*DOI 10.25140/2411-5363-2023-3(33)-130-140*
*УДК 004.41:004.94*

***Олександра Дифучина***
аспірант кафедри інформаційних систем та технологій
Національний технічний університет України «Київський політехнічний інститут
ім. Ігоря Сікорського» (Київ, Україна)
**E-mail:** dyfuchyna@gmail.com **ORCID:** https://orcid.org/0000-0002-5477-4533 **SCOPUS Author:** 57202215305

# МЕТОД ОПТИМІЗАЦІЇ ПАРАМЕТРІВ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

*На швидкодію паралельних алгоритмів сильно впливають параметри, що визначають характеристики підзадач та механізми їх взаємодії, а також обчислювальні ресурси, які використовуються для виконання програми. Тестування паралельних алгоритмів в реальних умовах є ресурсовитратним, з огляду на це у роботі пропонується метод оптимізації параметрів паралельних обчислень на основі Петрі-об'єктного моделювання та еволюційного алгоритму. У якості прикладу застосування методу побудована та досліджена модель паралельного алгоритму імітації дискретно-подійної системи.*

***Ключові слова:*** *паралельні обчислення, імітаційне моделювання, мережа Петрі, еволюційний алгоритм.*
*Табл.: 1. Рис.: 3. Бібл.: 14.*

**Актуальність теми дослідження.** Сучасні інформаційні технології потребують швидкої роботи алгоритмів, яку можна досягти за допомогою паралельних обчислень. Проте, в залежності від параметрів, що визначають характеристики підзадач та механізми їх взаємодії, використання паралельних обчислень може призвести як до прискорення, так і до сповільнення обчислень. На практиці проблему налаштування параметрів паралельних обчислень вирішують шляхом багатократного тестування програми на різних комп'ютерних платформах при різних наборах параметрів, щоб гарантувати коректність та ефективність обчислень. Очевидно, що даний метод налаштування є ресурсозатратним та не гарантує точний результат. На жаль, існуючі засоби проєктування та розробки програм ефективні для налагодження та оптимізації часу виконання лише однопотокових програм. З огляду на це, створення методу, спрямованого на вдосконалення процесу налагодження багатопотокових програм та підвищення ефективності використання паралельних обчислень в інформаційній технології, є актуальним науковим завданням.

**Постановка проблеми.** Математичні методи оцінювання ефективності паралельних обчислень спроможні вказати лише на існування обмеження на максимально досяжне прискорення за ідеальних умов вільного доступу до обчислювального ресурсу та відсутності синхронізації обчислень (тобто фактично відсутності взаємодії між частинами програми).

Існуючі засоби проєктування програм, такі як UML, дають змогу (досить узагальнено) представити графічно взаємодію окремих частин програми, проте не надають можливості будь-якого чисельного аналізу обчислень. Засоби моделювання, такі як високорівневі мережі Петрі, рекомендовані міжнародним стандартом ISO/IEC 15909-1:2019 [1] як техніка для специфікації паралельних і розподілених систем. Не зважаючи на те, що існує досвід розробки симуляторів обчислень на основі мереж Петрі, жоден з них не став на сьогоднішній день широко використовуваним у розробці паралельних обчислень.

Таким чином, наразі не існує уніфікованого методу створення моделі паралельних обчислень і, відповідно, не існує іншого, окрім реальної програми, засобу, який можна використовувати для оптимізації параметрів паралельної програми. Відсутність засобів моделювання стримує розробку високоефективних паралельних обчислень.

**Аналіз останніх досліджень і публікацій.** На сьогодні засобів моделювання паралельних обчислень відомо небагато, проте рух досліджень у цьому напрямку є. Так, у роботі [2] для моделювання пулу потоків використовуються синхронізовані автомати.

© Олександра Дифучина, 2023

Модель показує значне скорочення тривалості виконання завдань у випадку паралельних потоків.

Використання класичних мереж Петрі для моделювання багатопотокових застосунків викладено в [3]. Розробка інструменту візуалізації багатопотокової java-програми з використанням класичних мереж Петрі представлена в [4]. У роботі [5] описано можливі сценарії виникнення зависання багатопотокової програми та розглянуто моделювання цих сценаріїв класичною мережею Петрі та операційним графом, який запропонований авторами публікації, на прикладі простого застосунку.

У статті [6] розглянута концепція розробки коду програмного забезпечення за його моделлю, описаною мережею Петрі. За цією концепцією модель будується за вимогами, які висуваються до програмного забезпечення. На наступному етапі програмне забезпечення розробляється у відповідності до побудованої моделі та використовує модель для верифікації. Автоматизована генерація коду за моделлю програми у такій концепції гарантує її відповідність вимогам, а мережі Петрі мають бути складовою частиною розробки програмного забезпечення. Проте втілення цієї концепції у подальших роботах не знайдено.

