УДК 681.3

А.Ю. Дорошенко, О.С. Новак, П.А. Іваненко, А.М. Старушик

АВТОТЮНІНГ ПАРАЛЕЛЬНИХ ПРОГРАМ З ВИКОРИСТАННЯМ СИСТЕМИ АНАЛІЗУ ДАНИХ IBM WATSONS ANALYTICS

В даній роботі представлена аналітична модель методу автоматичного налаштування паралельних програм (автотьюнінгу). Описана програмна реалізація цієї моделі за основою формальних перетворень програм та використання експертних даних, як основи процесу оптимізації з подальшим аналізом отриманих результатів системою IBM Watsons Analytics. Представлені результати практичного експерименту які підтверджують ефективність використовуваного підходу при оптимізації паралельних програм.

Ключові слова: автотьюнінг, паралельні програми, IBM Watsons Analytics, оптимізація, статистичне моделювання.

Вступ

Паралельні обчислення – спосіб організації комп'ютерних обчислень, при якому програми розробляються, як набір взаємодіючих обчислювальних блоків, що працюють паралельно (одночасно) та являються декомпозицією певної задачі. Головна перевага паралельних програм – це можливість одночасного виконання певного набору операцій з метою досягнення високої продуктивності.

Незважаючи на загальновідомий закон Мура – емпіричне спостереження, зроблене Гордоном Муром, згідно якому потужність процесорів подвоюється кожні 18–24 місяців, реальна ситуація на поприщі паралельних обчислень є не настільки хорошою. Ефективність та продуктивність сучасних паралельних систем знаходиться нижче позначки в 10 % від теоретично можливої пікової величини, відповідно до звіту Міжвідомчої комісії з розвитку надпотужних обчислень США [1]. Яскравим прикладом цього є результат дослідження Національного науково-дослідницького обчислювального центру в області енергетики (NERSC), показаний на рис. 1.

Постійний технологічний ріст мікропроцесорної сфери, відповідно до закону Мура, призводить до різкого зростання кривої пікової продуктивності. Виникає певний дисбаланс з іншими технологічними та аналітичними аспектами, які розвиваються не так швидко, що призводить до значного розходження пікової можливої та реальної продуктивності паралельних систем.

Наявність потужної обчислювальної бази не гарантує факту ефективного її використання. Швидкодія паралельної системи значною мірою залежить від обраної архітектури, вдало розробленого алгоритму, правильної декомпозиції поставленої задачі та ряду інших факторів. Тому розробка продуктивного та ефективного паралельного рішення являється доволі нетривіальною задачею, тим паче в час масового використання багатоядерних мікропроцесорів.

1. Оптимізація паралельних програм

Складні науково-технічні задачі потребують значних обчислювальних потужностей, тому оптимізація паралельних програм являється невід'ємною частиною їх процесу розробки.

Царину методів оптимізації паралельних програм умовно можна поділити на дві групи: статичні та динамічні оптимізатори.

Статичні оптимізатори [2] – це додаткове програмне забезпечення та спеціалізовані компілятори, які мають певну

