УДК 004.4

С.Д. Погорілий, Д.Ю. Вітель

ВИКОРИСТАННЯ МЕРЕЖ ПЕТРІ ДЛЯ ПРОЕКТУВАННЯ ПАРАЛЕЛЬНИХ ЗАСТОСУВАНЬ

Створено сукупність САА-М схем, що відповідають найбільш поширеним workflow-шаблонам. Це надає змогу використовувати обидва підходи з метою перетворення представлень алгоритму і використання їх особливостей та переваг (проведення моделювання чи набору формальних трансформацій). З використанням розглянутих шаблонів побудовано узагальнену мережу Петрі алгоритму Флойда-Уоршала для набору паралельних потоків. Ця мережа справедлива для ряду архітектур (SMP, MPP, архітектури технології GPGPU: Fermi, G80), проте потребує певної конкретизації своїх елементів для кожної з них, що частково розглянуто в роботі.

Вступ

Алгоритм вирішення певної задачі може бути як централізованим, так і розподіленим [1]. У першому випадку він виконується на одному виділеному вузлі мережі, який володіє великим обсягом ресурсів (пам'ять, кількість паралельних потоків обробки тощо). Тому такий вузол є мультипроцесором (у частинному випадку – реалізація Simmetric MultiProcessing – SMP-архітектури). В другому випадку алгоритм виконується на окремих вузлах мережі, що обмінюються даними між собою. Це відповідає концепції мультикомп'ютерних систем (Massively Parallel Processing – MPP-архітектурі) – систем із масовим паралелізмом.

Слід зазначити, що реалізація алгоритму може також бути орієнтована на гетерогенні системи (декілька виділених вузлів виконують алгоритм на всіх своїх процесорах, обмінюючись один з одним локальними даними). Доцільно також виділити застосування, які ґрунтуються на технології GPGPU (General purpose graphic processing unit) і використовують графічні адаптери архітектур Fermi, Tesla [2]. Ці системи пропонують принципово інші методи реалізації паралельних застосувань і орієнтовані на певний клас задач, на якому вони демонструють максимальну продуктивність.

Наявність значної сукупності паралельних архітектур суттєво диференціює методи побудови паралельних програм для них. Це створює додаткові ускладнення при розробці високопродуктивних застосувань, оскільки вимагає від розробника необхідних знань особливостей побудови відповідної паралельної системи, а також відповідних парадигм паралельного програмування. Тому є важливим етап формалізованого проектування застосувань.

Потоки робіт (workflows) [3, 4] є представленням паралельного алгоритму чи системи у вигляді послідовності взаємопов'язаних етапів чи кроків. Вони є фактично функціональною декомпозицією, оскільки дозволяють виокремити в ній певні функції чи завдання (task). Кожне з таких завдань складається з процесів, що можуть виконуватись як паралельно, так і послідовно. Потоки робіт широко застосовуються як представлення реальної роботи систем різного роду. Як метод формалізації окремих складових частин у такий спосіб представленої системи в більшості випадків використовують математичний апарат мереж Петрі. Це надає можливість проводити моделювання роботи та аналіз темпоральних характеристик алгоритму або створюваної комп'ютерної системи.

Шаблони потоків робіт та САА-М

Далі розглянуто шаблони паралельного програмування і відповідні їм нотації схем паралельних алгоритмів з використанням математичного апарата модифікованих систем алгоритмічних алгебр (САА-М) [5 – 7].

Шаблон «Parallel split» описує розбиття гілки виконання на дві або більше, що виконуються паралельно (рис. 1).

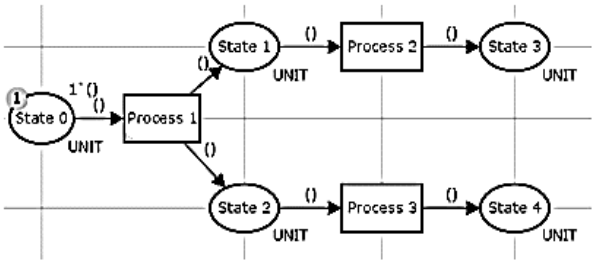

Рис. 1

САА-М схема:

$$parallelSplit = P_1 * (P_2 \dot{\vee} P_3)$$

– у випадку безпечної мережі (що залежить від способу її застосування в ієрархічній мережі). Якщо ж вхідне місце може приймати більше ніж одну мітку, то схема буде:

$$parallelSp\ lit = \dot{\bigvee_{i=1}^{n}} (P_1 * (P_2 \dot{\vee} P_3)).$$

Шаблон «Synchronization» – злиття декількох гілок виконання в одну послідовну гілку, виконання якої починається після закінчення роботи паралельних гілок (рис. 2).

