УДК 537.322.2
ДОІ 10.31471/1993-9981-2023-1(50)-20-30

ДІАГНОСТУВАННЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ ІЗ ЗАГАЛЬНОЮ ПАМ'ЯТТЮ ЗА ДОПОМОГОЮ МЕРЕЖ ПЕТРІ

О. Г. Малько, А. О. Малько
Івано-Франківський національний технічний університет нафти і газу,
вул. Карпатська, 15, м. Івано-Франківськ, Україна, 76019, E-mail:malko.pochta@gmail.com

З розробкою швидкодіючих ЕОМ застосовність і корисність моделювання значно зросли. Подання системи математичною моделлю, перетворення цієї моделі в команди для ЕОМ і виконання програми уможливили моделювання більших і складніших систем, ніж раніше. Це призвело, в результаті, до значних досліджень методів моделювання на ЕОМ та самих ЕОМ, оскільки вони беруть участь у моделюванні у двох ролях: як обчислювальні засоби та як об'єкт моделювання. Одним із найпоширеніших сучасних методів формалізації моделювання та аналізу обчислювальних систем є мережі Петрі. Сьогодні існує велика кількість як теоретичних досліджень у цій галузі, так і реалізацій практичних інструментаріїв для розробки розподілених алгоритмів, заснованих на мережах Петрі. Розробляється міжнародний стандарт мереж Петрі. Разом із тим, відчувається необхідність розвитку формалізму для більш адекватного та зручного представлення систем зі складною структурою. Сучасні системи часто є мультиагентними і мають ієрархічну, багаторівневу структуру. У зв'язку з цим останнім часом проводяться дослідження щодо розширення формалізму мереж Петрі за рахунок ідей об'єктно-орієнтованого підходу з метою отримання моделей, що явно відображають ієрархічну та мультиагентну структуру системи. Класичне послідовне програмування не може забезпечити створення програм, які ефективно використовують ресурси сучасних обчислювальних систем. Багатопотокові програми, що працюють з даними у спільній пам'яті, часто недостатньо ефективні, оскільки надмірно використовують примітиви синхронізації. При недостатньому використанні примітивів є ризик отримати програму, яка має проблеми із синхронізацією. Засоби динамічного аналізу паралельних програм не завжди можуть вирішити ці проблеми. Для низки програм потрібен статичний аналіз паралельних алгоритмів. Найбільш складною проблемою є діагностування можливості тупиків. У даній роботі розглядаються аспекти, пов'язані із застосуванням мереж Петрі для виявлення проблем синхронізації у програмах з спільною пам'яттю. Наводяться схеми перетворення основних примітивів синхронізації у модель мережі Петрі, і зведення розв'язання задач про відсутність тупиків у мережі Петрі до задачі математичного програмування.

Ключові слова: паралельне програмування, мережі Петрі, примітиви синхронізації, тупики, математичне програмування.

With the development of high-speed computers, the using and usefulness of simulations has increased significantly. Representing a system as a mathematical model, converting this model into commands for a computer, and executing a program on a computer made it possible to simulate larger and more complex systems than before. This has resulted in considerable research into computer simulation methods and computers themselves, as they participate in simulation in two roles: as computational tools and as the object of simulation. Petri nets are one of the most common modern methods of formalization of modeling and analysis of computer systems. Currently, there is a large amount of both theoretical research in this field and the implementation of practical tools for the development of distributed algorithms based on Petri nets. An international standard for Petri nets is being developed. At the same time, there is a need to develop formalism in order to more adequately and conveniently present systems with a complex structure. Modern systems are often multi-agent and have a hierarchical, multi-level structure. In this regard, research has recently been conducted on extending the formalism of Petri nets due to the ideas of an object-oriented approach in order to obtain models that clearly reflect the hierarchical and multi-agent structure of the system. Classical sequential programming no longer leads to the creation of programs that efficiently use the resources of modern computer systems. Multi-threaded programs working with data in shared memory are often inefficient because they overuse synchronization primitives. If you don't use synchronization primitives enough, you run the risk of getting a program that has some kind of synchronization problem. Means of dynamic analysis of parallel programs cannot always solve these problems. A number of applications require static analysis of parallel algorithms. The most difficult problem is diagnosing the possibility of dead ends. This paper considers the aspects related to the use of Petri

nets to detect synchronization problems in parallel programs that use shared memory. Schemes for converting the main primitives of synchronization into a Petri net model are presented, the mechanism of siphons for determining the activity of the Petri net is considered, and even the reduction of solving the problem of the absence of dead ends in the Petri net to the problem of mathematical programming.

Keywords: parallel programming, Petri nets, synchronization primitives, dead ends, mathematical programming.