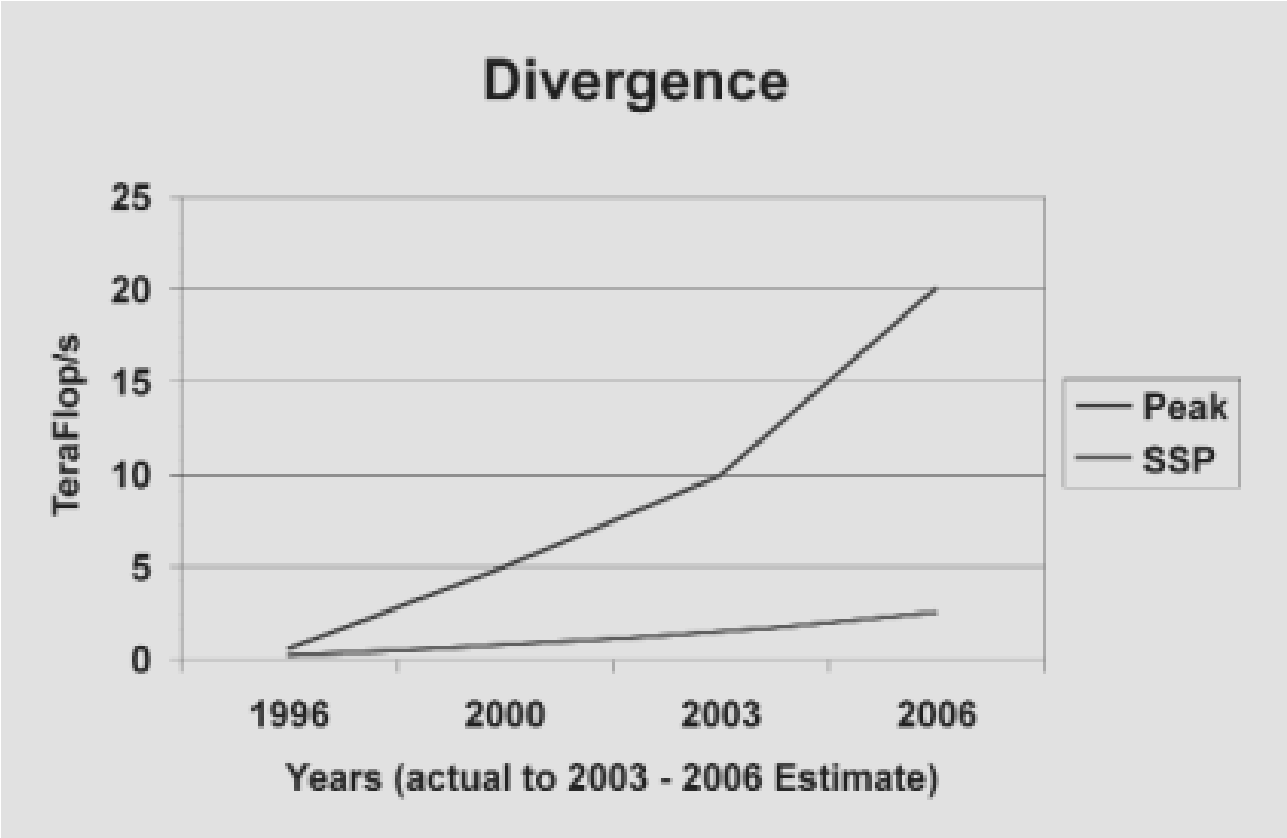

Рис. 1. Розходження можливої пікової продуктивності (Peak) та реальної продуктивності паралельних систем (SSP)

експертну базу правил, на основі якої проводиться статичний аналіз вихідного коду. Перевагами такого підходу є відносна простота та швидкість, оскільки виконується процес послідовного аналізу вихідної програми та заміни певних структур, на аналогічні оптимізовані структури описані в експертній базі правил. Основним недоліком являється те, що такий підхід працює лише з вихідним кодом як таким, не враховуючи специфіку робочого алгоритму, середовища виконання, показники якості та точності результату. Складність та архітектурні особливості сучасних обчислювальних систем значно ускладнюють розвиток даного підходу.

Динамічна оптимізація – дозволяє емпіричним методом, в автоматичному режимі визначити найкращу конфігурацію для паралельної програми та впровадити цю конфігурацію на рівні вихідного коду. Даний підхід потребує додаткового аналізу та ресурсів для імплементації та подальшої роботи.

2. Автотьюнінг

Розглянемо детальніше концепцію автоматичної оптимізації паралельних програм – автотьюнінгу [3].

