УДК 004.421.2:519.17

С.Д. ПОГОРІЛИЙ, О.О. КАМАРДІНА, Ю.С. КОРДАШ

ПРО ПІДВИЩЕННЯ ШВИДКОДІЇ АЛГОРИТМІВ ФОРМУВАННЯ МІНІМАЛЬНОГО ВКРИВАЮЧОГО ДЕРЕВА

Abstract: *Prim's and Kruskal's sequential algorithms are formalized, which regular schemes are built. A method for paralleling is suggested and parallel regular schemes of algorithms are formalized for release on cluster architectures.*
Key words: *RSA, PRSA, equivalent scheme transformation, minimal spanning tree, parallelism, graph.*

Анотація: *Формалізовано послідовні алгоритми Пріма та Крускала, для яких побудовано регулярні схеми. Запропоновано підхід до розпаралелювання та сформовано паралельні регулярні схеми алгоритмів для їх подальшої реалізації на кластерних архітектурах.*
Ключові слова: *РСА, ПРСА, еквівалентні перетворення схем, мінімальне вкриваюче дерево, паралелізм, граф.*

Аннотация: *Формализованы последовательные алгоритмы Прима и Крускала, для которых были построены регулярные схемы. Предложен подход к распараллеливанию, и сформулированы параллельные регулярные схемы алгоритмов для их дальнейшей реализации на кластерных архитектурах.*
Ключевые слова: *РСА, ПРСА, эквивалентные преобразования схем, минимальное покрывающее дерево, параллелизм, граф.*

1. Вступ

Існує багато задач, які вирішуються за допомогою багатьох методів теорії графів. Наприклад, задачі визначення шляху з мінімальною вартістю, який поєднує усі вузли графа. До таких задач зводиться проектування телефонних мереж або електронних мікросхем. Мета подібних задач – з'єднання вузлів деякого призначення між собою з мінімальними витратами ресурсів.

Для того, щоб вирішити цю проблему, існують добре відомі алгоритми Пріма, Борувки та Крускала [1, 2] формування мінімального вкриваючого дерева. Метою даної роботи є не тільки вирішення поставленої проблеми, а й удосконалення існуючих алгоритмів за допомогою сучасних технологій. Основною вимогою до будь-яких алгоритмів є час виконання. Особливо це актуально для задач великої розмірності.

Одним із способів зменшення часу виконання є розпаралелювання алгоритму або його частин з метою подальшої реалізації на кластерній архітектурі. Перетворити алгоритм на паралельний можна за допомогою досвіду, інтуїції або апарату еквівалентних перетворювань. У цій роботі буде розглянуто формування регулярних схем алгоритмів (РСА) [3] та отримання паралельних регулярних схем алгоритмів (ПРСА) для алгоритмів Пріма та Крускала й обґрунтування результатів за допомогою апарату еквівалентних перетворень. Для виконання цих перетворень застосовано математичний апарат модифікованих систем алгоритмічних алгебр (САА-М) [4].

Розглянемо послідовні алгоритми та виділимо в них місця, які є найбільш важкими з точки зору обчислення.

ISSN 1028-9763. Математичні машини і системи, 2005, №4

Формалізація послідовних алгоритмів

Розглянемо формалізацію алгоритму Пріма. Вважаємо, що неорієнтований граф представлений матрицею суміжності G, розмір якої $V*V$, де V – кількість вершин у графі. Введемо такі позначення:

C – поточна вершина;

FR - черга з пріоритетами („бахрома" [1]);

MST – мінімальне вкриваюче дерево (дерево MST);

Init(C) – функція введення початкової вершини;

C – початкова вершина.