1. Вступ

Обчислювальні системи є складними, часто великими і включають багато взаємодіючих компонент. Кожна компонента також може бути складною, оскільки взаємодіє з іншими компонентами системи. Це справедливо і для багатьох інших систем. Економічні системи, юридичні системи, системи управління дорожнім рухом, хімічні системи, біологічні системи складаються з багатьох окремих компонент, що взаємодіють один з одним великою кількістю зв’язків.

Отже, незважаючи на різноманітність модельованих систем, виділяється декілька загальних рис, які мають бути представлені в особливостях використовуваної моделі цих систем. Основна ідея полягає в тому, що системи складаються з окремих взаємодіючих компонент. Кожна компонента багатокомпонентної системи сама може бути системою, але її поведінку можна описати незалежно від інших компонент системи, за винятком точно визначених взаємодій з іншими компонентами. Кожна компонента має свій стан. Стан компоненти - це абстракція відповідної інформації, необхідної для опису її (майбутніх) дій. Стан компоненти зазвичай залежить від передісторії цієї компоненти, і може з часом змінюватися. Поняття "стан" дуже важливий при моделюванні компоненти. Наприклад, в моделі системи черг банку можуть бути присутніми декілька касирів і декілька клієнтів. Касири можуть бути вільні (чекаючи клієнта) або зайняті (обслуговуючи клієнта). Аналогічно клієнти можуть бути вільні (чекаючи, коли касир звільниться для обслуговування їх) або зайняті (при обслуговуванні їх касиром). У клінічній моделі стан пацієнта може бути критичним, серйозним, задовільним, хорошим або чудовим.

Діям компонент системи властиві сумісність або паралелізм. Дії однієї компоненти системи можуть виконуватися одночасно з діями інших компонент. У обчислювальній системі, наприклад, під управлінням ЕОМ можуть паралельно діяти багато периферійних пристроїв. В економічній системі одночасно виробники поставляють одну продукцію, тоді як продавці збувають іншу, а покупці використовують третю.

Поєднання природи дій в системі створює певні труднощі при моделюванні. Оскільки компоненти системи взаємодіють, потрібна синхронізація. Передавання інформації від однієї компоненти до іншої вимагає, щоб дії включених в обмін компонент були під час взаємодії синхронізовані. Це може привести до того, що одна компонента чекатиме іншу компоненту. Узгодження в часі дій різних компонент може бути дуже складним, а взаємодії між компонентами, що виходять в результат, важко описувати.

Класичне послідовне програмування вже не забезпечує створення програм, які ефективно використовують ресурси сучасних обчислювальних систем. Багатопотокові програми, що працюють з даними з спільною пам'яттю, часто недостатньо ефективні, оскільки надмірно використовують примітиви синхронізації. При недостатньому використанні примітивів синхронізації є ризик отримати програму, яка має проблеми із синхронізацією. Засоби динамічного аналізу паралельних програм не завжди можуть вирішити ці проблеми. Для низки програм потрібен статичний аналіз паралельних

алгоритмів. Найбільш складною проблемою є діагностування можливих тупиків.

Метою даної роботи є розробка методики діагностування тупиків шляхом відображення алгоритму у еквівалентну мережу Петрі з подальшим її аналізом.

Аналіз сучасних закордонних і вітчизняних досліджень і публікацій блеми.

Основні положення використання мереж Петрі для моделювання причиново наслідкових зв'язків у системах представлені у роботах [1], [2].

У роботі [3] досліджується застосування мереж Петрі для моделювання обчислювальних систем з пам'яттю, що розділяється. Досить багато уваги приділяється розширеним мережам Петрі шляхом введення дуг, що забороняють, переходам з ненульовим часом. Пропонується так звана система SOM, заснована мовою ADA для моделювання та верифікації паралельних процесів.

У роботі [4] пропонується оригінальний математичний апарат для ефективного виявлення відсутності тупиків у мережах Петрі. Використовується механізм сифонів та пасток.

Ряд робіт присвячено моделюванню багатопотокових додатків з використанням мереж Петрі. Це, наприклад, робота [5], де розглядається оптимізація багатопоточних додатків на багатопроцесорних обчислювальних системах (використовуються мережі Петрі з ненульовим часом переходу).

У роботі [6] розглядається верифікація паралельних програм із використанням мереж Петрі як графа доказів.

Робота [7] розглядає підмножину мереж Петрі — СТ-мережі (Casual–Time nets) — для моделювання автоматів паралельного виконання та пропонує засоби для побудови паралельних додатків та контрольованим потоком виконання.

Робота [8] використовує механізм інваріантів моделі мережі Петрі для визначення помилок синхронізації в програмах, а саме: тупиків, нескінченних внутрішніх циклів і втрати повідомлень при передчасному завершенні. У [9] також пропонується використати механізм інваріантів.

