УДК 004.853+004.855.5

О. В. Годич*, М. В. Давидов, Ю. В. Нікольський**, В. В. Пасічник**, Ю. М. Щербина***
* Львівський національний університет імені Івана Франка
** Національний університет "Львівська політехніка"

ОБЧИСЛЮВАЛЬНІ АСПЕКТИ АНАЛІЗУ ДАНИХ НА ОСНОВІ КАРТ КОХОНЕНА

Тенденції останнього десятиліття розвитку архітектури центральних обчислювальних процесорів чітко вказують на використання багатоядерного підходу зі збільшенням кількості процесорів кожні вісімнадцять місяців відповідно до закону Мура. Зміщення від архітектури швидких одноядерних до повільніших багатоядерних процесорів ставить питання про масштабовність великого класу обчислювальних алгоритмів, зокрема алгоритмів аналізу даних. Висвітлено результати дослідження з використання сучасних парадигм паралелізації у програмуванні з метою масштабування процесів аналізу даних на основі карт Кохонена.

Ключові слова: карти Кохонена, паралельні обчислення, аналіз даних.

The trends of the past decade in architecture of the central processing unit show a clear direction towards multi-core processors with the number of cores increasing every eighteen months according to the Moore's law. The shift from fast single-core to slower multi-core CPUs poses a question of scalability for a vast class of computational algorithms including algorithm used for data analysis. This paper presents the research result of using state of the art parallelisation programming paradigms to scale data analysis processes based on Self-Organising Maps.

Key words: Self-Organising Maps, parallel computations, data analysis.

Вступ

Самоорганізовна карта Кохонена – це сучасне математичне і програмне забезпечення обчислювальних систем, яке використовується для візуалізації й аналізу даних високої розмірності [1, 2]. Результати дослідження теорії і практичних застосування карт Кохонена авторами статті висвітлено у працях [3–9]. У [3] досліджено ефективність низки алгоритмів навчання карт Кохонена, завдяки чому істотно поліпшено якість класифікації з використанням класифікаторів на основі карт Кохонена. Висвітлені у працях [8, 9] дослідження забезпечують можливість усунення низки недоліків алгоритмів навчання карт Кохонена, пов'язаних із проблемою ініціалізації.

З огляду на тенденцію розвитку мікропроцесорної архітектури у бік багатоядерних процесорів, надзвичайно важливим стає питання масштабовності математичного і програмного забезпечення саме для систем із багатоядерними центральними процесорами. На питання паралелізації процесів навчання карт Кохонена почали звертати увагу із середини 1990-х, але на той час як середовище функціонування карт Кохонена використовували нейрокомп'ютери або суперкомп'ютери зі спеціалізованими процесорними архітектурами. У працях [10–13] подано широкий спектр підходів до паралелізації процесів навчання карт Кохонена. Найповніше стан сучасних досягнень відображено у [10] і [12]. Важливим недоліком цих досліджень є відсутність апробацій розроблених алгоритмів для обчислювальних систем на основі сучасних архітектур багатоядерних процесорів широкого використання від компаній Intel та AMD. Також у них відсутній порівняльний аналіз програмних технологій реалізації паралельних обчислень, що повністю залишає відкритим питання способу й ефективності програмної реалізації розроблених у цих дослідженнях алгоритмів паралельного навчання та використання карт Кохонена.