Init(C);

while (*кількість ребер в MST менше V*)

{

Запис у FR ребер, які виходять з C та ведуть до недеревних вершин;

If (*в FR існують ребра, які ведуть в одну вершину*) залишити ребро з найменшою вагою, а інші видалити;

Вибрати з FR ребро з мінімальною вагою;

Додати в MST вибране ребро;

C = вершина, приєднана до MST на поточній ітерації;

}

Розглянемо приклад виконання цього алгоритму. Введемо позначення ребер таким чином: 0 – 1(4), де 0 та 1 – номери вершин; (4) – вага ребра. Нехай граф заданий таким набором ребер:

0 – 1 (4);	1 – 4 (10);
0 – 2 (16);	1 – 5 (15);
0 – 3 (21);	2 – 5 (13);
1 – 2 (11);	3 – 4 (5);
1 – 3 (9);	4 – 5 (12).

Кроки виконання алгоритму зображені на рис. 1. Тонкими лініями позначені існуючі ребра, сірим кольором виділені ребра, що включені в чергу з пріоритетами, чорним кольором виділені ребра, що включені в дерево MST. Черга з пріоритетами вказана під графом на кожному кроці виконання алгоритму.

Початкова вершина – вершина 0. Ця вершина вже входить у дерево MST. Всі ребра, які виходять з вершини 0, становлять чергу з пріоритетами (рис. 1a). Ребро з найменшою вагою в черзі – ребро 0–1. Воно додається до дерева MST та вилучається з черги з пріоритетами. З'являються ребра 1–2 та 1–3, які можуть наблизити вершини 2 та 3 до дерева MST. Ребра 1–2 та 1–3 легші за ребра 0–2 та 0–3, тому черга з пріоритетами змінюється (рис. 1b). В черзі з пріоритетами ребром з мінімальною вагою є ребро 1–3. Воно вилучається з черги з пріоритетами та додається до дерева MST. Воно також наближує вершину 4 до дерева MST (рис. 1c). Алгоритм продовжує своє виконання, доки в дерево MST не буде включено (V–1) ребер (рис. 1f).

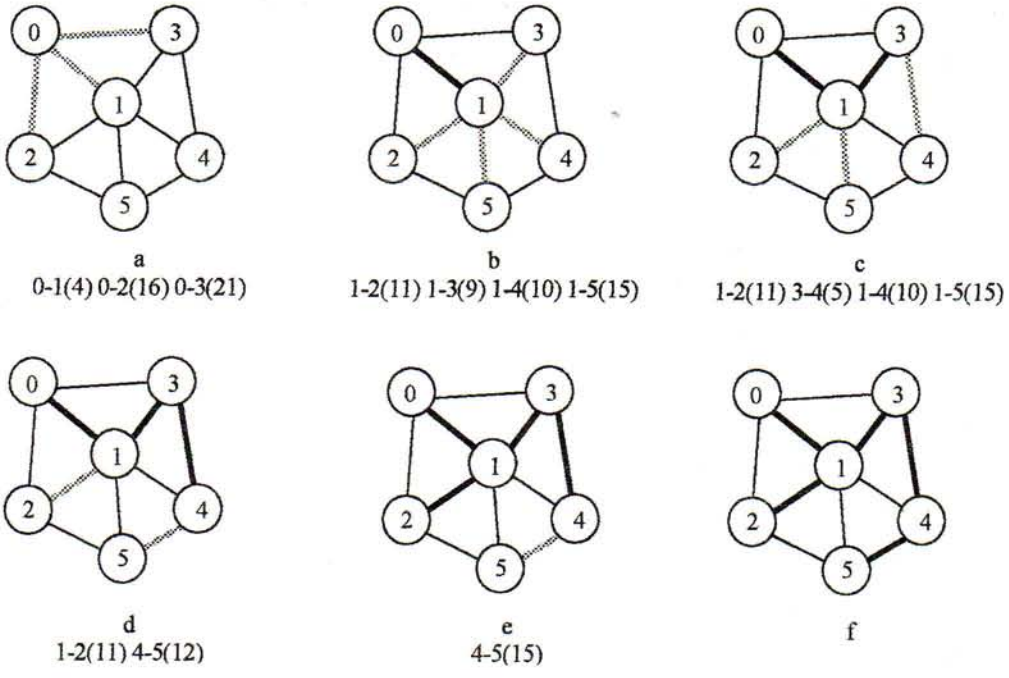

Рис. 1. Приклад роботи алгоритму Пріма

Дерево MST: 0-1(4) 1-3(9) 3-4(5) 1-2(11) 4-5(12).