Серед існуючих засобів тестування або налагодження багатопотокових програм майже всі орієнтовані лише на пошук помилок пов'язаних із взаємоблокуванням потоків (deadlock) або помилками неузгодженості даних. Метод статичного виявлення взаємоблокувань запропоновано в [7]. Розроблено спеціальний інструмент для виявлення взаємоблокувань [8]. У роботі [9] для дослідження ситуацій взаємоблокування у багатопотокових програмах автори пропонують спеціальний клас звичайної мережі Петрі під назвою мережа Гадара. У [10] запропоновано розгорнутий метод виявлення помилок неузгодженості даних у багатопотокових програмах з використанням специфічних мереж Петрі з даними (PD-net). У роботі [11] запропоновано фреймворк під назвою PVcon для динамічного виявлення помилок паралелізму. Експериментальні результати показують, що PVcon може ефективно виявляти вдвічі більше помилок у багатопотокових програмах, ніж інші методи. Однак, автори вказують, що у даній роботі вони зосередились на помилках багатопотоковості за виключенням проблем взаємоблокувань.

Тож, засоби тестування програм спрямовані, насамперед, на аналіз коректності виконання обчислень, але не на аналіз ефективності паралельних обчислень.

**Виділення недосліджених раніше частин загальної проблеми.** Проведений аналіз останніх досліджень і публікацій стосовно засобів моделювання паралельних обчислень та їх здатності оцінювати ефективність показав, що достатньо багато з них, використовують мережі Петрі. Проте звичайні мережі Петрі при детальному описі програми призводять до надто складних моделей. Тому моделюванню, як правило, підлягають найпростіші елементи паралельної програми: розділення на підзадачі, очікування завершення виконання підзадачі, блокування. Більш складні інструменти взаємодії між потоками, як wait/notify, розглядаються лише в окремих публікаціях. Майже усі засоби моделювання не враховують обмеженість використовуваного паралельною програмою ресурсу, що, звісно, зменшує точність таких моделей. До того ж, існуючі засоби моделювання недостатньо точно відтворюють механізми захоплення обчислювального ресурсу та механізми взаємодії підзадач паралельної програми, тому існує необхідність розвитку методів та засобів для оцінювання ефективності паралельних обчислень на основі моделей.

**Метою дослідження** є підвищення ефективності використання паралельних обчислень в інформаційних технологіях за рахунок оптимізації параметрів паралельних обчислень на основі моделей, що можуть бути використані для оцінювання часу виконання паралельного алгоритму.

**Виклад основного матеріалу.**

**Метод оптимізації параметрів паралельних обчислень.** Суть методу оптимізації параметрів полягає у визначенні набору параметрів, які впливають на ефективність алгоритму, та варіюванні цих параметрів у моделі для визначення оптимальних значень, що гарантують найвищу ефективність алгоритму. Критерієм оптимізації є мінімізація часу виконання алгоритму:

$$T(X) \to min,$$

де $X$ – множина досліджуваних параметрів паралельного алгоритму,

$T$ – час виконання алгоритму при заданій $X$.

Метод складається з низки етапів.

На *першому* етапі для побудови моделі використовуються шаблони моделювання механізмів багатопотокової програми. Розроблені фрагменти мереж Петрі використовуються для створення Петрі-об'єктів. Далі Петрі-об'єкти доповнюються у відповідності до об'єктів програми. Створюється список Петрі-об'єктів та встановлюються зв'язки між парами Петрі-об'єктів. Конструюється Петрі-об'єктна модель на основі списку Петрі-об'єктів. Результатом даного етапу є розроблений метод getModel(), який повертає Петрі-об'єктну модель багатопотокової програми.

На *другому* етапі залучається метод збору даних для моделі. Проводиться експериментальне дослідження часу витраченого на виконання інструкцій та визначаються їх статистичні характеристики. Результатом даного етапу є параметри часових затримок, які вносяться до моделі: середнє значення, середнє квадратичне відхилення та закон розподілу. Якщо відхилення невелике, то може бути прийняте рішення про детерміновану затримку.

На *третьому* етапі проводяться попередні експерименти з моделлю для визначення набору параметрів, що впливають на ефективність. Такими параметрами можуть бути: кількість потоків, кількість підзадач, розмір підзадачі, обсяг обчислювальних ресурсів тощо. Варіювання цих параметрів треба забезпечити при проведенні експериментів з моделлю. Це реалізується шляхом розміщення параметрів у методі getModel() і є результатом даного етапу. Якщо кількість параметрів менше або дорівнює 2, для пошуку оптимальних параметрів застосовується покроковий алгоритм оптимізації. В іншому випадку, коли кількість параметрів більше 2, застосовується евристичний еволюційний алгоритм пошуку оптимальних параметрів.

У випадку, коли кількість параметрів менше або дорівнює 2, на *четвертому* етапі відбувається підготовка до проведення експериментів. Клас Experiment, що підтримує JavaFx Application налаштовується під конкретну серію експериментів, щоб отримати графік результатів після імітації моделі. У даному класі мають бути налаштовані методи getResult() (одна точка на графіку при конкретних значеннях), getSeries() (отримання множини точок за якими буде побудовано графік), startExperiment() (проведення декількох серій експериментів з параметрами).

На *п'ятому* етапі запускається проведення експериментів, в результаті чого будуються графіки залежності спостережуваної величини від змінюваного параметра. В результаті даного експерименту можна аналізувати вплив двох параметрів на спостережувану величину.

На *шостому* етапі проводиться аналіз результатів експериментів. Якщо мінімальне значення спостережуваної величини не відстежується, варто змінити значення параметру і продовжити експерименти.