Концепція автотьюнінгу, останнім часом стала стандартом для вирішення задачі оптимізації паралельних програм. Її популярність зумовлена простотою застосування й незалежністю від якісних характеристик обчислювальної системи, оскільки зазвичай оптимізація виконується в цільовому обчислювальному середовищі, а отримана конфігурація паралельної програми залишається ефективною допоки не змінюються характеристики середовища виконання. Такий підхід дозволяє абстрагуватися від конкретного обчислювального середовища на етапі створення паралельної системи та в подальшому оптимізувати її для цільової платформи.

Звичайно повністю автоматизувати процес неможливо із-за декількох факторів: варіанти програми задаються розроб-

ником, оскільки він являється експертом в даній предметній області та знає які характеристики програми можуть значно вплинути на оптимізацію; тільки розробник може правильно оцінити роботу автотьюнера, оскільки інколи важливим є не тільки час виконання, а й точність результату, використані ресурси тощо.

Концепція автотьюнінгу уже довела свою універсальність й ефективність, проте вона не позбавлена недоліків: необхідність використання автотьюнера, значний час його роботи, інтеграція термів автотьюнера у вихідний код програми.

Особливістю автотьюнінгу є те, що найкраща конфігурація програми визначається емпіричним способом, на основі статистичних даних, отриманих автотьюнером. Конфігурація – це набір параметрів у вихідному коді програми, які тією чи іншою мірою впливають на продуктивність паралельної програми (кількість потоків, ітерацій, величина розбиття тощо). Автотьюнер формує проміжні конфігурації на основі заданих правил та оцінює їх ефективність в конкретному обчислювальному середовищі. На основі аналізу отриманих статистичних даних формується найкраща конфігурація для паралельної програми, яка вводиться на рівні вихідного коду.

В цілому алгоритм дії автотьюнера можна розбити на наступні кроки:

– виділення та аналіз метаданих з вихідного коду;

– формування, на основі виділених метаданих, множини конфігурацій для вихідної програми, що являє собою сукупність параметрів які тією чи іншою мірою впливають на продуктивність паралельної програми. Кожна конфігурація являє собою унікальну трансформацію вихідного коду. В результаті якої генерується новий варіант програми;

– емпіричний аналіз ефективності отриманих конфігурацій;

– визначення найбільш ефективної конфігурації.

У загальному випадку кількість ітерацій роботи автотьюнера дорівнює кількості виділених конфігурацій, оскільки перевіреною має бути кожна з них, для подальшого емпіричного аналізу. Тобто час роботи автотьюнера прямо пропорційний кількості виділених конфігурацій, що не завжди зручно в умовах обмежених ресурсів. В такому випадку раціонально ввести певні часові обмеження на, по закінченню яких ітерації автотьюнера закінчуються та проводиться емпіричний аналіз отриманих результатів. Також, одним із підходів до оптимізації роботи автотьюнера є використання спеціалізованих пошукових алгоритмів [4], що вибиратимуть наступну конфігурацію для перевірки. Інтелектуальне задання конфігурацій може значно зменшити множину, але потребує додаткового аналізу на етапі інтеграції автотюнера. Дані підходи значно зменшують час роботи автотюнера, але в загальному випадку потребують додаткових ресурсів для імплементації.

Процес емпіричного аналізу ефективності виділених конфігурацій також являється нетривіальною задачею та може займати значний час. Велика кількість параметрів та, як результат, конфігурацій підвищують складність аналізу, виділення трендів та кореляцій з характеристиками програми. Тому для аналізу отриманих автотьюнером результатів часто використовують окремі програмні комплекси, що дозволяють знаходити статистичні залежності між великою кількістю параметрів, так звана гібридна модель автотьюнінгу [5].

3. Програмна реалізація автотьюнера

Програмна реалізація автотьюнера створена на основі використання правил перетворення вихідного коду – термів, за допомогою яких виділяються експертні знання на основі яких генеруються нові конфігурації вихідної програми для подальшого аналізу. Терми являють собою інструмент для перетворення вихідного коду, заданого розробником. Автотьюнер виділяє необхідну інформацію з термів та на основі отриманих експертних даних проводить декларативні перетворення (без зміни логіки роботи) програми та аналізує отримані результати.