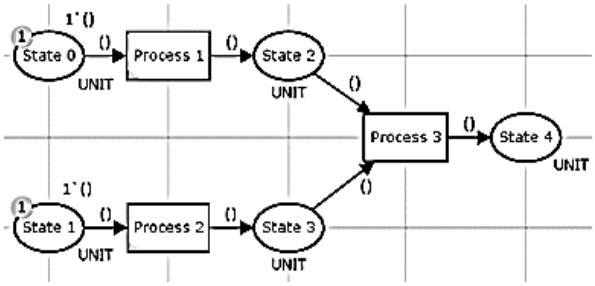

Рис. 2

Вважаючи мережу безпечною, схема шаблону:

$$sync = (P_1 * T(\alpha) * S(\beta) * P_3) \dot{\vee} (P_2 * T(\beta) * S(\alpha)),$$

де $T(x)$ – оператор встановлення умови x, $S(x)$ – оператор очікування встановлення цієї умови. У випадку небезпечної мережі схема узагальнюється аналогічно до попередніх схем.

У програмуванні реалізується шляхом очікування потоком завершення іншого потоку та виконання наступної дії. В цьому випадку даному потоку в мережі відповідатиме одна з паралельних та послідовна гілки. Слід зазначити, що у випадку небезпечної мережі можливі тупикові ситуації (англ. deadlocks).

Шаблон «Exclusive choice» – шаблон вибору певної гілки виконання в залежності від заданої умови (рис. 3). Виконується лише обрана гілка.

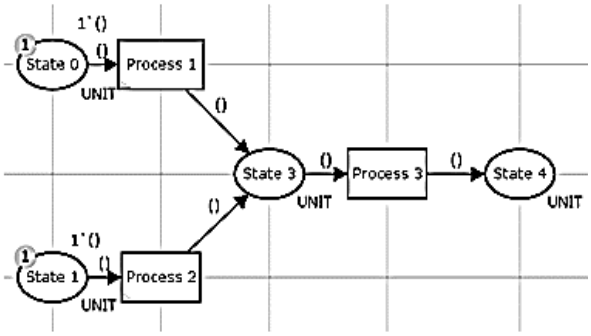

Рис. 3

САА-М схема для безпечної мережі: $exclChoice = P_1 *_{cond} (P_2 \vee P_3)$ – 2 гілки виконання. Для n гілок отримаємо схему:

$$exclChoice = P_0 * \prod \begin{pmatrix} \alpha_1 & \dots & \alpha_n \\ P_1 & \dots & P_n \end{pmatrix}.$$

Відповідає оператору if, switch (case) в мовах програмування. Для випадку багатьох гілок виконання має бути передбачена гілка за замовчуванням. В іншому випадку потік виконання може бути просто знищений.

«Simple merge» – шаблон злиття гілок виконання (рис. 4), що створені мережею Exclusive choice.

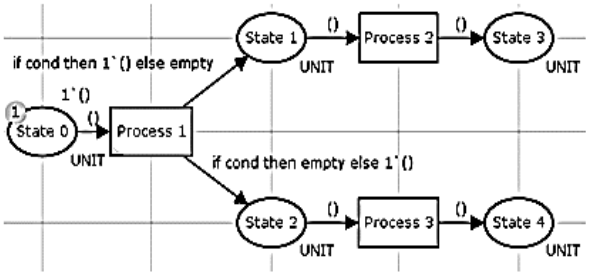

Рис. 4

Тобто виконується одна з послідовних гілок, а після цього виконуються спільні для них задачі.

САА-М схема:

$$simpleMerge = \prod \begin{pmatrix} \alpha_1 & \dots & \alpha_n \\ P_1 & \dots & P_n \end{pmatrix} * P_{n+1}.$$

Місце P_1 має містити не більше однієї мітки.

Вибір декількох гілок (Multichoice) – у залежності від заданих умов відповідна гілка або буде виконуватись паралельно з іншими, або ні (рис. 5).

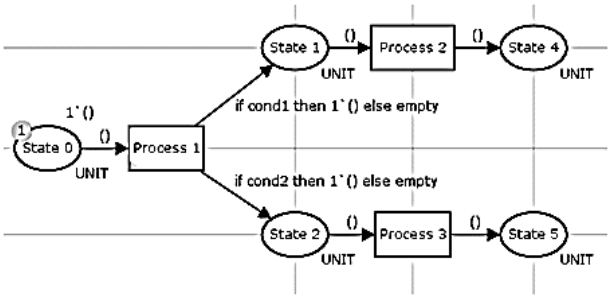

Рис. 5

САА-М схема:

$$multiChoic\ e = P_0 * \dot{\vee}_{x=1}^{n} \left(\underline{cond_x} * P_x \right).$$

Використана альфа-фільтрація для обриву відповідної гілки обчислень.

Злиття гілок (Multimerge) – створені паралельні гілки виконують специфічні для себе завдання, потім виконують завдання, що спільні для усіх (рис. 6).

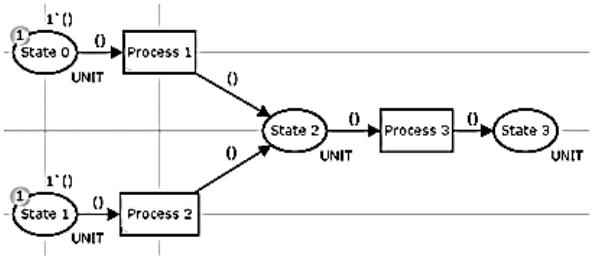

Рис. 6

САА схема:

$$multiMerge = \dot{\vee}_{x=1}^{n} \left(P_x * P \right).$$

Етап (Milestone) – виконання певного процесу продовжується, якщо інший процес знаходиться в певному стані (рис. 7).