На *сьомому* етапі внаслідок проведення експериментів на попередніх етапах відбувається аналіз отриманих графіків та визначається оптимальне значення параметру, що відповідає мінімальному значенню спостережуваної величини. За потреби проводиться деталізоване дослідження в околі оптимуму.

У випадку, коли кількість параметрів більше 2, на *четвертому* етапі відбувається підготовка до застосування еволюційного алгоритму. Встановлюється кількість параметрів та їх область варіювання, кількість особин в популяції.

На *п'ятому* етапі запускається еволюційний алгоритм та отримуються результати його виконання. Результатом даного етапу є такий набір параметрів з області варіювання, що гарантує мінімальний час виконання алгоритму.

Еволюційний алгоритм, що застосовується для пошуку оптимальних значень, є узагальненою версією генетичного алгоритму [12]. Елементом популяції (особиною) являється набір значень параметрів паралельних обчислень по два або більше параметри. Початкова популяція (генерування 0) формується з випадкових значень, розкиданих в області допустимих значень параметрів. Кожний елемент популяції запускається у «життя», тобто в імітаційну модель паралельних обчислень. Результатом такої життєдіяльності елемента популяції є відгук моделі (час виконання алгоритму). Набори параметрів, які отримали в процесі імітації великі значення відгуку моделі, знищуються. Таким чином відбір особин популяції здійснюється за значенням відгуку моделі. Особини популяції, що пройшли відбір, допускаються до схрещування. Схрещування здійснюється для випадково обраних пар особин популяції шляхом змішування частин наборів параметрів. При цьому доцільно не розривати параметри, що відносяться до одного варіанту. Мутація здійснюється додаванням випадкового відхилення до результату, який отриманий в результаті схрещування, додаванням з рівною ймовірністю -1, 0 або 1. Кожна наступна популяція (генерування j) формується з елементів, що пройшли відбір на попередньому генеруванні (генерування j-1), та з елементів, що створені в результаті схрещування та мутації. У правилі зупинення еволюційного пошуку користувач задає кількість генерувань. Після виконання всіх генерувань повертається набір значень параметрів з найменшим часом виконання алгоритму.

Адаптивний покроковий алгоритм оптимізації зі змінним кроком застосовується у випадку, коли кількість параметрів, що досліджуються, є невеликою (один або два). Він є точним, оскільки множина значень кожного параметра є дискретною та обмеженою. Алгоритм полягає в тому, що при фіксованому значенні одного параметра експериментально будується залежність часу виконання паралельного алгоритму по всім можливим значенням другого параметра. Кількість таких експериментальних одновимірних залежностей дорівнює кількості можливих значень другого параметра. Оптимальне значення задається тією одновимірною експериментальною кривою, на якій досягається мінімальне значення часу виконання паралельного алгоритму. Значення першого параметра відповідає оптимуму відповідної одновимірної експериментальної кривої, значення другого параметра – це значення, для якого ця крива побудована. Тож, при кількості досліджуваних параметрів не більше 2 слід застосовувати описаний метод, оскільки він є точним.

**Застосування методу.** Застосування методу оптимізації параметрів паралельних обчислень здійснювалося на прикладі паралельного алгоритму імітації дискретно-подійної системи. Результати розробки моделі паралельного алгоритму та її дослідження опубліковані в роботі [13]. Послідовний алгоритм імітації відтворює впорядковану в часі послідовність подій. Тому його паралельна реалізація є нетривіальним завданням.

У загальному вигляді алгоритм імітації складається з кроків, кожен з яких включає пошук моменту найближчої події та реалізацію події. Реалізація події – це зміна стану моделі. Реалізація в Java паралельного алгоритму імітації включає потоки, що відтворюють частини системи, що моделюються, переміщуючи локальний час *tLoc* до найближчої події. Взаємодія між потоками реалізується двома парами умов *wait/notify*: перша контролює порожній стан буфера зовнішніх подій, а друга - досягнення максимального значення стану буфера.

Основний потік створює потоки *sim*, імітуючи частини системи, поки не буде досягнуто час моделювання *tMod*. Кожен потік *sim* чекає, поки його буфер зовнішніх подій не буде порожнім, а потім запускає імітацію, доки не буде досягнуто найближчого моменту його зовнішньої події *tLim*. Коли зовнішня подія виконується, потік може продовжувати імітацію, переміщуючи час до наступної найближчої зовнішньої події таким же чином. Щоб уникнути ситуації, коли один потік *sim* займає ресурс комп'ютера, потік призупиняє моделювання, коли достатньо зовнішніх подій надано для іншого потоку.

Для дослідження прискорення алгоритму створено модель, що складається з груп послідовних подій. Оскільки послідовні події набагато важче розпаралелювати, оцінене прискорення буде найменшим з усіх, що можна отримати в загальному випадку. Модель можна розділити на групи залежно від того, скільки потоків буде використано для обчислення імітації. Таким чином, кількість потоків і складність обчислень, що виконуються кожним потоком, можуть змінюватися простим способом.

Тож, паралельний алгоритм імітації реалізується такими методами:

1) Метод *main*, який встановлює значення часу моделювання *tMod*, створює потоки, об'єднує потоки та друкує результати.