Роботи [10] і [11] пропонують моделі мереж Петрі для основних примітивів синхронізації, але не пропонують методів доказу відсутності помилок синхронізації в програмах, що моделюються.

Висвітлення невирішених раніше частин загальної проблеми.

На теперішній час значний інтерес викликають засоби моделювання та аналізу складних паралельних та розподілених систем. Такими системами є, наприклад, обчислювальні машини та комплекси з паралельною та розподіленою архітектурою, паралельні програми та алгоритми, протоколи взаємодії (комунікаційні, верифікуючі), моделі обчислювальних процесів. При дослідженні обчислювальних системи нас можуть цікавити її різні властивості. Математичні методи у багатьох випадках дозволяють отримати однозначну відповідь на такі питання. Тут велике значення має початковий вибір використовуваного формалізму, тобто способу моделювання реальної системи. Для вирішення завдань аналізу та верифікації в теорії паралельних та розподілених обчислень на даний час пропонуються різні способи моделювання реальних систем. До найбільш відомих формалізмів можна віднести кінцеві автомати, алгебри процесів, CCS Р.Мілнера, мови трас, а також різні їх модифікації, у тому числі з додаванням конструкцій часу та ймовірності [2]. Різні класи моделей мають різні властивості виразності та алгоритмічної розв'язуваності. Причому ці два параметри моделі, як правило, суперечать один одному: обраний формалізм може виявитися або дуже виразним і не піддаватися аналізу, або дозволяти ефективно вирішувати всі

необхідні проблеми, але при цьому бути занадто слабким, і недостатньо повно описувати модельовану систему. Для формалізації процесів у обчислювальних системах необхідно звернутися до абстракцій умов, подій, причинно-наслідкових зв'язків тощо. Якщо ж цим абстракціям поставити у відповідність певні графічні примітиви і пов'язати їх лініями, що несуть певну логіку, то вийде якась мережа, тобто. графічний образ процесу. Переагою мережевих методів опису та аналізу процесів є те, що абстракції, що використовуються в них, близькі до інтуїтивних уявлень про процеси. Мережі Петрі - математичний апарат для моделювання динамічних дискретних систем[8]. Вперше описаний Карлом Петрі. Популярність мереж Петрі викликана вдалим уявленням різних типів об'єктів, присутніх у багатьох системах, що моделюються, і «подійним» підходом до моделювання. Вони мають найкращі можливості для опису взаємозв'язків і взаємодій паралельно працюючих процесів. Мережі Петрі є потужним інструментом дослідження систем, що моделюються, завдяки їх можливості опису багатьох класів дискретних, асинхронних, паралельних, розподілених, недетермінованих систем, завдяки наочності представлення їх роботи, розвиненому математичному та програмному апарату аналізу. Мережі Петрі розроблялися спеціально для моделювання тих систем, що містять взаємодіючі паралельні компоненти, наприклад, апаратне та програмне забезпечення ЕОМ. Існують певні області, де мережі Петрі є ідеальним інструментом для моделювання. Це області, у яких події відбуваються синхронно і незалежно. Однією з таких областей є апаратне та програмне забезпечення ЕОМ.

Висвітлення основного матеріалу дослідження

Багато-потокові програми, що працюють з даними у спільній пам'яті, часто недостатньо ефективні, оскільки надмірно використовують примітиви синхронізації. При недостатньому використанні примітивів синхронізації є ризик отримати програму, яка має в якомусь вигляді проблеми із синхронізацією. Засоби динамічного аналізу паралельних програм не завжди можуть вирішити ці проблеми.

Для ряду програм потрібен статичний аналіз паралельних алгоритмів. Найбільш складною проблемою є діагностування можливості тупиків. ***Метою даної роботи є вирішення задачі діагностування тупиків у алгоритмах шляхом застосування перетворення алгоритму в еквівалентну мережу Петрі з подальшим її аналізом.***

Структура мережі Петрі визначається її позиціями, переходами, вхідними та вихідними функціями (відображеннями) [1], тобто кортежем множин $C = (P, T, F, I, O, \mu_0)$. Для структурного опису математичної моделі мережею Петрі застосовуються такі позначення і відповідності.

Звичайна мережа Петрі, де:

$P = \{p_1, p_2, \ldots, p_n\}$ - множина позицій (умов);

$T = \{t_1, t_2, \ldots, t_m\}$ - множина переходів (подій);

$\mu : P \to N$ - відображення множини позицій у множину невід'ємних цілих чисел;

$I : T \to P$ - вхідна функція переходів - відображення переходів (подій) у комплект їхніх вхідних позицій (передумови);

$O : T \to P$ - вихідна функція переходів - відображення переходів (подій) у комплект їхніх вихідних позицій (післяумов);

$I : P \to T$ - вхідна функція позицій - відображення позицій (умов) у комплекти їхніх вхідних переходів (перед подій);

$O : P \to T$ - вихідна функція переходу - відображення позицій у комплекти їхніх вихідних переходів (післяподій).