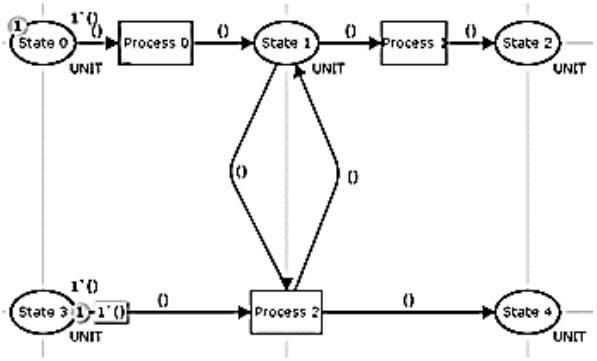

Рис. 7

САА-М схема:

$$Milestone = (P_0 * S(\alpha) * P_1) \dot{\vee} (P_2 * T(\alpha)).$$

Цикли (Arbitrary Cycles) – шаблон, що описує циклічне виконання завдань. Об'єднує усі можливі цикли (рис. 8).

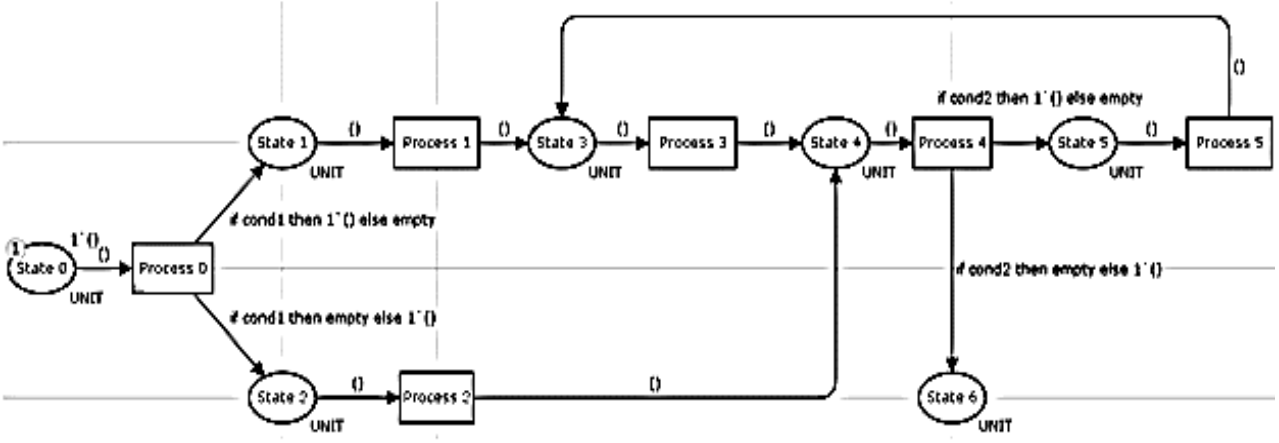

Рис. 8

САА схема**:**

$$cicles = A^* \underset{cond1}{(} (B^* D) \vee C)^* E^* \underset{cond2}{\{} F^* D^* E\}.$$

Тут E – не тотожний оператор (у термінах САА-М), а умова виходу з циклу відбувається тоді коли друга умова буде false.

Крім зазначених шаблонів існують інші, опис яких можна знайти в [3, 4]. Їх САА-М схеми можна подати в аналогічний спосіб.

Проектування алгоритму з використанням математичного апарату мереж Петрі

Апарат мереж Петрі може бути використаним як у процесі нисхідного, так і висхідного проектування. Проте, для складних систем зазвичай використовують комбінацію цих двох підходів. У будь-якому випадку на певному етапі виникає питання про формалізацію структур даних системи чи алгоритму. Математичний апарат кольорових мереж Петрі надає можливість створювати відповідні типи даних.

Слід зазначити, що при декомпозиції алгоритму доцільно використовувати наведені шаблони потоків робіт. Розглянемо застосування мереж Петрі для реалізації паралельного алгоритму Флойда – Уоршала [8] з використанням шаблонів: Arbitrary Cycles – три вкладені цикли алгоритму можна представити у вигляді однієї мережі цього шаблону з складеною умовою виходу; Multiple Instances with a Priori Design-Time Knowledge – використовується для породження паралельних потоків; Parallel Split – використовується як для створення паралельних потоків (проте в однопотоковій мережі Петрі алгоритму використаний для паралельної ініціалізації змінних-ітераторів); Sequence; Milestone – використаний для емалювання послідовного зчитування даних із пам'яті й т. д.