2) Метод *run*, який очікує на зовнішню подію, поки *tLoc<tMod*, потім встановлює *tLim* і запускає метод *goUntil*, якщо *tLim< tMod*.

3) Метод *goUntil*, який поки *tLoc<tLim* чекає, якщо буфер зовнішніх подій досяг максимального розміру або цей буфер порожній, потім переміщує *tLoc* до найближчої події та виконує подію; після цього досягається момент *tLim* і повинна бути виконана зовнішня подія.

4) Метод *output*, який змінює стан моделі відповідно до події та за потреби додає дані про зовнішні події до буферів інших частин моделі.

Метод *main* викликає метод run для кожного потоку та чекає його завершення. Метод *run* викликає метод *goUntil*, який, у свою чергу, викликає метод *output*.

Структура моделі відтворює структуру програми. Класами, з яких побудована модель алгоритму, є *Main* і *ThreadModel*. Клас *ThreadModel* агрегує класи *RunSim*, *GoUntilSim*, *OutputSim*, *WaitSim*, які імітують методи "*run*", "*goUntil*", "*output*" і блок синхронізації дій (guarded block) відповідно, та клас *Control*, який зберігає значення моментів часу *tLoc*, *tLim*, *tMin*. Класи *Main*, *RunSim*, *GoUntilSim*, *OutputSim*, *WaitSim* є підкласами класу *PetriSim* і мають в своєму описі мережу Петрі. Клас *WaitSim* визначається мережею Петрі блоку синхронізації дій. Детальний опис мереж та процесу побудови моделі наведено в [13]. На рисунку 1 [13] представлено зв'язки Петрі-об'єктів моделі. Побудова мереж Петрі-об'єктів та конструювання моделі здійснювалось у програмному забезпеченні Parallel Program Simulation [14].