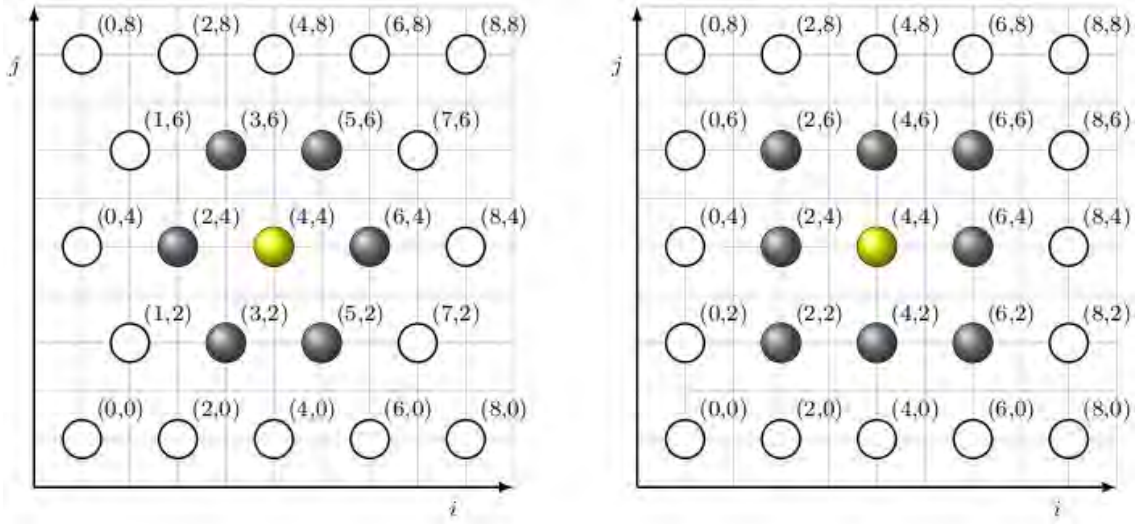
У цій статті висвітлено результати дослідження реалізації ефективних обчислювальних процесів самоорганізації карт Кохонена із застосуванням обчислювальних машин із

багатоядерними центральними процесорами. Розроблено алгоритм навчання на основі декомпозиції карт Кохонена з метою реалізації паралельних обчислень процесів навчання і розпізнавання вхідних даних. Подано детальний огляд результатів використання розробленого алгоритму для програмних реалізацій на основі двох сучасних парадигм реалізації багатопотокових обчислень – акторної моделі [14, 15] із використанням об'єктно-функціональної мови програмування Scala [16, 17] і методу MapReduce [18] із застосуванням мови програмування Java [19]. Усі експерименти використовували стандартні тестові набори даних [20].

Декомпозиція та навчання карт Кохонена

Найпоширенішим методом декомпозиції карт Кохонена з метою реалізації навчання із використанням багатопроцесорних систем є MP (англ. map-paritioning) та DT (англ. data-partitioning) [10]. Суть методу MP полягає у поділі елементів карти між декількома потоками, які відповідають за пошук елемента найкращого наближення і модифікацію вагових векторів у виділеній групі елементів. Метод DT передбачає розподіл даних між кількома потоками і застосування batch-методу навчання карт Кохонена. Це є істотною незручністю, оскільки, з одного боку, може виникнути проблема з потребою завантаження усіх даних для навчання, а з іншого – використовуючи batch-метод навчання карт Кохонена, важче досягти "хорошого" топологічного впорядкування карти [1]. Головним недоліком методу MP вважають потребу синхронізації між потоками після завершення пошуку елемента-переможця та після модифікації вагових векторів. Автори праці [11] стверджують, що ефективно реалізовувати цей метод можна лише з використанням архітектури SMP (Symmetric multiprocessing), і пропонують альтернативний підхід. З огляду на все більше поширення комп'ютерних систем із багатоядерними процесорами, які належать до SMP-архітектури, є доцільним розроблення модифікованого алгоритму навчання декомпонованих карт, що базується на методі MP.

Доцільно нагадати, що самоорганізовною картою Кохонена називають нелінійне, впорядковане і гладке відображення $\Phi : X \to M$, де X – неперервний лінійний метричний простір великої розмірності, M – дискретний метричний двовимірний простір із топологією, яка визначається розташуванням його елементів у вузлах ґратки. На рис. 1 зображено дві найпоширеніші схеми формування ґратки елементів – стільникову і прямокутну.