$I(p_i)$ - множина вхідних переходів позиції p_i (передподії умові p_i);

$O(p_i)$ - множина вихідних переходів позиції p_i (післяподії умови p_i);

$I(t_i)$ - множина вхідних позиції переходу t_i (передумови події t_i);

$O(t_i)$ - множина вихідних позиції переходу t_i (післяумови події t_i);

$\#(p_i, I(t_j))$ - кратність вхідної позиції p_i для переходу t_j є число появ позиції у вхідному комплекті переходу;

$\#(p_i, O(t_j))$ - кратність вихідної позиції p_i для переходу t_j є число появ позиції у вихідному комплекті переходу.

Граф мережі Петрі відповідає її структурі і може бути представлений кортежем $G = (P, T, F, \mu)$, де $F \subseteq (P \times T) \cup (T \times P)$ - множина спрямованих дуг $P \cap T = \varnothing$. В графах мереж Петрі використовують такі позначення: коло ○ є позицією, а планка | – переходом.

Початковий стан мережі Петрі задається за допомогою маркування її позицій $\mu_0 : P \to N$, яке полягає у наданню кожній позиції мережі деякого числа маркерів. Присутність маркера у позиції трактується як виконання деякої умови. Спрацювання переходів змінює розмітку сітки у відповідності до правил спрацювання переходів. Отже, розмітку μ_0 можна представити вектором з $|P|$ пронумерованих елементів, у якому *i*-тий елемент дорівнює кількості маркерів у *i*-тій позиції. У загальному випадку кількість маркерів у вузлі може бути більшою за одиницю.

Перехід дозволений якщо кількість маркерів у його вхідних позиціях не менше кількості дуг які йдуть від кожної вхідної позиції до переходу.

Нове маркування μ' здійснюється шляхом видалення маркерів з вхідних позицій переходу з наступним розміщення маркерів у кожній вихідній позиції (по одному маркеру на кожну дугу). Це можна представити як функцію наступного стану $\mu' = \delta(\mu, t_j)$ з маркуванням μ і переходом $t_j \in T$, який можливий тоді і тільки тоді, коли $\mu(p_i) \geq \#(p_i, I(t_j))$ для всіх $p_i \in I(t_j)$. Тоді якщо $\delta(\mu, t_j)$ визначена, то $\mu' = \delta(\mu, t_j)$ де $\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$ для всіх $p_i \in I(t_j)$.

Розширення запуску позначається як $\mu'(\sigma)$, де σ - послідовність переходів, що переводять μ у μ''. Множина усіх маркувань, досяжних з μ_0, позначається як $R(\mu_0)$.

Мережа Петрі також може задаватися за допомогою матриці інцидентності $C = [c_{ij}]$ для якої

$$c_{ij} = \begin{cases} 1, \text{ якщо } t_j \in I(p_i) \cap O(p_i); \\ -1, \text{ якщо } t_j \in I(p_j) \cap O(p_i) \wedge i \neq j; \\ 0, \text{ у іншому випадку}. \end{cases}$$

Аналіз моделей систем на базі мереж Петрі зводиться до аналізу наступних властивостей [2]**.**

Безпека. Одна з найважливіших властивостей мереж Петрі, яка повинна моделювати реальний пристрій, — безпека. Позиція $p_i \in P$ мережі Петрі $C = (P, T, I, O)$ з початковим маркуванням μ є безпечною, якщо $\mu'(p_i) \leq 1$ для будь-якої $\mu' \in R(C, \mu)$**.** Мережа Петрі безпечна, якщо безпечна кожна її позиція. Якщо інтерпретувати мережі як умови і події, маркування кожної позиції повинна бути безпечною. Безпека дозволяє реалізувати позицію тригером, але в більш загальному випадку можна використовувати лічильник. Проте будь-який апаратно-реалізований лічильник

обмежений за максимальним числом, яке він може представити.

Обмеженість. Обмеженість є узагальненням властивості безпеки. Позиція є k- безпечною або k- обмеженою, якщо кількість маркерів в ній не може перевищувати ціле число k. Зауважимо, що межа k' за числом маркерів, які можуть знаходитися в позиції, може бути функцією від позиції (наприклад, позиція p_i може бути 3-безпечною, тоді як позиція p_j — 8-безпечною).

Збережуваність. При моделювання мережами Петрі систем розподілу ресурсів важливою властивістю є збережуваність . Тут необхідно щоб маркери які представляють ресурси, ніколи не створювались і не знищувались. Простий спосіб зробити це - задати, щоб загальне число маркерів у мережі залишалося сталим.