Представимо алгоритм у вигляді інтерпретованої схеми.

Запис у FR ребер, які виходять з C та ведуть до недеревних вершин, можна представити як цикл з V ітерацій. Тілом цього циклу є альтернатива, в якій перевіряється, чи існує ребро до вершини та чи не належить ця вершина дереву MST.

$$i<V\ \{\mathrm{G[C,\ i]}\neq 0 \cup i\notin MST\ \ (FR \leftarrow G[C,i],E)\}.$$

Блок, який обробляє чергу з пріоритетами:

If (*в FR існують ребра, які ведуть в одну вершину*) залишити ребро з найменшою вагою, а інші вилучити з черги.

Чергу з пріоритетами можна представити як структуру даних у вигляді двонаправленого (для зручності) списку. Його можна поділити на дві частини: ребра з пріоритетом були включеними до складу дерева MST та нові ребра, які виходять з поточної вершини. Черга з пріоритетами буде представлена парами ребер, які ведуть в одні й ті ж вершини. З цих пар треба вилучати ребра з більшою вагою. Така обробка може бути представленою за допомогою двох вкладених циклів та альтернативи.

$$fr1\neq fr\_border.next\{fr2\neq NULL\ \{fr1.e.v2=fr2.e.v2 \cap fr1.e.weight > fr2.e.weight\ (fr1.e=fr2.e,E)*delete(fr2.e)\}*$$
$$*\ fr1=fr1.next\},$$

де $fr1$ та $fr2$ – вказівники в FR; e – ребро в черзі з пріоритетами; $v2$ – номер вершини, в яку веде ребро; fr_border – вказівник на кінець лівої частини черги з пріоритетами.

Якщо реалізувати паралелізм за описаною концепцією, то цю частину можна спростити. Значення вказівників $fr1$ та $fr2$ можна змінювати в тілі одного циклу:

$$fr1 \neq fr\_border next\{fr1.e.v2 = fr1.e.weight > fr2.e.weight (fr1.e = fr2.e, E) * delete(fr2.e) * fr1 = fr1.next * * fr2 = fr2.next\}.$$

Пошук ребра з мінімальною вагою можна представити у вигляді циклу з альтернативою:

$$\min = fr.e.weight * _{fr \neq NULL} \{fr = fr.next * _{fr.e.weight < \min} (\min = fr * mark(fr))\},$$

де fr – це вказівник в FR, а $mark(fr)$ – функція помітки ребра з мінімальною вагою.

Функцію додавання ребра в дерево MST позначимо як $addMST(mark)$.

Тоді вся формула алгоритму Пріма матиме вигляд

$$Init(c) * _{MST.size < V} \{_{i<V} \{_{G[C,i] \neq 0 \cup i \notin MST} (FR \leftarrow G[C,i], E)\} *$$

$$fr1 \neq fr\_border next\{fr1.e.v2 = fr1.e.weight > fr2.e.weight (fr1.e = fr2.e, E) * delete(fr2.e) * fr1 = fr1.next * * fr2 = fr2.next\} * \min = fr.e.weight * _{fr \neq NULL} \{fr = fr.next * _{fr.e.weight < \min} (\min = fr * mark(fr))\} * addMST(mark)\}.$$

Щоб застосувати еквівалентні перетворення до алгоритму, треба його представити у вигляді РСА.

Позначення для предикатів:

$MST.size < V \Rightarrow \alpha$;

$i < V \Rightarrow \phi$;

$G[C,i] \neq 0 \Rightarrow \beta$;

$i \notin MST \Rightarrow \gamma$;

$fr1 \neq fr\_border.next \Rightarrow \eta$;

$fr1.e.v2 = fr2.e.v2 \Rightarrow \mu$;

$fr1.e.weight > fr2.e.weight \Rightarrow \nu$;

$fr.e.weight < \min \Rightarrow \rho$;

$fr \neq NULL \Rightarrow \pi$.