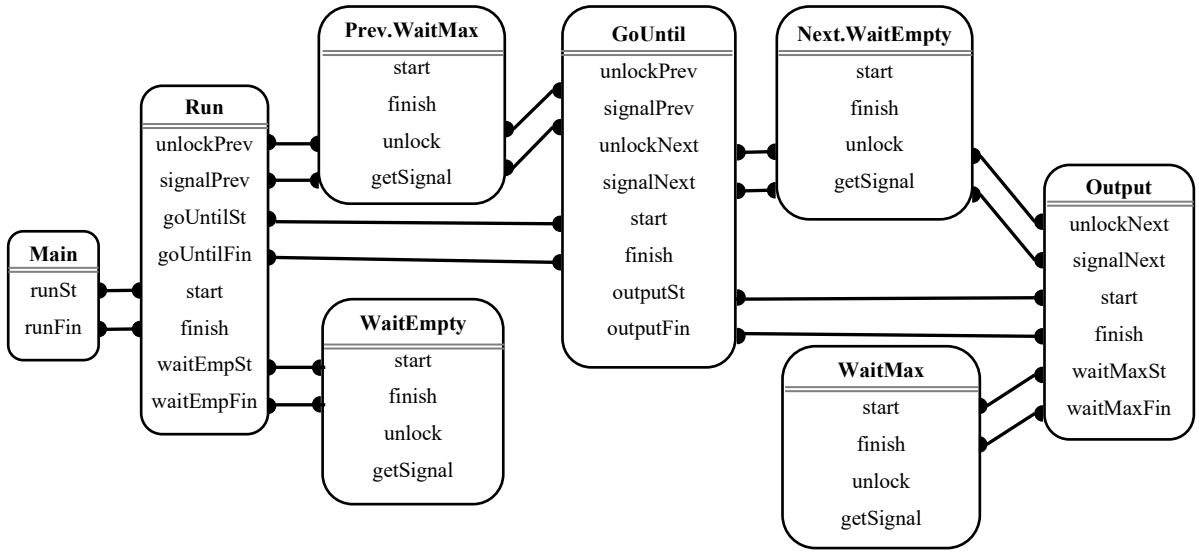


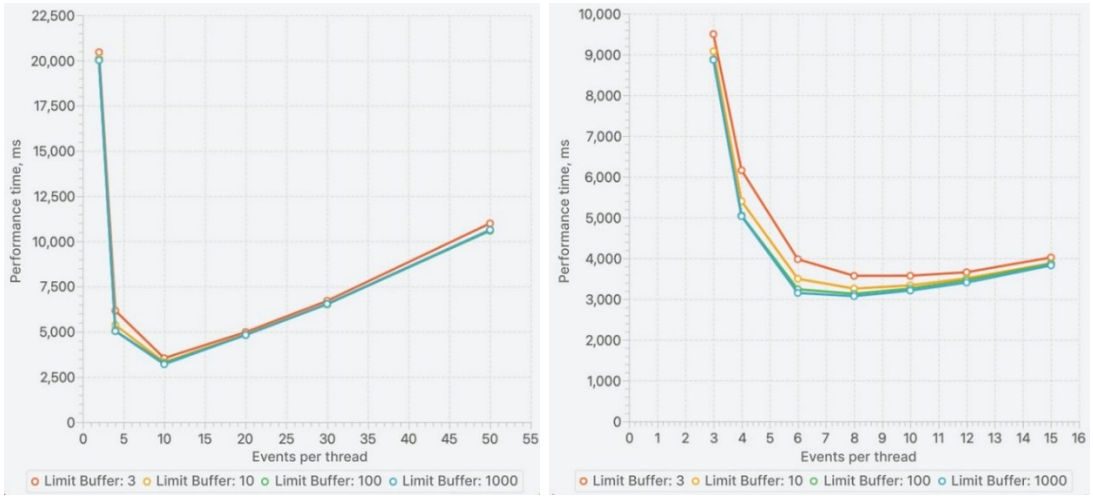

*Рис.1. Ототожнення позицій Петрі-об'єктів Main, Run, GoUntil та Output.*

Після побудови моделі були заміряні та встановлені параметри моделі, близькі до спостережуваних у експерименті з реальним комп'ютерним ресурсом. Були витримані співвідношення часових затримок основних інструкцій, які впливають на алгоритм.

Набір параметрів, що впливає на ефективність алгоритму складається зі складності підзадач в одному потоці та ліміту буфера зовнішніх подій.

Для дослідження впливу складності підзадач у потоці та ліміту буфера зовнішніх подій на час виконання було проведено експеримент із зазначеними параметрами моделі. Для пошуку оптимальних значень спершу застосовувався покроковий алгоритм оптимізації. Спершу визначалось оптимальне значення ліміту буфера зовнішніх подій. Результати моделювання, отримані у випадку використання 2 ядер та при складності моделі у 400 подій, представлені на рисунку 2. Як видно з рисунку, збільшення значення ліміту від 100 не спричиняє значного зменшення часу, адже криві для 100 і 1000 подій накладаються. Тож обмеження ліміту згори складає 1000 подій. Однак, з точки зору завантаженості комп'ютерного ресурсу великий обсяг ліміту потребує великих затрат пам'яті, що може призвести до сповільнення виконання алгоритму. З рисунку видно, що при збільшенні значення ліміту від 100 до 1000 подій виграш у часі невеликий, але затрати в пам'яті зі збільшенням ліміту значно збільшуються. Тож проаналізувавши результати, можна зробити висновок, що оптимальним значенням ліміту буфера зовнішніх подій є 100 подій.

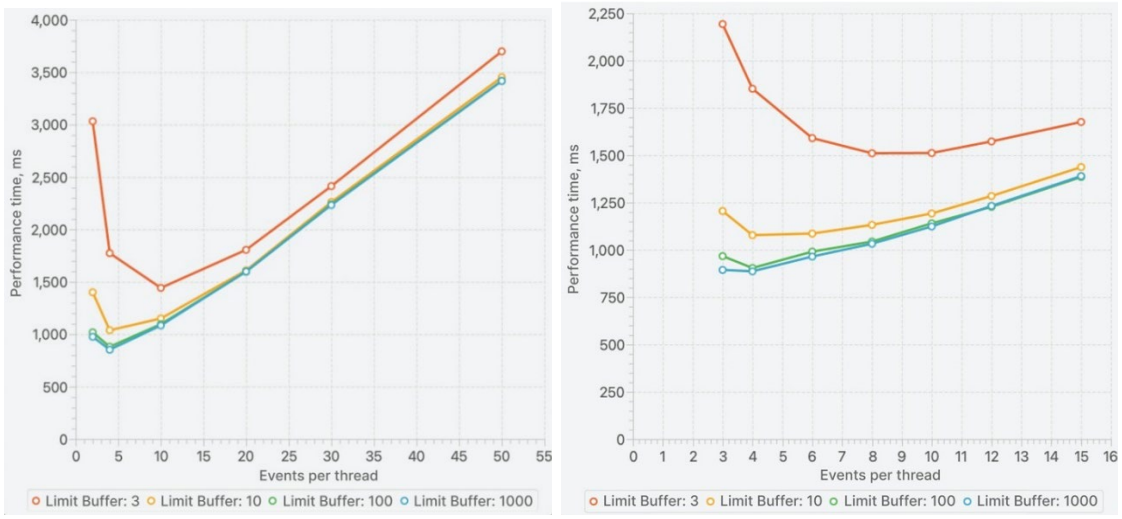
Аналізуючи результати моделювання далі, здійснювався пошук оптимального значення складності підзадачі у потоці. З рисунку 2(а) видно, що мінімальний час виконання досягається при кількості подій, оброблених одним потоком, рівній 10. Була проведена деталізація дослідження в околі оптимуму, яка показала, що насправді найменший час виконання досягається при 8 подіях у потоці (рис. 2(б)).




(а) (б)

*Рис.2. Вплив складності підзадачі у потоці та ліміту буфера зовнішніх подій на час виконання алгоритму в моделі: (а)загальний графік, (б) деталізований графік.*
Джерело: розроблено автором.