Синтаксично терми являються коментарями, тому їх введення не впливає на вихідний код програми, не змінює структури паралельної програми та не впливає на компіляцію.

На даний момент розроблюваний автотьюнер підтримує два терми.

Терм tuneVarTerm задає область значень числової змінної та включає у себе такі параметри:

– var – ідентифікатор змінної;

– start – початкове значення змінної;

– stop – кінцеве значення змінної;

– step – крок зміни значення змінної (за замовчуванням дорівнює 1).

Наприклад, для змінної threshold із значенням, що лежить у межах [1...800] та кроком зміни значення 1 терм виглядає наступним чином:

```
//tuneVarTerm var=threshold start=1 stop=800 step=1

int threshold = 1;
```

Терм bidirCycle ідентифікує цикл, що не залежить від порядку ітерації та може бути реверсивно зміненим. Наприклад, автотюнер випробує пряму та зворотню ітерацію для наступного циклу:

```
//bidirectionalCycle
for (int i=0; i < val; i++)
{
        //do something
}
```

Зміна напрямку ітерації циклу впливає на ефективність використання програмного кешу і, як наслідок, на загальний час виконання паралельної програми.

Особливістю розробленого автотюнера є те, що аналіз отриманих результатів виконується за допомогою зовнішнього суперкомп'ютера IBM Watsons Analytics [6], що мінімізує час на аналіз та дозволяє знайти не тільки очевидні, а й приховані глибокі кореляції вхідних даних.

4. IBM Watsons Analytics

Проблема пошуку прихованих закономірностей та взаємозв'язків – відома і за межами машинного навчання. За останні 20 років було створено багато методологій інтелектуального аналізу даних і згідно з опитуванням [7], 43 % спеціалістів цієї області використовують методологію CRISP-DM, водночас як частка всіх інших методологій складає 30 % (27 % припадає на власні методології). І хоча у відомих методологій існують відмінності, загалом процес пошуку даних можна представити у такому вигляді:

– постановка задачі;

– початковий аналіз даних;

– підготовка даних;

– моделювання;

– оцінка;

– використання результатів.

IBM Watson Analytics – це інтелектуальна служба аналізу та візуалізації даних, яка дозволяє користувачам самостійно та швидко виявляти явні та приховані закономірності введених даних. Завдяки аналізу структури даних, когнітивним можливостям да відкритому API [8] користувачі можуть вільно працювати з даними та отримувати необхідні результати.

IBM Watson Analytics являє собою аналітичний інструмент який створювався для того, щоб максимально автоматизувати процес аналізу даних, а саме: початковий аналіз даних, підготовку даних та моделювання. Процес роботи з Watson Analytics виглядає так:

– вибір джерела даних. Як джерело можна використовувати як дані з електронної таблиці, так і автоматично імпортувати дані з сторонніх джерел (наприклад, з Twitter чи Salesforce);

– дослідження. За допомогою навідних запитань про завантажені дані, Watson допомагає візуалізувати дані для подальшого аналізу;

– прогнозування. Ватсон у автоматичному режимі виділяє основні кореляції в даних, та дає можливість користу-

вачу уточнювати їх за допомогою питань на природній мові;

– монтування. Система надає користувачам можливість зберегти оброблені дані у вигляді наглядних інфографіків для подальшої презентації.

