вип. 6, Донецьк, 1999, С. 6-14.
3. Цой С., Цхай С.М. Прикладная теория графов. - Алма-Ата: Наука, 1971, 500 с.
4. W. Bär, V. Lapko, O. Moldovanova, A. Pererva, D. Rasinkov, V. Svjatnyj: Das Simulations- und Service-Zentrum für automatisierte Grubenbewetterungsnetze. In: D. Möller (Hrsg), Simulationstechnik, 14. Symposium in Hamburg, September 2000, S.223-228.

Дата надходження до редакції 12.06.2005 р.