З метою оцінки точності алгоритму покрокової оптимізації було проведено аналогічні експерименти з реальною програмою. Результати дослідження впливу двох параметрів багатопотокового алгоритму на час виконання реальної програми представлені на рисунку 3. Для проведення експерименту використовувався комп'ютер MacOS, оснащений двоядерним процесором Intel Core i5 з частотою 1,8 ГГц. Паралельний алгоритм реалізований мовою Java. Виявлено, що найменший час роботи алгоритму забезпечується, якщо складність завдання, що виконується одним потоком, наближається до 4 подій, а ліміт буфера зовнішніх подій дорівнює 100. Варто додати, що час виконання алгоритму сильно залежить від низки факторів таких, як вплив сторонніх запущених застосунків, підключення до інтернету тощо. Також, загально відомо, що Java має властивість прискорювати обчислення при збільшенні навантаження на обчислювальний ресурс (так званий розгін). У зв'язку з цим дуже важливо додавати розігрів обчислювального ресурсу (warm up) [15].




(а) (б)

*Рис.3. Вплив складності підзадачі у потоці та ліміту буфера зовнішніх подій на час виконання алгоритму в реальній програмі: (а)загальний графік, (б) деталізований графік.*
Джерело: розроблено автором.

Як бачимо, оптимальні значення складності підзадачі у потоці та ліміту буфера зовнішніх подій, знайдені за допомогою методу, достатньо близькі до значень, отриманих в результаті експериментів з реальною програмою.

Для того, щоб оцінити вплив похибки у визначенні оптимального значення шляхом моделювання на час виконання алгоритму, було проведено додаткові експерименти з реальною програмою із встановленим значенням параметрів – складність підзадачі та ліміт буфера зовнішніх подій. У першому експерименті складність підзадачі складала 4 події (результат пошуку оптимальних параметрів для реальної програми), у той час, як у другому експерименті даний параметр склав 8 подій (результат моделювання). Ліміту було встановлено значення 100 подій. У таблиці 1 наведені результати обох експериментів. Бачимо, що різниця у часі виконання незначна. Вплив похибки у визначенні оптимального значення на швидкодію алгоритму складає 6%. Тобто, різниця у пошуку оптимального параметру з моделлю і з реальною програмою не значно вплинула на значення швидкодії алгоритму. При необхідності, для отримання точного значення варто провести уточнюючий експеримент на реальній програмі, оскільки модель є все таки спрощеним представленням програми і оцінка часових затримок була достатньо грубою. Проте, враховуючи, що модель дає можливість проводити безліч експериментів і тим самим звузити область пошуку оптимальних значень, можна стверджувати про пришвидшення та спрощення процесу налагодження параметрів паралельних обчислень. Таким чином, модель може бути використана для пошуку найкращих параметрів для запуску паралельного алгоритму.

*Таблиця 1 – Швидкодія алгоритму при оптимальних параметрах, знайдених експериментально та за допомогою методу.*

| Складність підзадачі | Ліміт буфера | Час виконання, мс |
|---|---|---|
| 4 | 100 | 1025,625 |
| 8 | 100 | 1086,5 |

Джерело: розроблено автором.

Слід додати, що у попередньому дослідженні стосовно налаштування параметрів пулу потоків за допомогою Петрі-об’єктного моделювання [16] була розрахована похибка моделювання, яка не перевищувала 8%. Це свідчить про достатню точність моделювання при використанні Петрі-об’єктного підходу.

У випадку застосування еволюційного алгоритму для пошуку оптимальних значень складності підзадач та ліміту буфера зовнішніх подій було отримано результати близькі до попередніх. Оптимальне значення складності підзадачі в експериментах з моделлю та реальною програмою склало 8 і 4, відповідно. У рамках застосованого генетичного алгоритму, особиною є набір параметрів, що досліджується, у нашому випадку це складність підзадачі та ліміт буфера зовнішніх подій. Область варіювання встановлена відповідно від 2 до 50 подій для складності підзадачі та від 3 до 1000 подій для ліміту.

Варто зазначити, що еволюційний алгоритм не слід застосовувати для малої кількості параметрів (один або два) У випадку такої кількості параметрів більш точним та швидким буде покроковий алгоритм. Однак, провівши дослідження навіть на двох параметрах, еволюційний алгоритм показав схожі результати з покроковим алгоритмом: найменший час виконання паралельного алгоритму імітації спостерігається при невеликій складності підзадачі та, відповідно, великій кількості потоків. Такий результат підтверджує, що таке дослідження має сенс і користь, оскільки традиційно кількість потоків обирається відповідно до кількості ядер процесора. Тобто, якщо керуватися загальноприйнятою логікою, в даному прикладі для обчислення системи з 400 подій оптимальним було б використання 4 потоків та відповідно, по 100 подій в підзадачі. Однак по

результатам експериментів видно, що дані параметри не гарантують ефективне виконання алгоритму. Тож, дане дослідження показує, що загальноприйняте правило визначення кількості потоків є не універсальним рішенням і різні задачі потребують різних параметрів.

**Висновки відповідно до статті.** Запропоновано метод оптимізації параметрів паралельних обчислень на основі Петрі-об'єктного моделювання. Оптимізація відбувається за рахунок застосування еволюційного алгоритму або покрокового алгоритму оптимізації на Петрі-об'єктній моделі паралельних обчислень. Продемонстровано приклад застосування методу на паралельному алгоритмі імітації дискретно-подійної системи. Знайдені оптимальні значення достатньо точно відповідають таким, що були виявлені при експериментуванні з паралельним алгоритмом в реальних умовах. Отримані результати свідчать про коректність пошуку оптимальних параметрів на моделі. Подальший розвиток методу полягає у підвищенні точності вимірювання часових затримок базових інструкцій задля підвищення точності моделювання.