Далі побудована однопотокова мережа алгоритму (рис. 9). Це ієрархічна мережа, що має наступні підмережі: Iter – перевіряє умову закінчення алгоритму та повертає нове значення змінних циклу; Select – вибірка із сховища даних за відповідною адресою; Update – оновлення сховища за відповідною адресою. Сама дія алгоритму представлена у вигляді переходу process. Дана мережа є WF-мережею, проте вхідне і вихідне місця її об'єднані в одне – Confirmation. При цьому використані мережі шаблонів були дещо спрощені. Потік в мережі формалізується міткою виконання, що на різних етапах може мати різні типи даних. Інший підхід до представлення системного потоку полягає у виділенні окремого типу маркерів. Таким чином процес виконання полягає і переходах маркерів даного типу з одного місця на інше.

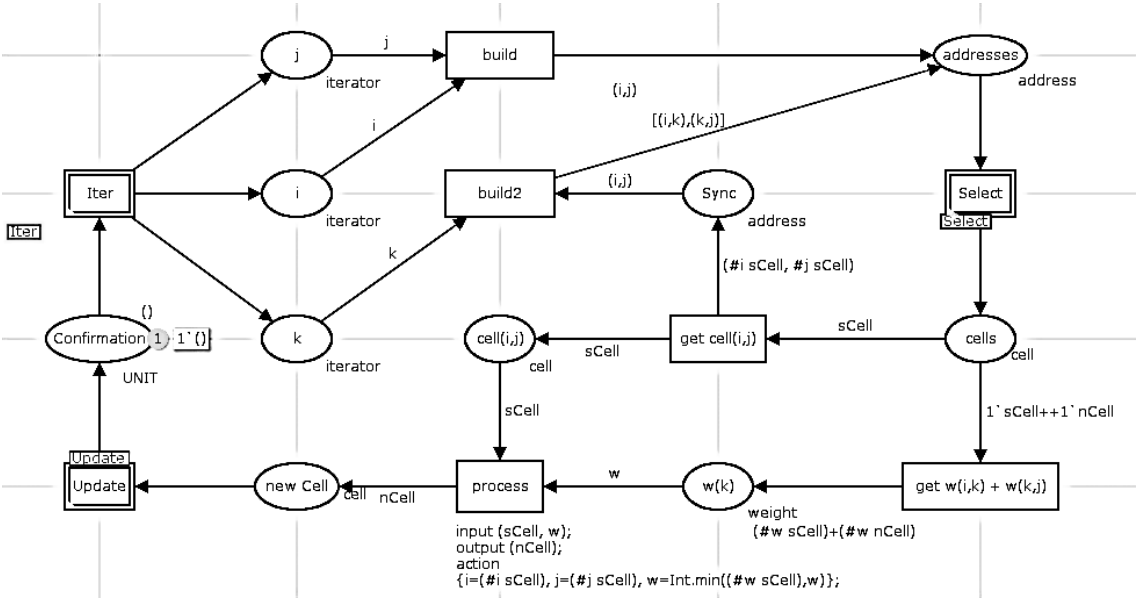

Рис. 9

Слід зазначити, що дана мережа уніфікує доступ до сховища даних, тобто мережа не залежить від організації самого сховища. Воно, в свою чергу, може бути як розподіленим у системі, так і централізованим. Далі наведено реалізації операцій роботи зі сховищем.

Select-мережа. Дана мережа (рис. 10) приймає на вхід адресу комірки пам'яті, що формалізована у вигляді двох індексів, та повертає саму комірку. При цьому місце address не є безпечним, що дозволяє фактично моделювати чергу адрес на зчитування даних. Саме зчитування йде послідовно.

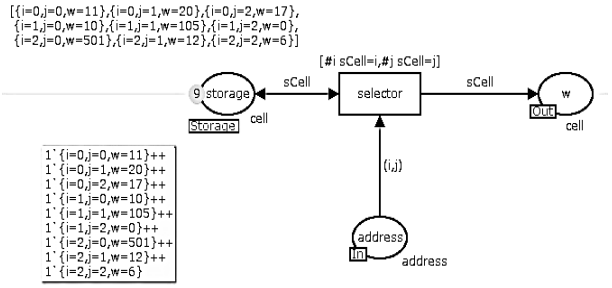

Рис. 10

Update-мережа. Приймає на вхід комірку пам'яті (тобто адресу і значення) для оновлення і видає підтвердження про оновлення сховища (рис. 11). Без даного підтвердження мережа була б описом асинхронної операції зміни.

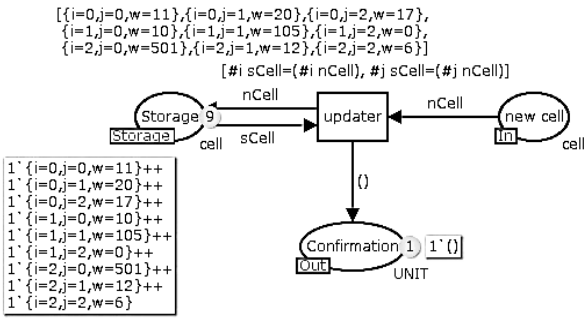

Рис. 11

Далі наведемо *мережу Iter* (рис. 12). Змінні циклів представлені у вигляді окремих місць, а cicle Controller забезпечує завершення роботи алгоритму. Використаний шаблон для циклічного виконання в загальній схемі має векторний ітератор.