5. Практичний експеримент

Як дані для тестування обрано заміри часу виконання адаптивного алгоритму сортування, який виконує сортування злиттям чи вставкою у залежності від розміру блока числового масиву даних. Частина вихідного коду має наступний вигляд:

```
//tuneVarTerm var=threshold start=1 stop=800 step=1
 int threshold = 1;
 if (high-low < threshold)
 {
   InsertionSort(array, low, high);
 }
 else
 {
   MergeSort(array, low, high);
 }
```

В рамках даного експерименту використовувались дані про сортування множини випадкових чисел потужністю 10^6. Як можливі параметри маємо $C = \{Tcn, Th\}$, де – кількість потоків, які виконують обчислення одночасно, – розмір блоку, при якому відбувається переключення алгоритмів сортування. Як метрика для оцінки моделі з заданими Tcn та Th використовується T – час виконання алгоритму для конфігурації, заданої Tcn та. В нашому випадку $Tcn = \{2, 4, 6, ..., 16\}$ та $Th = [1, 800]$.

Конфігурація експериментального середовища:

– Процесор Intel® Core™ i5-2410M (кеш 3Мб, до 2.90 Гц)

– 4 ГБ DDR2 RAM.

Графічне моделювання вхідних даних системою IBM Watsons Analytics показано на рис. 2.

При аналізі даних IBM Watson Analytics виявив явну кореляцію між кількістю потоків та результуючим часом виконання (рис. 3 та 4).

Також було виявлено деяку додаткову інформацію, а саме: негативну кореляцію між часом виконання та розміром блока (при низькій кількості потоків), як показано на рис. 5. Слід зазначити, що для великої кількості потоків такої залежності нема.