## Список використаних джерел

1. Systems and software engineering. High-level Petri nets. Part 1: Concepts, definitions and graphical notation (ISO/IEC 15909-1:2004) [Electronic resource] / International Organization for Standardization – Access mode : https://www.iso.org/standard/38225.html

2. Modeling and analysis of thread pools in an industrial communication platform [Electronic resource] / F. S. de Boer, I. Grabe, M. M. Jaghoori, A. Stam, W. Yi // International Conference on Formal Engineering Methods ICFEM 2009: Formal Methods and Software Engineering : Lecture Notes in Computer Science / eds. K. Breitman A. Cavalcanti. – 2009. – № 5885. – P. 367-386. – Access mode : https://link.springer.com/chapter/10.1007/978-3-642-10373-5_19

3. Kavi, K. Modeling Multithreaded Applications Using Petri Nets [Electronic resource] / K. Kavi, A. Moshtaghi, Dj. Chen // International Journal of Parallel Programming. – 2002. – № 30(5). – P. 353-371. – Access mode : https://www.researchgate.net/publication/220091454_Modeling_Multithreaded_Applications_Using_Petri_Nets

4. Katayama, T. Proposal of a Supporting Method for Debugging to Reproduce Java Multi-threaded Programs by Petri-net [Electronic resource] / T. Katayama, H. Nakamura, Y. Kita // Journal of Robotics, Networking and Artificial Life. – 2014. – № 1 (3). – P. 207-211. – Access mode : https://alife-robotics.org/article/vol1issue3/14955.pdf

5. Giebas, D. Deadlocks Detection in Multithreaded Applications Based on Source Code Analysis [Electronic resource] / D. Giebas, R. Wojszczyk // Applied Sciences. – 2020. – № 10 (2). – P. 532. – Access mode : https://www.mdpi.com/2076-3417/10/2/532

6. Gold, R. Petri Nets in Software Engineering [Electronic resource] / R. Gold // Arbeitsberichte – Working Papers. – Fachhochschule Ingolstadt – University of Applied Sciences, Ingolstadt, 2004. – № 5. – Access mode : https://nbn-resolving.de/urn:nbn:de:bvb:573-203.

7. Owe, O. Deadlock detection of active objects with synchronous and asynchronous method calls [Electronic resource] / O. Owe, I. C. Yu // Norsk Informatikkonferanse (NIK). – Halden, Norway, 2014. – Access mode : http://ojs.bibsys.no/index.php/NIK/article/view/19.

8. Software Verify LTD. Deadlock Detection and Thread Monitoring [Electronic resource]. – Access mode : https://www.softwareverify.com/products/#threads

9. Liao, H. Concurrency Bugs in Multithreaded Software: Modeling and Analyzing Using Petri Nets / H. Liao // Discrete Event Dynamic Systems. – 2013. – № 23(2). – P. 157-195.

10. Xiang, D. Detecting Data Inconsistency Based on the Un-folding Technique of Petri Nets / D. Xiang // IEEE Transactions on Industrial Informatics. – 2017. – № 13(6). – P. 2995-3005.

11. Xu, Z. PVcon: Localizing Hidden Concurrency Errors With Prediction and Verification // IEEE Access. – 2020. – № 8. – P. 165373-165386.

12. Стеценко, І. В. Моделювання систем : навч. посіб. / І. В. Стеценко. – Черкаси : ЧДТУ, 2010.

13. Stetsenko, I. V. Parallel algorithm development and testing using Petri-object simulation / I. V. Stetsenko, O. A. Pavlov, O. Dyfuchyna // International Journal of Parallel, Emergent and Distributed Systems. – 2021. – № 36(6). – P. 549-564.

14. Dyfuchyna, O. Parallel Program Simulation (PPS) [Electronic resource] / O. Dyfuchyna. – Access mode : https://github.com/sashadif/PPS

15. JMH - Java Microbenchmark Harness. [Electronic resource] – Access mode : https://jenkov.com/tutorials/java-performance/jmh.html

16. Stetsenko, I. V. Thread Pool parameters tuning using simulation / I. V. Stetsenko, O. A. Pavlov, O. Dyfuchyna //Advances in Intelligent Systems and Computing. – 2020. – № 938. – P. 78-89.

## References

1. International Organization for Standardization. (2004). *Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation.* (ISO/IEC 15909-1:2004). Retrieved from https://www.iso.org/standard/38225.html.

2. de Boer, F.S., Grabe, I., Jaghoori, M.M., Stam, A., Yi, W. (2009). Modeling and analysis of thread pools in an industrial communication platform. *International Conference on Formal Engineering Methods ICFEM 2009: Formal Methods and Software Engineering. Lecture Notes in Computer Science, (5885)*, 367-386. DOI: 10.1007/978-3-642-10373-5_19.

3. Kavi, K., Moshtaghi, A., Chen, Dj. (2002). Modeling Multithreaded Applications Using Petri Nets. *International Journal of Parallel Programming*, *30(5),* 353–371. DOI: 10.1023/A:1019917329895.