Перехід до мережі для паралельних архітектур (рис. 13) здійснюється наступним чином. Кожен потік представлений у вигляді окремої мітки з індексом потоку. Локальні ресурси потоку помічаються також його індексом. Крім того в мережу додається синхронізація потоків.

У мережу додано додатковий етап ініціалізації – Initialize. Він призначений для створення паралельних потоків та ініціалізації їх локальних ресурсів. Перехід read file зчитує дані з файлу. Його використання в мережі пояснюється лише тим, що остання використовується в процесі

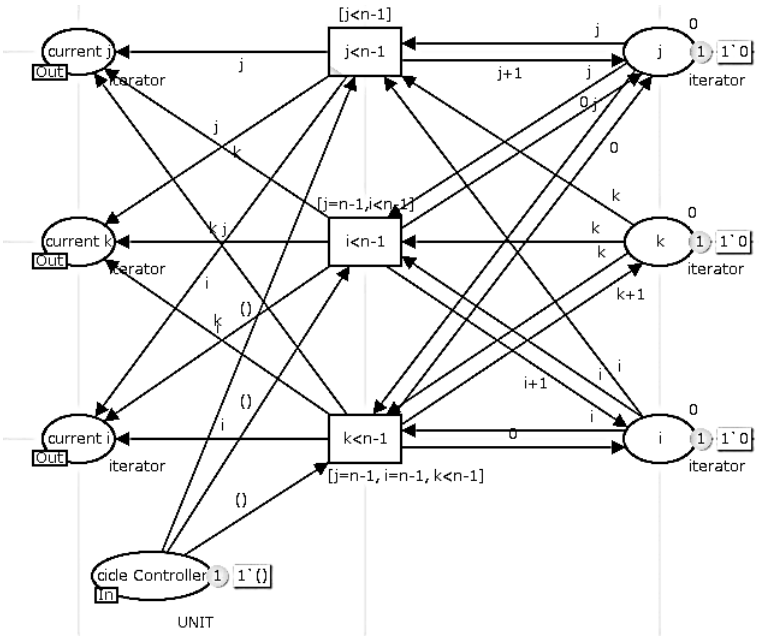

Рис. 12

моделювання роботи алгоритму та перевірки результатів. Проте з точки зору проектування даний блок може бути або відсутній або має ініціалізувати сховище даних.

Після виконання розглянутого переходу сховища Storage та Iterator storage зберігатимуть необхідні дані. Також будуть створені мітки потоків, подальше виконання яких не буде синхронізоване до переходу Barrier у мережі Get iterators (рис. 14).

Також потребують змін наведені мережі роботи зі сховищем. Було змінено тип даних вхідного місця мережі Select, що дозволило подавати їй на вхід список адресів, за якими йде послідовний вибір даних. При роботі із багатьма потоками слід також синхронізувати їх доступ до мереж Select та Update (зробити їх потокобезпечними).