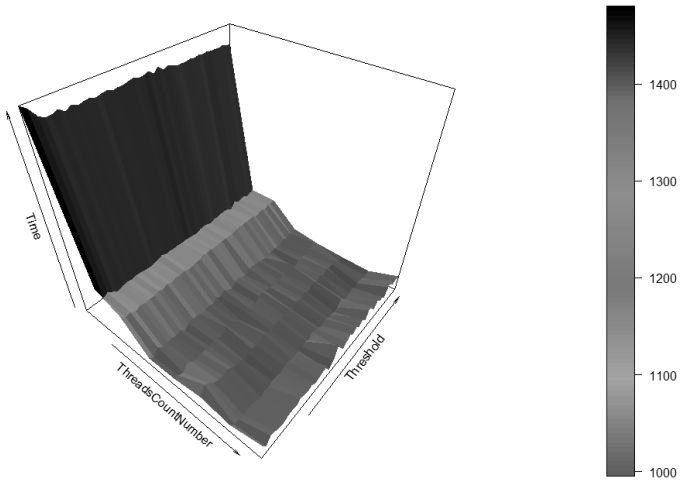

Рис. 2 Графічне моделювання даних

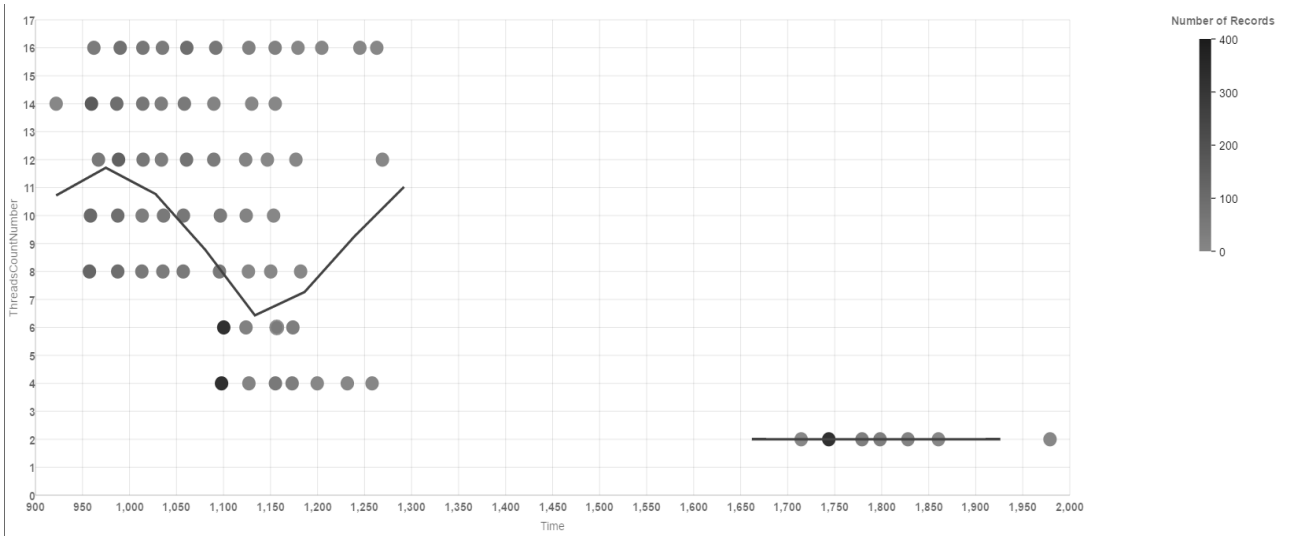

Рис. 3. Результат аналізу даних за допомогою IBM Watsons Analytics

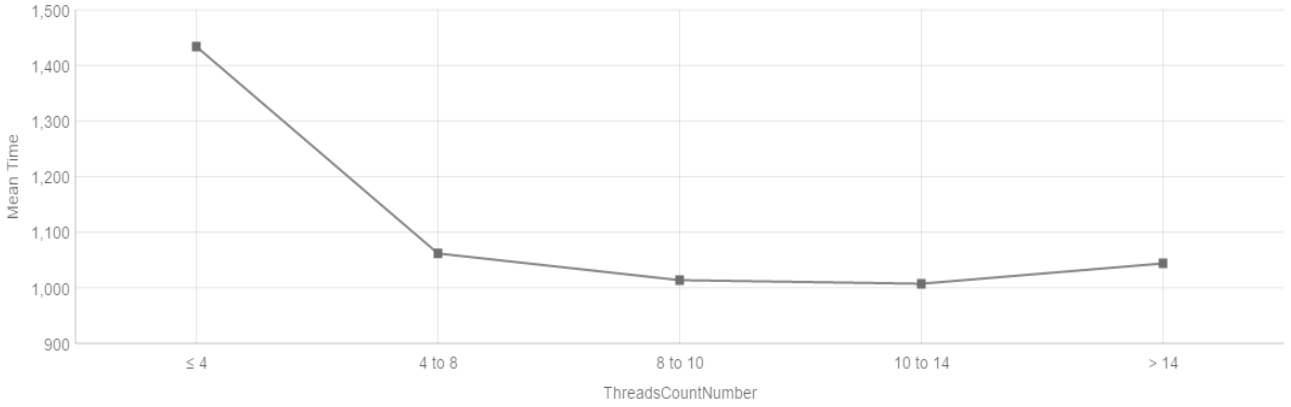

Рис. 4. Кореляція між кількістю потоків та результуючим часом виконання

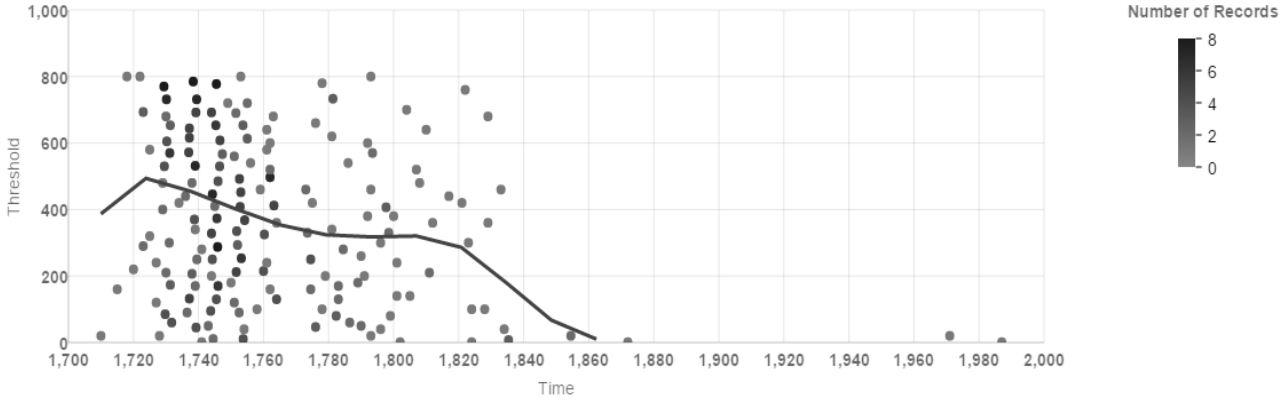

Рис. 5. Додатковий результат аналізу даних за допомогою IBM Watsons Analytics

6. Аналіз роботи IBM Watsons Analytics

Для виявлення таких кореляцій IBM Watsons Analytics використовує *r-to-t* перевірку Фішера [9]. Цей метод полягає у тому, що перевіряється статистична важливість коефіцієнта кореляції Пірсона. За допомогою трансформації Фішера (формула 1), коефіцієнт Пірсона r перетворюється в значення t, чия вага може бути перевірена на t-розподілі.

$$t = \frac{1}{2 * \ln\left(\frac{1+r}{1-r}\right)}. \qquad (1)$$

В даному випадку, для перевірки кореляції між кількістю потоків та часом виконання була побудована таблиця.

Таблиця. Статистичні значення

Статичний показник	Значення
T	-49.82
Df	3278
Sig.	0.00
Effect Size (*ES*)	0.43

Тут t – значення t-критерію Стьюдента. Так як в даному випадку розміри вибірок однакові, то він обчислювався за формулою 2, де M_1, M_2 – середнє арифметичне, σ_1, σ_2 – стандартне відхилення, N – розмір вибірок;

$$t = \frac{|M_1 - M_2|}{\sqrt{\frac{\sigma_1^2 + \sigma_2^2}{N}}}. \qquad (2)$$

Інші параметри означають: *df* – кількість ступенів свободи; *Sig.* – *p*-значення, що відповідає вірогідності виникнення похибки першого роду; *Effect size* – вимір значущості статистичного результату. В даному випадку використовується квадрат кореляції Пірсона (r^2).