Мережа Петрі $C=(P,T,I,O)$ з початковою маркуванням μ називається збережною, якщо для всіх $\mu' \in R(C,\mu): \sum_{p_i \in P} \mu'(p_i) = \sum_{p_i \in P} \mu(p_i)$. .

Активність. Іншою властивістю при моделюванні мережами Петрі систем розподілу ресурсів є безвихідь. Перехід називається *активним,* якщо він не заблокований (не безвихідний). Перехід t_j мережі Петрі C називається *потенційно запустимим* у маркуванні μ якщо існує маркування $\mu' \in R(C,\mu)$, у якому перехід t_j дозволений. Перехід *є активним* з маркуванням μ, якщо він може бути потенційно запущеним всякому маркуванні з $R(C,\mu)$. Отже, якщо перехід активний, то завжди можливо перевести мережу Петрі з її поточного маркування в маркування, у якому запуск переходу стане дозволеним.

Виявлення тупиків у багатопотоковому обчислювальному середовищі із спільними ресурсами. Отже, необхідним є визначення однієї характеристики — активності (liveness).

Якщо перехід у мережі Петрі ніколи не може бути запущений, такий стан називається тупиком. Можливість настання стану тупика в мережі Петрі вказує на можливості стану тупика і в модельованому обчислювальному середовищі.

Для аналізу тупиків у мережі з маркуванням застосовується наступна класифікація:

– активність рівня 0 має перехід t_j, який ніколи не може бути запущений;

– активність рівня 1 має перехід t_j, якщо він може бути потенційно запущеним, тобто існує таке допустиме і досяжне маркування $\mu' \subset R(C,\mu)$, у якому цей перехід дозволено;

– активність рівня 2 має перехід t_j якщо для будь-якого цілого n існує послідовність запусків t_j, у якій t_j присутнє принаймні n разів.

– активність рівня 3 має перехід t_j, якщо існує нескінченна послідовність запусків, у якій t_j присуне необмежену кількість разів;

– активність рівня 4 має перехід t_j, якщо для будь-якого $\mu' \subset R(C,\mu)$, існує така послідовність σ запусків, що t_j дозволений у $\delta(\mu',\sigma)$.

Перехід, що володіє активністю рівня 0, називається пасивним. Перехід, що володіє активністю рівня 4, називається активним. Мережа Петрі володіє активністю рівня i, якщо кожен її перехід володіє активністю рівня i.

Наявність у мережі переходів з активністю 0 з впевненістю підтверджує наявність тупиків у модельованій системі.

Мережа Петрі не містить тупиків, якщо для будь-якого досяжного маркування є принаймні один дозволений перехід. Позиція p обмежена, якщо існує така

константа k, що $\mu(p) \le k$ для всіх $\mu \in R(\mu_0)$. Мережа Петрі називається обмеженою, якщо всі позиції мережі обмежені.

Підмножина позицій $S \subseteq P$ називається сифоном (syphon), якщо будь-який вхідний перехід в S є також вихідним переходом у S, тобто $O(S) \subseteq I(S)$. Відповідно є пастка S (trap), якщо $I(S) \subseteq O(S)$. Сифон (пастка) називається мінімальним, якщо він не містить інших сифонів (пасток).

Існує кілька основних підходів до аналізу активності мереж Петрі – метод дерева досяжності, метод матричних рівнянь та метод сифонів та пасток. Однак ці методи можуть показати необхідність, але не можуть показати достатність для доказу існування тупиків у моделі. У цій роботі використовується перетворення рівнянь мережі Петрі у рівняння для методу математичного моделювання.

Процес формування коректного алгоритму паралельної обробки та верифікації складається з низки кроків, які можуть бути описані таким чином:

1. Розробка моделі алгоритму.
2. Представлення моделі алгоритму мовою мереж Петрі.
3. Верифікація побудованої моделі Петрі та повернення до першого пункту для уточнення моделі алгоритму.

Моделювання багатопотокової обробки та синхронізації за допомогою мереж Петрі. Для представлення моделі мовою мереж Петрі потрібно відобразити основні конструкції вхідної моделі на вихідній мові. У цій роботі застосовуються примітиви синхронізації двох типів:Mutex та Event.

Примітив типу Mutex є засобом взаємовиключення обчислювальних потоків і включає наступні методи:

– WaitAndLock — очікувати на звільнення з негайним захопленням
– Release – звільнення.

Mutex має атрибут власності, тобто, властивість належності обчислювальному потоку, що його захопив. Цей примітив потребує підтримки операційною системою, може призвести до контекстного перемикання потоків і не переводить обчислювальне ядро у стан активного очікування. Примітив Mutex можна подати у вигляді моделі мережі Петрі (рис.1).

Наявність маркера у позиції p_1 показує, що Mutex знаходиться у вільному стані. Ця позиція повинна використовуватися кількома обчислювальними потоками. З появою маркера в позиції p_0 та наявності маркера в позиції p_1 стає можливим перехід t_0, маркер у позиції p_1 тепер відсутній, що унеможливлює здійснення переходів, які залежать від наявності маркера в позиції p_1.