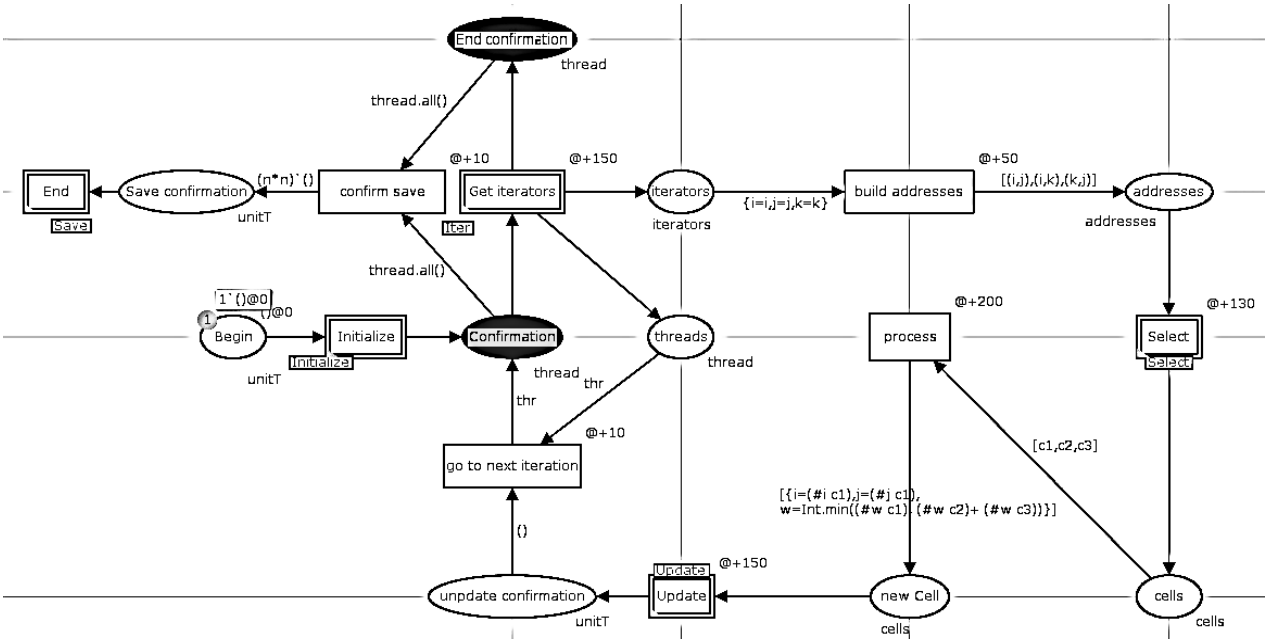

Рис. 13

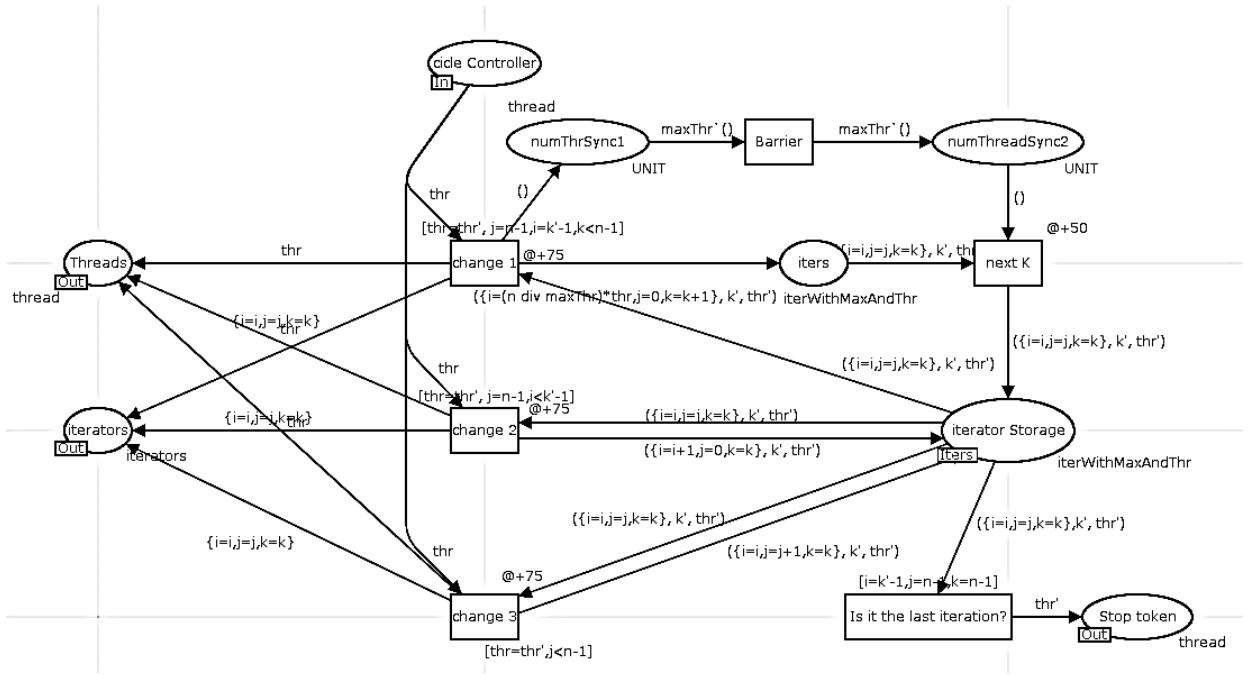

Рис. 14

Нова мережа Select (рис. 15) має наступні особливості. Кожен із потоків подає на її вхід свій список адрес. Внутрішня синхронізація за допомогою місць SyncInOut1, SyncInOut2 вибирає із утвореної черги списки по одному, послідовно обробляючи їх. Використаний шаблон Critical Section.

Останньою мережею, що не була розглянута є мережа End (рис. 17). Її задачі – конвертація даних у відповідний формат та запис у файл. Вона використовується при моделюванні. Для цілей проектування може бути замінена на іншу.