На основі даних із таблиці, Watson Analytics робить висновки, що

– оскільки t є статистично значущим, можна відкинути 0-гіпотезу про те, що кореляція Пірсона між часом та кількістю потоків дорівнює 0;

– оскільки $r = -0.66$, то можна казати про наявність явної кореляції між часом та кількістю потоків;

– присутній явний тренд на зменшення часу виконання при збільшенні кількості потоків.

Висновки

Різноманіття та складність сучасних паралельних архітектур практично унеможливлює створення паралельних програм, ефективних на будь-якій із них. Кожен паралельний програмний комплекс потребує певної конфігурації для конкретного середовища виконання для досягнення максимальної ефективності роботи.

Методологія автотьюнінгу дозволяє значно скоротити етап оптимізації паралельних програм, тим самим покращити якість програмного забезпечення та зекономити час його розробки.

Імплементація автотьюнера за допомогою метаданих (термів) полегшує та ще більше автоматизує процес оптимізації та пошуку ефективних конфігурацій за рахунок того, що структура та логіка вихідного коду не змінюється, а терми, розміщені в коментарях, абсолютно не впливають на роботу програми.

Емпіричний аналіз результатів системою IBM Watsons Analytics дозволяє знаходити явні та не явні кореляції між множинами параметрів та результатами швидкодії, та подати ці дані у зрозумілому форматі. Так мінімізується процес аналізу даних в загальному, оскільки, як спеціалізована платформа, IBM Watsons Analytics має оптимізовані алгоритми аналізу та значні обчислювальні ресурси.