Позначення для операторів:

$Init(C) \Rightarrow A$;

$FR \leftarrow G[C,i] \Rightarrow B$

$fr1.e = fr2.e \Rightarrow C$;

$delete(fr2.e) \Rightarrow D$;

$fr1 = fr1.next * fr2 = fr2.next \Rightarrow H$;

$\min = fr.e.weight \Rightarrow J$;

$fr = fr.next \Rightarrow K$;

$\min = fr * mark(fr) \Rightarrow L$;

$addMST(mark)\} \Rightarrow M$.

Тоді формула може бути представлена у такому вигляді:

$$A *_{\alpha} \{\phi \{_{\beta \wedge \gamma} (B,E)\} *_{\eta} \{_{\mu \wedge \nu} (C,E) * D * H\} * J *_{\pi} \{K *_{\rho} (L,E)\} * M\}.$$

Останнє співвідношення є послідовною РСА алгоритму Пріма.

Виконаємо формалізацію алгоритму Крускала. Як і раніше, неорієнтований граф представлений матрицею суміжності G, розмір якої $V * V$, де V – кількість вершин у графі. Введемо позначення:

Forest – ліс піддерев MST;

MST_i – піддерево MST, де $i = [1..V]$;

$List$ – список ребер графа.

Формування невідсортованого списку $List$ ребер графа;

Сортування Quicksort;

$j = 0;$

while (*кількість ребер в Forest менше* V)

{

if(*вершини ребра List[i] з'єднують два різних піддерева MST*) об'єднати ці два піддерева MST;

$j = j + 1;$

}

Розглянемо приклад виконання цього алгоритму (рис. 2). Позначимо ваги ребер у вигляді вже відсортованого списку:

0 – 1 (4);	4 – 5 (12);
3 – 4 (5);	2 – 5 (13);
1 – 3 (9);	1 – 5 (15);
1 – 4 (10);	0 – 2 (16);
1 – 2 (11);	0 – 3 (21).

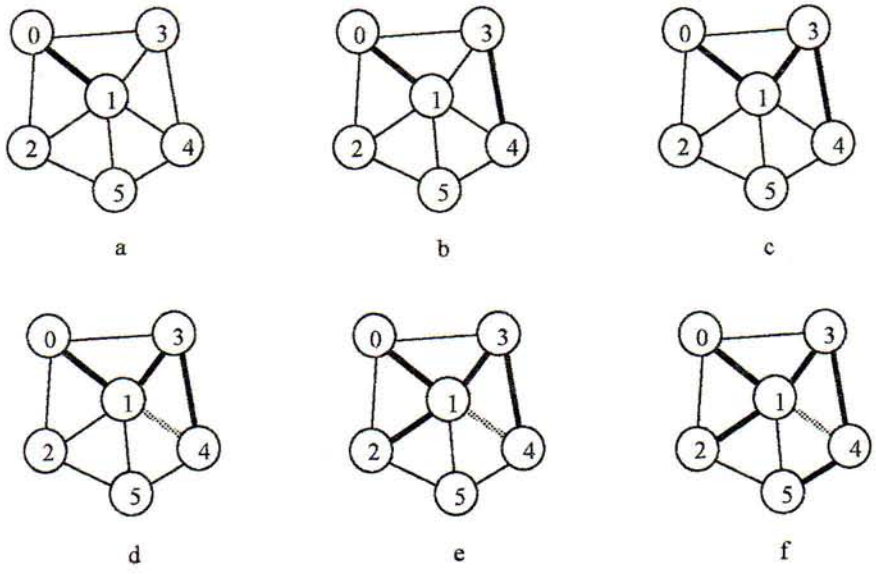

Рис. 2. Приклад роботи алгоритму Крускала

Всі вершини графа представляють піддерева MST. Тепер ці піддерева з'єднуються між собою ребрами з найменшою вагою (рис. 2a–2c). Ребро 1–4 не може бути приєднаним до дерева MST, тому що воно поєднує вершини одного й того ж піддерева (рис. 2d). Алгоритм припиняє роботу, коли до дерева MST приєднано (V–1) ребер (рис. 2f).