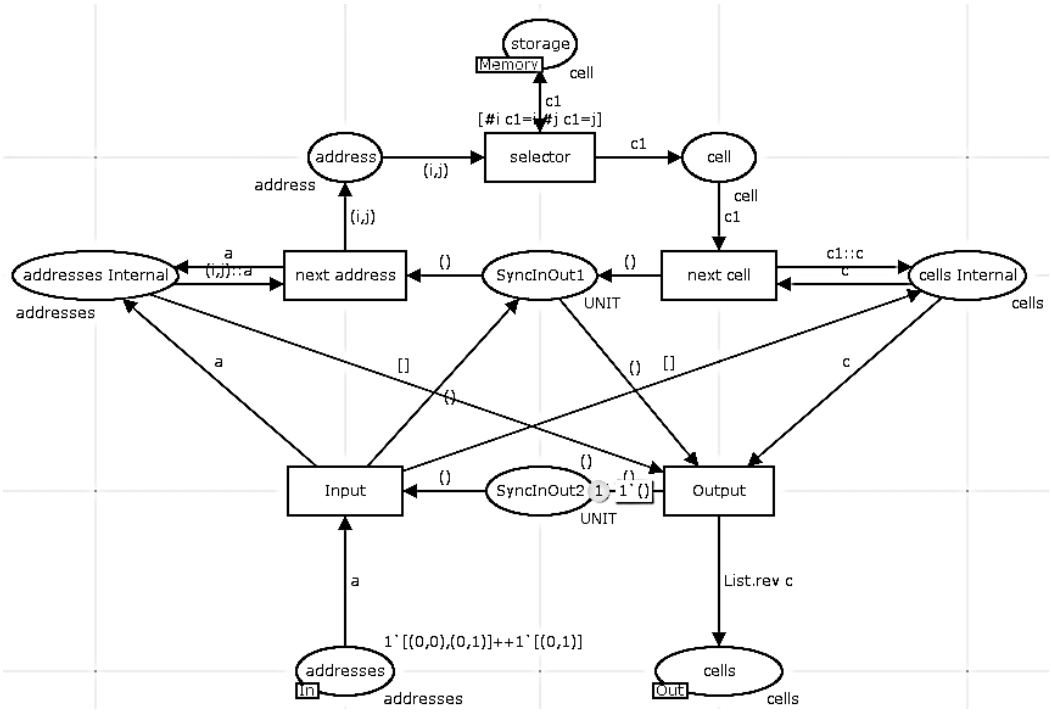

Рис. 15

Мережа Update (рис. 16) виконує зміну комірок пам'яті за аналогічним принципом. У вхідному місці формується множина комірок на оновлення. Мережа в процесі роботи вибирає з неї по одній комірці та виконує оновлення. Слід зазначити, що роботу із сховищем можна організувати і в паралельний спосіб. Для цього в мережі Select слід ускладнити перехід selector, подаючи йому на вхід список адресів. Він, у свою чергу, має зробити перевірку наявності цих адресів у сховищі і повернути відповідний список комірок. Для операції Update виконуються аналогічні дії для її перетворення у паралельну операцію. Як і в попередньому випадку Update є реалізацією синхронного оновлення сховища.

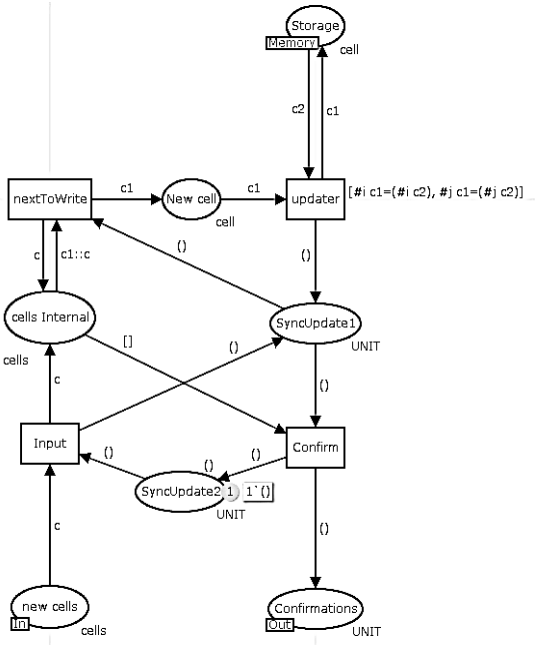

Рис. 16

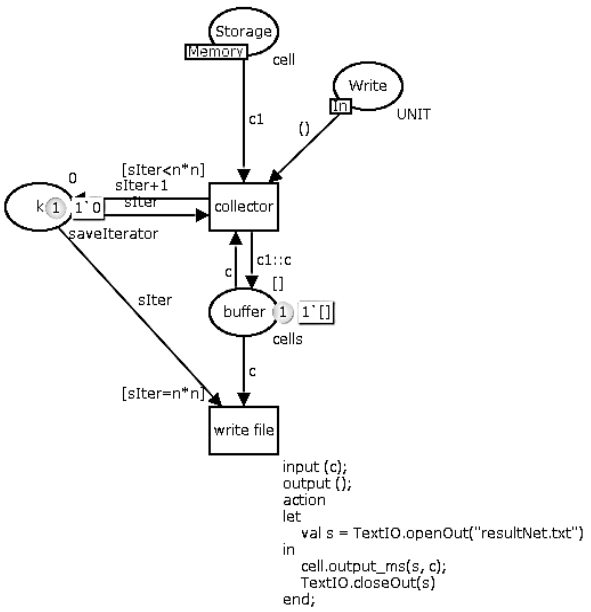

Рис. 17

Виникає питання: для яких комп'ютерних архітектур справедлива багатопотокова мережа Петрі представлення алгоритму?