а *б*

Рис. 1. Структури ґраток: стільникова (а) і прямокутна (б)

На рис. 1 елементи зображені кружальцями з відповідним координатами. Сірим кольором виділено елементи, які перебувають у безпосередньому сусідстві з елементом із координатами $(4,4)$. Зауважимо, що значення координат не є суттєвим – головне, щоб вони відповідали

регулярній структурі ґратки. Елементи вихідного простору позначають $m_k \in M$, $k = \overline{1, |M|}$, $|M|$ – кількість елементів карти. Із кожним елементом m_k асоційований вектор $w_k \in W \subset X$ ($|W| = |M|$), який називають *ваговим*. Конфігурація ґратки визначається взаємним розміщенням елементів $m_k \in M$, які мають цілочислові координати й утворюють евклідів простір. Отже, кожен k-й елемент ґратки ототожнено з двома величинами – ваговим вектором w_k та цілочисловими координатами (i, j), які відповідають структурі та позиції елемента у ґратці. Позначимо цей факт $m_k = \langle w_k, (i, j) \rangle$, $k = \overline{1, |M|}$. Декомпозиція ґратки полягає у поділі множини елементів карти $m_k \in M$ на підмножини однакової потужності. Це уможливлює пошук елементів-переможців і адаптацію вагових векторів паралельно для кожної підмножини.

На рис. 2 зображена блок-схема алгоритму навчання декомпонованої карти. Головною відмінністю запропонованого алгоритму від MP є повторне використання потоків відразу після того, як вони звільнилися, без очікування решти потоків. Такий підхід дещо скоротив час перебування потоків у блоці синхронізації.