Результат роботи алгоритму: 0-1(4) 3-4(5) 1-3(9) 1-2(11) 4-5(12).

Процес формування списку невідсортованих ребер можна представити двома вкладеними циклами та альтернативою, яка перевіряє наявність ребра в матриці суміжності.

$$i<V\{j<i\{G[i,j]\neq 0\,(List \leftarrow G[i,j])\}\}.$$

Як вже було зазначено, $List$ – це невідсортований список усіх ребер графа. Швидке сортування позначимо як $S(List)$.

Обробка ребер у відсортованому списку представляється як цикл з вкладеною альтернативою. |Forest| – кількість ребер у лісі піддерев MST. MergeMST – оператор з'єднання двох піддерев MST.

$$|Forest|<V\{List[j].v1\notin MST^{a} \cap List[j].v2\notin MST^{b}(mergeMST,E)\}.$$

Тоді вся формула алгоритму Крускала матиме вигляд

$$i<V\{j<i\{G[i,j]\neq 0(List\leftarrow G[i,j])\}\}^{*}S(List)^{*}$$

$$^{*}|Forest|<V\{List[j].v1\notin MST^{a} \cap List[j].v2\notin MST^{b}(mergeMST,E)\}.$$

Побудуємо РСА, зробивши всі необхідні позначення.

Позначення для предикатів:

$i < V \Rightarrow \alpha$;

$j < i = \beta$;

$G[i, j] \neq 0 \Rightarrow \gamma$;

$|Forest| < V \Rightarrow \phi$;

$List[j].v1 \in MST^{a} \cap List[j].v2 \in MST^{b} \Rightarrow \eta$.

Позначення для операторів:

$List \leftarrow G[i, j] \Rightarrow A$;

$mergeMST \Rightarrow C$;

$S(List) \Rightarrow S$.

З урахуванням позначень РСА запишеться як

$$\alpha\{\beta\{\gamma(A,E)\}\}^{*}S^{*}\phi\{\eta(C,E)\}.$$

Останнє співвідношення є послідовною РСА алгоритму Крускала.

3. Формування схем паралельних алгоритмів

Розглянемо формування ПРСА алгоритму Пріма.

Найбільший інтерес для розпаралелювання мають цикли. В даному алгоритмі в єдиному циклі операція обробки черги з пріоритетами споживає найбільше обчислювальних ресурсів. Час обробки черги з пріоритетами можна скоротити, якщо ввести паралелізм за даними. Чергу з пріоритетами розіб'ємо на дві частини: ребра з пріоритетом бути приєднаними до дерева MST, та нові ребра, які на поточній ітерації можуть наблизити недеревні вершини до дерева MST. Ці частини розіб'ємо на процеси способом, показаним на рис. 3.

Такий розподіл можливий, коли порядок ребер у двох частинах однаковий, тобто коли існують ребра, які ведуть з кожної вершини в кожну. Якщо граф ненасичений, то відсутність ребер у матриці суміжності можна позначити з максимально можливою вагою. Алгоритм сприйматиме їх як існуючі у графі. Потім такі ребра у процесі роботи алгоритму будуть видалені як дуже важкі.