Вимоги до систем SMP-архітектури такі: всі потоки мають можливість доступу до усієї пам'яті системи і розглянуті мережі задовольняють цим вимогам. Для MPP-архітектури кожен потік володіє своїм сховищем даних, що виявляється розподіленим за всією системою. Тому наявні операції зі сховищем не є придатними. Але оскільки в ієрархічній схемі дані операції є узагальненими, то схема виявляється справедливою для цієї архітектури. Проте необхідна нова конкретизація мереж Update та Select. У цьому випадку змінюється тип даних місця Storage. Кожна мітка в даному місці є коміркою пам'яті, що несе інформацію як про свою адресу, так і про дані. Тому, переходячи до розподіленої системи формат адреси змінюється. Для MPP-архітектур він включатиме в себе номер підсистеми (тобто номер сховища), якому належить комірка. Отримана мережа буде більш загальною, оскільки для SMP-архітектури цей номер буде однаковим для усіх потоків. Крім того, для відображення нерівноправного доступу потоків до певного сховища в розподіленій системі (локальний потік має прямий доступ до сховища, інші – через певний інтерфейс та мережу), схема Select та Update мають включати додаткову перевірку потоку на належність його номеру та даних, що запитуються одній підсистемі.

Подібна ситуація виникає і для GPGPU-систем. У цьому випадку змін зазнає підмережа Initialize. Основний потік, що помічений вхідним маркером мережі є потоком CPU. Він ініціалізує дані сховища, що фактично є даними в глобальній пам'яті GPU. Створювані потоки мають вже номер, що складається із трьох компонент – номера групи потоків, та трьох компонент – номера потоку в групі. В системі такої архітектури доступ до пам'яті всіх потоків є спільним. Тому зазначені схеми операцій зі сховищем справедливі в цьому випадку. Побудована мережа описує випадок, коли всі GPU потоки виконують цикл по всім трьом індексам. Це не є оптимальним рішенням для даної архітектури. Крім того бар'єрна синхронізація всіх потоків у всіх групах може призвести до тупикових ситуацій (deadlocks). Тобто багатопотокова мережа Петрі не відображає особливостей архітектури GPGPU і тому вимагає подальшої конкретизації.

Висновки

Запропоновано використання апарату мереж Петрі для проектування паралельних застосувань суперкомп'ютерних систем різної архітектури (SMP, MPP, архітектури технології GPGPU: Fermi, Tesla, Kepler).

Створено сукупність CAA-M схем, що відповідають найбільш поширеним workflow-шаблонам. Це надає змогу використовувати обидва підходи з метою перетворення представлень алгоритму для цих підходів і використання їх особливостей та переваг (проведення моделювання чи набору формальних трансформацій).

З використанням розглянутих шаблонів побудовано мережу Петрі алгоритму Флойда–Уоршала для набору паралельних потоків (рис. 13). Отримана ієрархічна мережа є узагальненою, оскільки не містить в

собі особливостей архітектури системи на які мають виконуватись паралельні потоки. З одного боку це дає змогу застосувати отримані мережі для різних архітектур, а з іншого, частина мереж потребує подальшої конкретизації, що дозволяє створити представлення застосування чи алгоритму у термінах відповідної цільової архітектури. Наведено особливості модифікації мережі для різних архітектур.

1. *Таненбаум Э.* Архитектура комп'ютера / 5-е изд. – СПб.: Питер, 2007. Р. 634–667.
2. *David B. Kirk, Wen-mei Hwu*. "Programming Massively Parallel Processors: A Hands-on Approach". Published by Elsevier corp.
3. http://www.workflowpatterns.com/patterns/
4. *Russell N., ter Hofstede A.H.M., van der Aalst W.M.P., Mulyar N.* Workflow Control-Flow Patterns : A Revised View. BPM Center Report BPM-06-22, BPMcenter.org, 2006.
5. *Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К.* Многоуровневое структурное проектирование программ. – М.: Финансы и статистика, 1989. – 192 с.
6. *Погорілий С.Д., Мар'яновський В.А., Бойко Ю.В., Вітель Д.Ю.* Формування узагальнених паралельних схем алгоритму Флойда – Уоршала // Системні дослідження та інформаційні технології. – 2010. – № 1. – С. 52–69.
7. *Погорілий С.Д., Трибрат М.І., Вітель Д.Ю.* Дослідження паралельних версій алгоритму Флойда – Уоршала для SMP- та MPP-архітектур. // Математичні машини і системи. – 2011. – № 4. – С. 20–30.
8. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы, построение и анализ; пер. з англ. Красикова И.В., Ореховой Н.А., Романова В.Н.; под ред. Красикова И.В. – М.: Издательский дом «Вильямс», 2005. – С. 719–725.

Одержано 15.03.2012

Про авторів:

Погорілий Сергій Демьянович,
доктор технічних наук,
професор,

Вітель Дмитро Юрійович,
аспірант першого року навчання.

Місце роботи авторів:

Київський національний університет
імені Тараса Шевченка,
01601, м. Київ,
вул. Володимирська, 60.
Тел.: (093) 080 7975.
E-mail: sdp@univ.net.ua,
viteldmitry@gmail.com