Після захоплення маркера обчислювальний потік виконує код критичної секції (позиція p_2), виконується перехід t_1, який передає маркер у позицію p_1, уможливлюючи здійснення операцій, пов'язаних з наявністю маркера у цій позиції переходів.

Примітив типу Event використовується для встановлення факту виникнення будь-якої події, може перебувати у двох станах — сигнальному та несигнальному і включає наступні методи:

– Set - встановити об'єкт у сигнальний стан;
– Reset – встановити об'єкт у несигнальний стан;
– Wait – очікувати настання сигнального стану.

На відміну від об'єктів типу Mutex, об'єкти типу Event не мають атрибуту приналежності. Тут розглядається варіант об'єктів типу Event, що потребує ручного переведення стану в несигнальний після того, як рівно один з числових потоків отримав доступ до нього.

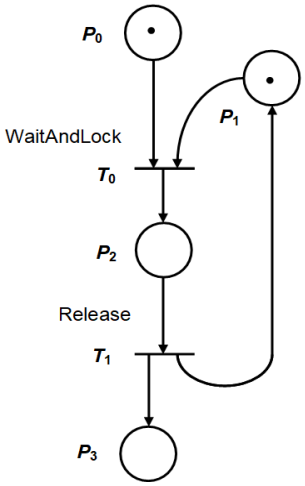

Рисунок 1 – Схема мережі Петрі примітиву Mutex

Примітив Event поряд з примітивом Mutex вимагає підтримки операційною системою, що може призвести до перемикання контексту обчислювальних потоків і не переводить обчислювальне ядро в стан активного очікування.

Тут враховується наступний факт: якщо примітив Event знаходиться у сигнальному стані, то наступні виклики методів Set не чинять на цей стан впливу. Якщо примітив Event знаходиться в несигнальному стані, то наступні виклики методу Reset не впливають на цей стан. Дані умови призводять до більш точної та більш складної моделі, ніж у роботах [10] та [11].

Мережа Петрі для примітиву Event представлена на рис. 2. Позиція p_0 є входом методу Set. Якщо примітив Event вже знаходиться у стані «сигнал» (що визначається наявністю маркера в позиції p_1), то операція розміщення маркера у позицію p_0 буде еквівалентна порожній операції. Якщо ж цей примітив не перебуває в стані сигналу, то операція розміщення маркера у позицію p_0 переводить примітив у стан сигнал.

Позиція p_5 являє собою вхід методу Reset. Розміщення маркера у даній позиції переводить примітив Event в стан «сигнал не встановлений» незалежно від первісного стану примітиву.

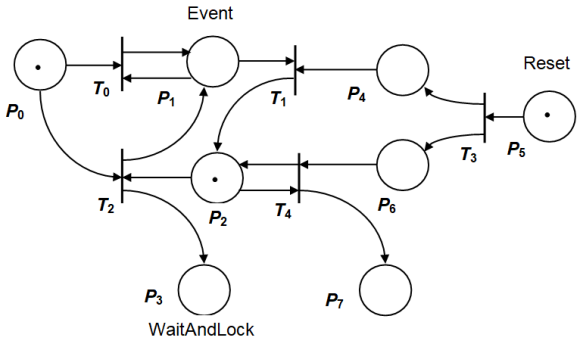

Рисунок 2 – Схема мережі Петрі примітиву Event

Виходом методу Reset є позиція p_7 - поява маркера у даній позиції сигналізує про завершення роботи методу.

Якщо у позиції p_3 вже є маркер, то метод Wait завершується успіхом.

Сукупна мережа Петрі обчислювальної моделі може формуватися як вручну, так і автоматизованим способом. Автоматизований спосіб передбачає, або повний аналіз тексту на вихідній мові програмування, що можливо, але технічно трудомістко, або синтез мережі Петрі за описом на спрощеній мові, що містить тільки конструкції, пов'язані з керуванням обчислювальними потоками та механізмами синхронізації. Синтаксичний та семантичний аналіз подібної мови можна реалізувати за допомогою загальнодоступних засобів побудови компіляторів.

Після синтезу сукупної моделі мережею Петрі необхідно визначити, чи можливі у ній тупики. Є три основні підходи до виявлення даної властивості:

- побудова дерева досяжності,
- метод матричних рівнянь
- метод сифонів та пасток.

Перші два методи достатньо досліджені, тому тут розглядається метод сифонів та пасток [1], [2].