4. Katayama, T., Nakamura, H., Kita, Y. (2014). Proposal of a Supporting Method for Debugging to Reproduce Java Multi-threaded Programs by Petri-net. *Journal of Robotics, Networking and Artificial Life*, *1(3),* 207-211. DOI: 10.2991/jrnal.2014.1.2.3.

5. Giebas, D., Wojszczyk, R. (2020). Deadlocks Detection in Multithreaded Applications Based on Source Code Analysis. *Applied Sciences*, *10(2),* 532. DOI: 10.3390/app10020532.

6. Gold, R. (2004). Petri Nets in Software Engineering. *Arbeitsberichte – Working Papers*, (5), Fachhochschule Ingolstadt – University of Applied Sciences, Ingolstadt. URL: https://nbn-resolving.de/urn:nbn:de:bvb:573-203.

7. Owe O., Yu I.C. (2014). Deadlock detection of active objects with synchronous and asynchronous method calls. *Norsk Informatikkonferanse (NIK)*. Halden, Norway. URL: http://ojs.bibsys.no/index.php/NIK/article/view/19.

8. Software Verify LTD. (2023). *Deadlock Detection and Thread Monitoring.* Retrieved from https://www.softwareverify.com/products/#threads.

9. Liao, H. (2013). Concurrency Bugs in Multithreaded Software: Modeling and Analyzing Using Petri Nets. *Discrete Event Dynamic Systems, 23(2),* 157–195. DOI: 10.1007/s10626-012-0139-x.

10. Xiang, D., (2017). Detecting Data Inconsistency Based on the Un-folding Technique of Petri Nets. *IEEE Transactions on Industrial Informatics*, *13(6),* 2995-3005. DOI: 10.1109/TII.2017.2698640.

11. Xu, Z. (2020). PVcon: Localizing Hidden Concurrency Errors With Prediction and Verification, *IEEE Access, 8,* 165373-165386. 0.1109/ACCESS.2020.3022992.

12. Stetsenko, I.V. (2010). *System Modeling*. Cherkasy: ChDTU.

13. Stetsenko, I.V., Pavlov, O.A., Dyfuchyna, O. (2021) Parallel algorithm development and testing using Petri-object simulation. *International Journal of Parallel, Emergent and Distributed Systems*, 36(6), 549-564. DOI: 10.1080/17445760.2021.1955113.

14. Dyfuchyna, O. (2023). *Parallel Program Simulation (PPS)*. Retrieved from https://github.com/sashadif/PPS

15. Jenkov J. (2015). *JMH - Java Microbenchmark Harness*. Retrieved from https://jenkov.com/tutorials/java-performance/jmh.html

16. Stetsenko, I.V., Pavlov, O.A., Dyfuchyna, O. (2020). Thread Pool parameters tuning using simulation. *Advances in Intelligent Systems and Computing, 938,* 78-89. DOI: 10.1007/978-3-030-16621-2_8.

Отримано 20.09.2023

TECHNICAL SCIENCES AND TECHNOLOGIES

*UDC 004.41::004.94*

***Oleksandra Dyfuchyna***

Ph.D. student of Department of Information Systems and Technologies
National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute» (Kyiv, Ukraine)
**E-mail:** dyfuchyna@gmail.com **ORCID:** https://orcid.org/0000-0002-5477-4533 **SCOPUS Author:** 57202215305

# THE METHOD FOR PARALLEL COMPUTING PARAMETERS OPTIMIZATION

*Modern information technologies require fast performance of algorithms, which can be achieved with the help of parallel computing. However, depending on the parameters that determine the characteristics of the subtasks and the mechanisms of their interaction, the use of parallel computing can lead to both speeding up and slowing down the computations. Testing parallel algorithms in real conditions is resource-consuming, in view of this, the paper proposes a method for optimizing the parameters of parallel computing based on Petri-object modeling and the evolutionary method.*

*Nowadays, there is no unified method for creating a model of parallel computing and, accordingly, there is no tool other than a real program that can be used to optimize the parameters of a parallel program. The lack of simulation tools hinders the development of highly efficient parallel computing. Despite the fact that there are not many tools for simulating parallel computing, there is a movement of research in this direction. The analysis of the existing tools of testing multithreaded programs showed that they are aimed, first of all, at the analysis of the correctness of the execution of computations, but not at the analysis of the efficiency of parallel computing.*

*The research objective is to improve the efficiency of using parallel computing in information technologies by optimizing the parameters of parallel computing based on Petri-object models, which can be used to estimate the execution time of a parallel algorithm.*

*The proposed method is based on the simulation of parallel computing by a stochastic Petri net and the application of models in an evolutionary method or a step-by-step optimization algorithm to find parameters that will ensure the efficient performance of parallel computing. As an example of the application of the method, a model of a parallel algorithm for discrete-event system simulation was built and investigated. The optimal values of the parameters found using the method quite accurately correspond to those found during experimentation with the parallel algorithm in real conditions.*

***Keywords:*** *cyber-attack simulation, penetration test, vulnerabilities, Petri net.*

*Tables: 1. Fig.: 3. References: 14.*