Література

1. Federal plan for high-end computing: Report of the High-end computing revitalization task force (HECRTF) [Електронний ресурс]. Режим доступу до ресурсу: http://www.nitrd.gov/pubs/2004_hecrtf/20040702_hecrtf.pdf
2. Automatic Parallelization with Intel Compilers [Електронний ресурс]. Режим доступу до ресурсу: http://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers
3. Naono K., Teranishi K., Cavazos J., Suda R . Software automatic tuning from concepts to state-of-the-art results – New York: Springer, 2010. 240 p.
4. Asanovic K. The Landscape of Parallel Computing Research: A View From Berkeley. Technical Report, University of California, Berkeley, 2006.
5. Дорошенко А.Ю, Іваненко П.А., Новак О.С. Гібридна модель автотьюнінгу з використанням статистичного моделювання. *Проблеми програмування*. 2016. № 4. С. 27–32.
6. IBM Watson Analytics [Електронний ресурс]. Режим доступу до ресурсу: https://www.ibm.com/ru-ru/marketplace/watson-analytics
7. Analytics of data-mining and data-science methodology [Електронний ресурс]. Режим доступу до ресурсу: https://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html
8. Watson Analytics Developer Center [Електронний ресурс]. Режим доступу до ресурсу: https://developer.ibm.com/api/view/id-160:title-Watson_Analytics
9. Fisher r-to-t test [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/support/knowledgecenter/SSWLVY_1.0.1/com.ibm.spss.analyticcatalyst.help/analytic_catalyst/fisher_r_to_t.html

References

1. Federal plan for high-end computing: Report of the High-end computing revitalization task force (HECRTF) – Retrieved from http://www.nitrd.gov/pubs/2004_hecrtf/20040702_hecrtf.pdf
2. Automatic Parallelization with Intel Compilers – Retrieved from http://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers
3. Naono K., Teranishi K., Cavazos J., Suda R . Software automatic tuning from concepts to state-of-the-art results – New York: Springer, 2010. – 240 p.
4. Asanovic K. The Landscape of Parallel Computing Research: A View From Berkeley. Technical Report, University of California, Berkeley, 2006
5. Doroshenko A.Yu., Ivanenko P.A., Novak O.S., Hybrid model of autotuning that use statistical modeling. Problems in Programming. 2016, N 4. P. 27–32.
6. IBM Watson Analytics – Retrieved from https://www.ibm.com/ru-ru/marketplace/watson-analytics
7. Analytics of data-mining and data-science methodology – Retrieved from https://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html
8. Watson Analytics Developer Center – Retrieved from https://developer.ibm.com/api/view/id-160:title-Watson_Analytics
9. Fisher r-to-t test – Retrieved from https://www.ibm.com/support/knowledgecenter/SSWLVY_1.0.1/com.ibm.spss.analyticcatalyst.help/analytic_catalyst/fisher_r_to_t.html.

Одержано 10.12.2017

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу
теорії комп'ютерних обчислень,
професор кафедри автоматики та
управління в технічних системах
НТУУ "КПІ".
Кількість наукових публікацій в
українських виданнях – понад 150.

Кількість наукових публікацій в
зарубіжних виданнях – понад 50.
Індекс Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

Іваненко Павло Андрійович,
молодший науковий співробітник.
http://orcid.org/0000-0001-5437-9763,

Новак Олександр Сергійович,
аспірант.
http://orcid.org/0000-0002-1665-7360.

Старушик Артем Миколайович,
магістрант,
https://orcid.org/0000-0002-6419-7792.

Місце роботи авторів:

Національний технічний університет
України «КПІ імені Ігоря Сікорського»
03056, Київ, пр. Перемоги, 37

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.

E-mail: starushykart@gmail.com,
doroshenkoanatoliy2@gmail.com