Введемо дві функції від сифона S:

$$f(S) = \min\{\ \mu(S) \mid \mu \in R(p_0)\}\,, \qquad (1)$$

де $\mu(S)$ загальна кількість маркерів у S, тобто $\mu(S) = \sum_{p \in S} \mu(p)$;

$$f(S) = \min\{ S \mid \mu = \in \mu_0 + CY, \mu \geq 0, Y \geq 0\}, \quad (2)$$

де C - матриця інцендентності мережі Петрі, μ - вектор маркування мережі.

Вираз $\mu = \in \mu_0 + CY$ і є *рівнянням стану мережі*. З базової теорії мереж Петрі випливає, що будь-яке досяжне маркування задовольняє рівняння стану, але зворотне невірно.

Сифон S є потенційним тупиком тоді і тільки тоді, коли $f(S) = 0$. Під цим розуміється, що $F(S) \geq f(S)$. Отже, будь-який сифон S, такий, що $F(S) > 0$ не є потенційним тупиком. Таким чином, мережа Петрі не містить тупиків, якщо кожен сифон S цієї мережі або містить марковану пастку, або $F(S) > 0$.

Для перевірки відсутності тупиків мережі Петрі з використанням даної властивості проблема повинна бути вирішена для кожного мінімального сифона, що не містить маркованої пастки. Цього можна уникнути, перетворюючи ці проблеми на еквівалентне завдання квадратичного програмування, що випливає з наступного.

Нехай $\{S_1, S_2, \ldots, S_n\}$ є множина мінімальних сифонів, які не містять маркованих пасток. Мережа Петрі не містить тупиків, якщо $F>0$, де

$$F = \min\{ \sum_{i=1}^{n} z_i \mu(S_i) \mid \mu = \mu_0 + CY, \; , \sum_{i=1}^{n} z_i = 1, \mu \geq 0 Y \geq 0, z_i \geq 0\}$$

Щоб уникнути перерахування, краще використовувати вираз $F(S)$, що визначається наступним чином:

$$F(S) = \min\left\{\mu(S_i) \mid \mu = \mu_0 + CY, \mu \geq 0 Y \geq 0\right\},$$

де C матриця інцендентності мережі Петрі,

Будь-який сифон, такий, що $F(S) > 0$, не є потенційним тупиком, а μ і Y чисельні вектори. Вираз $\mu = \mu_0 + CY$ суть рівняння стану мережі. Любий сифон S такий, що $F(S) > 0$ не є потенційним тупиком.

Ця проблема зводиться до задачи лінійного програмування. Використання лінійного програмування вимагає перевірки всіх мінімальних сифонів, і ефективність цього методу залежить від їх кількості. Відомо [12], що загальна кількість мінімальних сифонів (пасток) швидко виходить за межі практичного застосування і що, у гіршому випадку, це експоненційне зростання.

У цьому випадку замість використання поліноміальних за обчислювальною складністю алгоритмів розв'язувалося задача математичного програмування. Методи розв'язування задач математичного програмування викладено у [13].

Даний метод був застосований для визначення відсутності тупиків у моделі статичного пулу обчислювальних потоків, що застосовується для реалізації модифікованого захищеного середовища [14]. За алгоритмом, представленим на внутрішній мові опису моделі, була синтезована мережа Петрі, зображена на рис. 3, за якою було сформовано завдання математичного програмування. Вирішення цього завдання показало, що умови відсутності тупиків задовольняються і, отже, мережа Петрі не містить тупиків.