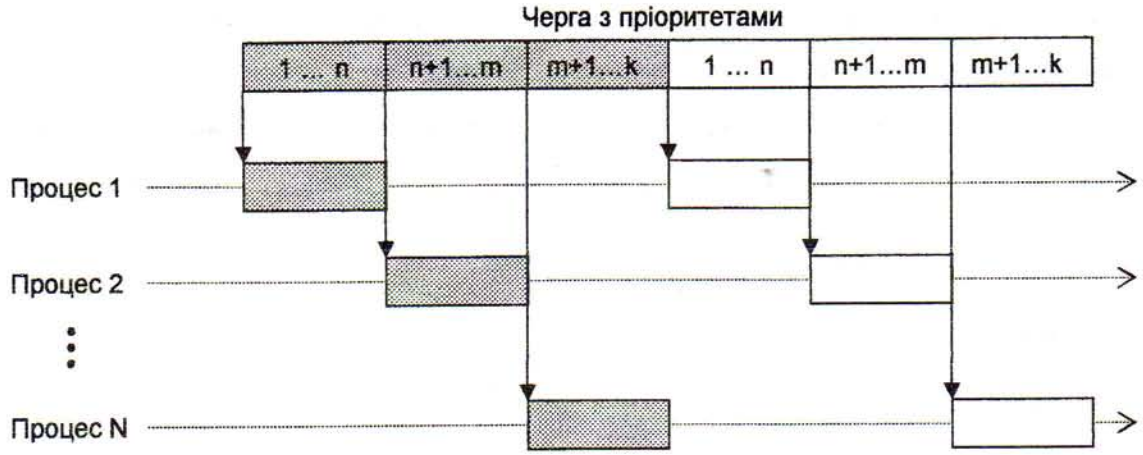

Рис. 3. Розпаралелювання алгоритму Пріма

Пошук ребра з мінімальною вагою теж можна прискорити за допомогою схожого спрощеного методу. В результаті обробки черги з пріоритетами залишиться тільки частина ребер з пріоритетом бути приєднаними до дерева MST. Кожен процес обробляє частину черги з пріоритетами, в якій він може знайти ребро з мінімальною вагою та повернути його головному процесу. Потім з повернених дочірніми процесами ребер вибирається ребро з мінімальною вагою.

За допомогою апарату еквівалентних перетворень отримаємо паралельний алгоритм Пріма, в якому використовуються описані концепції. Для цього застосуємо формулу $_{\alpha}(A,E)=\underline{\alpha}\,A$, щоб позначити фільтри [5]:

$$A^{*}{}_{\alpha}\{\phi\{\underline{\beta\wedge\gamma}B\}^{*}{}_{\eta}\{\underline{\mu\wedge\nu}C*D*H\}*J^{*}{}_{\pi}\{K*\underline{\rho}L\}*M\} \tag{1}$$

Умовно позначимо

$$\eta=\eta_1\vee\eta_2\vee\ldots\vee\eta_m;$$

$$\eta_i => fr1\neq fr\_begin+(fr\_border.next/m)*i,$$

де $i\in[1..m]$; m – кількість процесів; fr_begin – початок черги з пріоритетами; fr_border – кінець лівої частини черги з пріоритетами.

Тоді (1) буде мати вигляд

$$A^{*}{}_{\alpha}\{\phi\{\underline{\beta\wedge\gamma}B\}^{*}{}_{\eta_1\vee\eta_2\vee\ldots\vee\eta_m}\{\underline{\mu\wedge\nu}C*D*H\}*J^{*}{}_{\pi}\{K*\underline{\rho}L\}*M\}. \tag{2}$$

Якщо умова циклу – диз'юнкція, то (2) можна представити у вигляді паралельної формули. Для зручності позначимо $\underline{\mu\wedge\eta}\,C*D*H => Z$:

$$A^{*}{}_{\alpha}\{\phi\{\underline{\beta\wedge\gamma}B\}*\#_{\eta_1}\{Z\}\|\,{}_{\eta_2}\{Z\}\|\ldots\|\,{}_{\eta_m}\{Z\}\#*J^{*}{}_{\pi}\{K*\underline{\rho}L\}*M\}. \tag{3}$$

У цій схемі вказівники $fr1$ та $fr2$ локальні для кожної гілки виконання. Символами "#...#" позначається частина алгоритму, оператори якої повинні виконуватися паралельно. А символами "||" розділяються оператори, що належать до паралельних гілок.

Скориставшись розподільчою властивістю паралельного виконання за формулою $\#A \| B\#*C = \#A*C \| B*C\#$, можна розподілити пошук ребра з мінімальною вагою в черзі з пріоритетами. Позначимо за $X => J*_{\pi}\{K*\underline{\rho}L\}$:

$$A*_{\alpha}\{_{\phi}\{\underline{\beta \wedge \gamma}B\}*\#_{\eta_1}\{Z\}*X \|_{\eta_2}\{Z\}*X \| \ldots \|_{\eta_m}\{Z\}*X\#*Y*M\}. \quad (4)$$

де Y – пошук ребра з мінімальною вагою серед результатів пошуку дочірніх процесів.