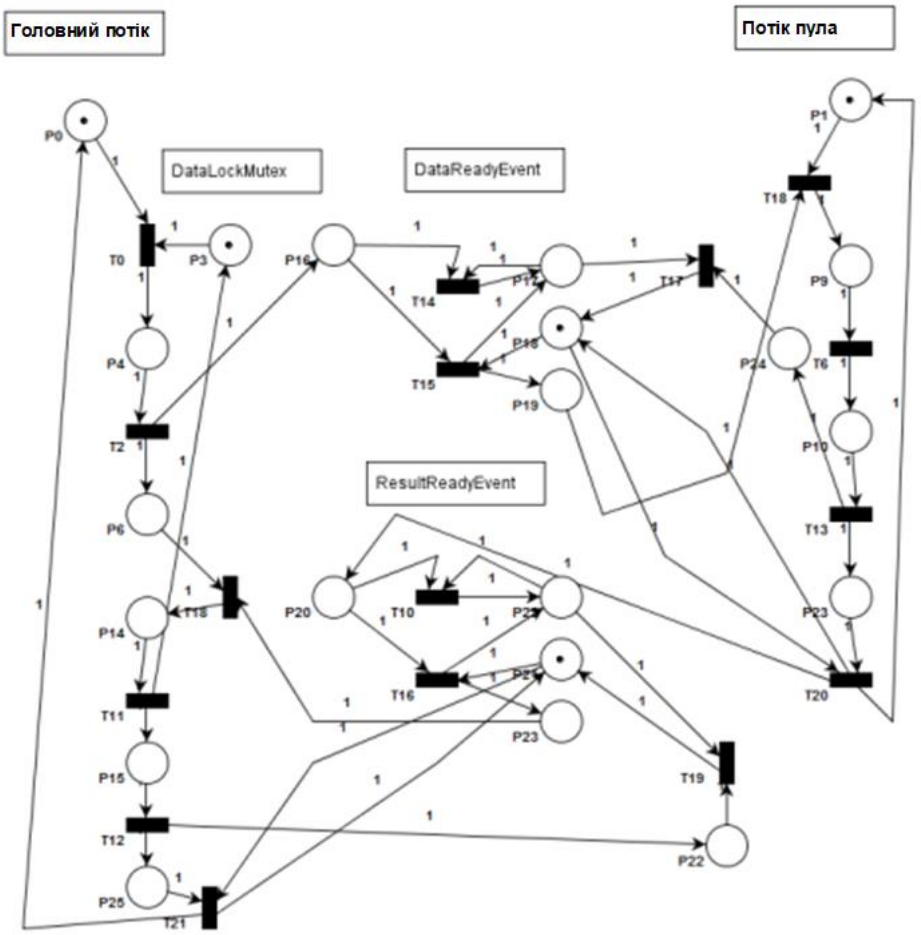

Рисунок 3 – Синтезована мережа Петрі

Висновки

У роботі розглянуто можливість застосування алгоритму діагностування обчислювальних потоків шляхом використання механізму моделювання мереж Петрі.

За мовою опису моделі була синтезована еквівалентна мережа Петрі. Аналіз еквівалентної мережі Петрі методом математичного програмування дозволив встановити, що ця мережа, а отже й оригінальний алгоритм не має тупиків.

Дуже цікавим та перспективним завданням є автоматизація побудови мереж Петрі за заданими специфікаціями обчислювальних потоків та розширення числа моделей синхронізуючих примітивів.

Другим цікавим завданням може бути автоматична побудова найефективнішої моделі багатопоточних обчислень з використанням найменшої кількості примітивів синхронізації.

Список використаних джерел

1. Peterson J. Petri net theory and the modeling of systems, Prentice Hall; 1St Edition, 1981, 290 p.

2. Murata T. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE. 1989. Vol. 77, No 4. P. 109-118.

3. Vallejo F., Gregorio J, Gonzalez Harbour M., Drake J. Shared Memory Multiprocessor operating System with Extended Petri Net Model *IEEE transactions on parallel and distributing systems*. 1994. Vol. 5, No 7. P. 88-94.

4. Feng Chu, Xiao-lan Xie D. Analysis of Petri Nets За допомогою Siphons і Mathematical

Programming. *IEEE Transactions of Robotics and Automation.* 1997. Vol. 13, No. 6, P. 105-112.

5. Govindarajan F., Suciu W. Timed Petri NetModels з Multithreaded Multiprocessor Architectures, IEEE Preceedings if the 7-th International Workshop на Petri Nets and Performance Models. 1998 Vol. 12, No. 6, P. 125-130.

6. Takaoka T. A Systematic Approach Parallel Verification, Department of Computer Science of University of Ibaraki. 1995. Vol. 10, No. 6, P. 125-130.

7. Pommereua F. Petri Nets Executable Specifications of High-Level Timed Parallel Systems. *Science of University of Ibaraki*. 2005. Vol. 7, No. 5, P. 25-30.

8. Bruce P. Detection of Control Flow Errors in parallel Programs at Compile Time. *International Journal of Distributed and parallel Systems* (IJDPS). 2010. Vol. 1, N 2, 115-123.

9. Padidar S. Parallel Program verification: A Brief Introduction. *International Journal of Parallel Programming* 2010.Vol. 30, No 5. P. 1-23,

10. Kavi K, Moshtaghi A., Deng-Jyi C. Modeling Multithreaded application using Petri nets. *International Journal of Parallel Programming*. 2002. Vol. 30, No 5. P. 1-23,

11. Kavi K, Bukhles P., Bhat U. Isomorphism Between Petri net and Dataflow Graphs. *IEEE Transactions on Software Engineering*. 1987. Vol. 13, № 10. P. 36-42.

12. Stetsenko V., Dorosh V.., Dyfuchyn Anton Petri-object simulation: sofyware package and complexity. Intelligent Data Acquisition and Advanced Computing Sysytems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference. IEEE, 2015. Vol.1. N 10. P. 56-44.

13. Minoux M. Programmation Mathematique: Theorie and Algorithms. - Dunod, Paris, France, 1983. 313 p.

14. Dietsch D., Heizmann M., Klumpp D., Naouar M., Podelski A.. Verification, Model Checking, and Abstract Interpretation: 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17–19, 2021. P. 112-120.