Схеми (3) та (4) представляють ПРСА алгоритму Пріма з дотриманням описаних концепцій.

Розглянемо формування ПРСА алгоритму Крускала. В цьому алгоритмі використовується швидке сортування [2]. Більшу частину обчислень займає сортування списку ребер. Сортування можна прискорити, якщо розподілити список ребер між процесами (рис. 4). Розмір частин списку ребер може бути довільним, але бажано, щоб окремі частини були рівними. Після сортування ми отримаємо декілька відсортованих частин всього списку ребер, вказівники в яких позначають ребра з найменшою вагою в кожній з частин списку. Тепер треба вибирати ребра з мінімальною вагою з тих ребер, які помічені вказівниками. В частині, з якої було вибрано ребро для з'єднання з деревом MST, вказівник переходить на наступне ребро.

Рис. 4. Розпаралелювання алгоритму Крускала

Для РСА алгоритму Крускала застосуємо операцію фільтрації

$$\alpha\{\beta\{\underline{\gamma}A\}\}*S*_{\phi}\{\underline{\eta}C\}. \quad (5)$$

Умовно позначимо $List = part_1 \cup part_2 \cup \ldots \cup part_n$, де $part_i$ – це частини списку $List$, які не перетинаються; n – кількість процесів. Позначимо сортування частин списку через $S_1 \ldots S_n$. Тепер будемо сортувати частини списку:

$$\alpha\{\beta\{\underline{\gamma}A\}\}*S_1*S_2*\ldots*S_n*_{\phi}\{\underline{\eta}C\}. \quad (6)$$

Сортування однієї частини не впливає на сортування інших частин, тому за формулою ${}_{\alpha}\{A*B\}={}_{\alpha}\{A\}\,\|\,{}_{\alpha}\{B\}$ розпаралелимо сортування. Операції по вибору ребра для включення в дерево MST позначимо через Y:

$$\alpha\{\beta\{\underline{\gamma}A\}\}*\#S_1\,\|\,S_2\,\|\,\ldots\|\,S_n\#*Y^*_{\phi}\{\underline{\eta}\,C\}. \qquad (7)$$

4. Висновки

Виконано формалізацію послідовних алгоритмів Пріма та Крускала, для яких були побудовані РСА. За допомогою еквівалентних перетворювань отримано ПРСА для алгоритмів Пріма (3, 4) та Крускала (7), які реалізовано на кластерних архітектурах.

У кожному випадку використовувалася концепція паралелізму за даними. Особливість алгоритмів полягає в тому, що вони обробляють велику кількість даних. Розпаралелювання алгоритмів за керуванням не дало такого приросту у швидкості обчислень, як розпаралелювання за даними, яке зменшує обсяг роботи для кожного з процесів.

СПИСОК ЛІТЕРАТУРИ

1. Седжвик Р. Фундаментальные алгоритмы на С++. – Москва – Санкт-Петербург – Киев, 2002. – Глава 5: Алгоритмы на графах. – С. 250–265.
2. Ривест Р., Кормен Т., Лейзерсон Ч. Алгоритмы: построение и анализ. – МЦНМО, 2000. - 898с.
3. Погорілий С.Д. Автоматизація наукових досліджень. Основоположні математичні відомості. Програмне забезпечення. – Киев: ВПЦ "Київський Університет", 2002. – 292с.
4. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра, языки, программирование. – Киев: Наукова думка, 1989. – 380с.
5. Погорілий С.Д., Камардіна О.О. Дослідження та створення інструментальних засобів автоматизованої трансформації схем алгоритмів // Проблемы программирования. – 2004. – № 1– 2. – С. 417–421.

Стаття надійшла до редакції 06.04.2005