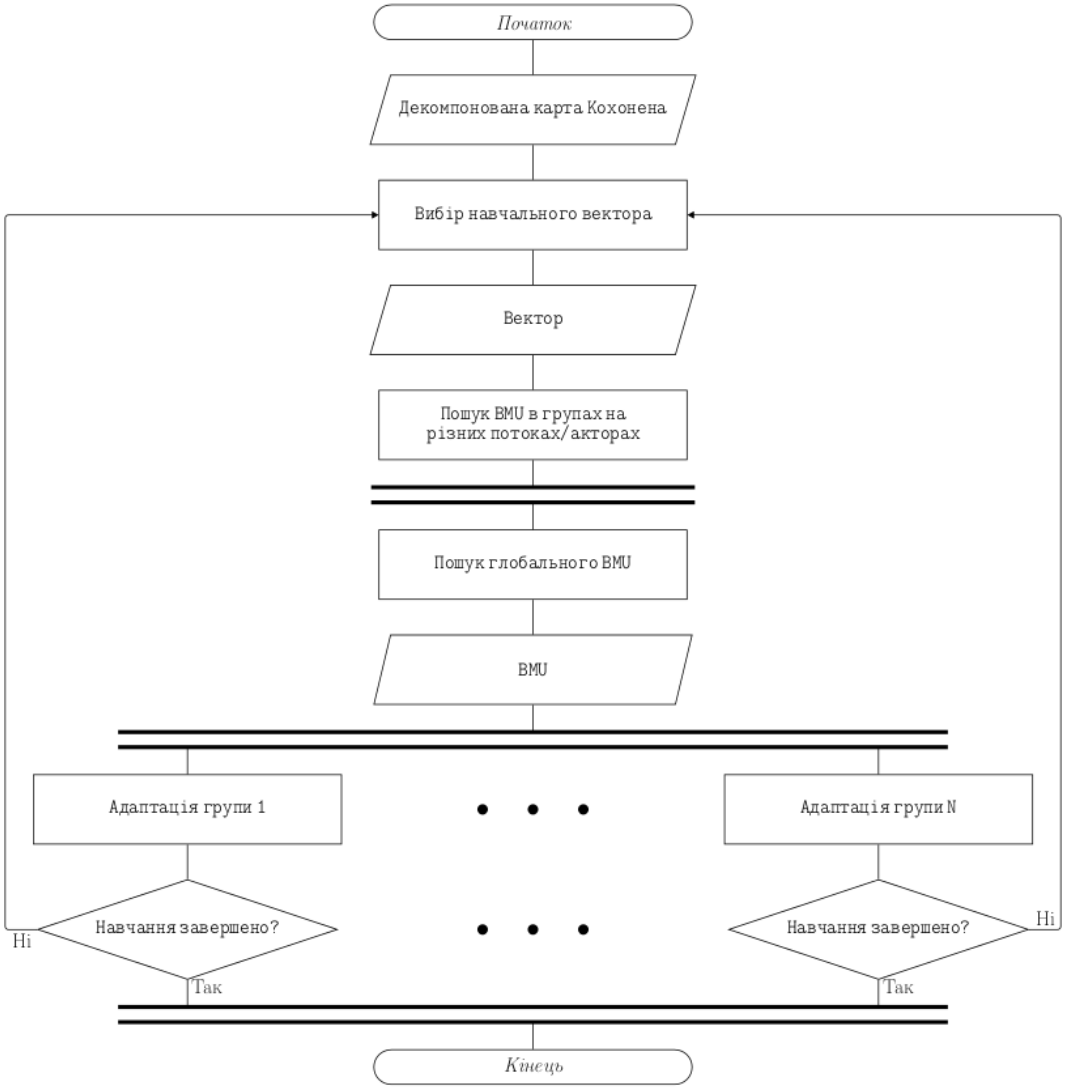


Рис. 2. Блок-схема навчального алгоритму декомпонованої карти

Сам процес декомпозиції карти є тривіальним поділом елементів карти на таку кількість груп, яку бажано опрацьовувати паралельно. Зазначимо, що запропонований метод не залежить від

топології ґратки, оскільки пошук елемента-переможця не пов’язаний з нею, а для модифікації вагових векторів треба знати його розташування і вектор елемента-переможця. Детальніше методи навчання карт Кохонена описано у [1–9].

Порівняння ефективності MapReduce й акторної моделі

Використання акторної моделі програмування багатопотокових програм є прозорішим для реалізації, оскільки не потрібна явна синхронізація потоків. Додатково конкретна бібліотека чи мова, яка реалізує підтримку акторної моделі, відповідає за вибір оптимальної кількості потоків, необхідних для виконання обчислень. У цьому дослідженні використано стандартну бібліотеку акторів мови Scala [14, 15]. Алгоритм реалізації багатопотокових обчислень на основі парадигми MapReduce реалізовано самостійно на основі використання сучасних засобів стандартної потокової бібліотеки мови Java 1.6 [19]. Як альтернативою можна скористатися готовими рішеннями, такими як Hadoop. Усі експерименти здійснено на комп’ютері з процесором Intel Xeon E5504, який має чотири ядра із частотою 2Ghz кожне. Ідентичні тести виконувались багаторазово для використання середнього часу виконання для усунення випадково неуспішних чи успішних тестів.

Подані тут результати експериментів отримано для тестових еталонних наборів даних Lung Cancer та Iris Plants. Цей вибір зумовлений значною різницею у розмірності вхідного простору – 56 для Lung Cancer та 4 для Iris Plants. Очікується, що навчання на наборі даних Lung Cancer забезпечить ефективніше використання обчислювальних ресурсів у разі багатопотокового виконання.

Результат для набору даних Lung Cancer

На рис. 3 і 4 подано графіки, які показують час виконання десяти експериментів для десяти різних конфігурацій декомпозиції карти Кохонена. Кількість груп декомпозиції було збільшено від однієї, що відповідає послідовному навчанню, до десяти.

В акторній моделі кількість системних потоків контролювалася реалізацією бібліотеки. Для MapReduce встановлена кількість груп визначала кількість системних потоків. Порівнюючи графіки на рис. 3 та 4, можна переконатися, що для MapReduce істотно збільшується час виконання, якщо кількість груп перевищує кількість ядер процесора (чотири). Водночас, у акторної моделі ця різниця не є настільки суттєвою.

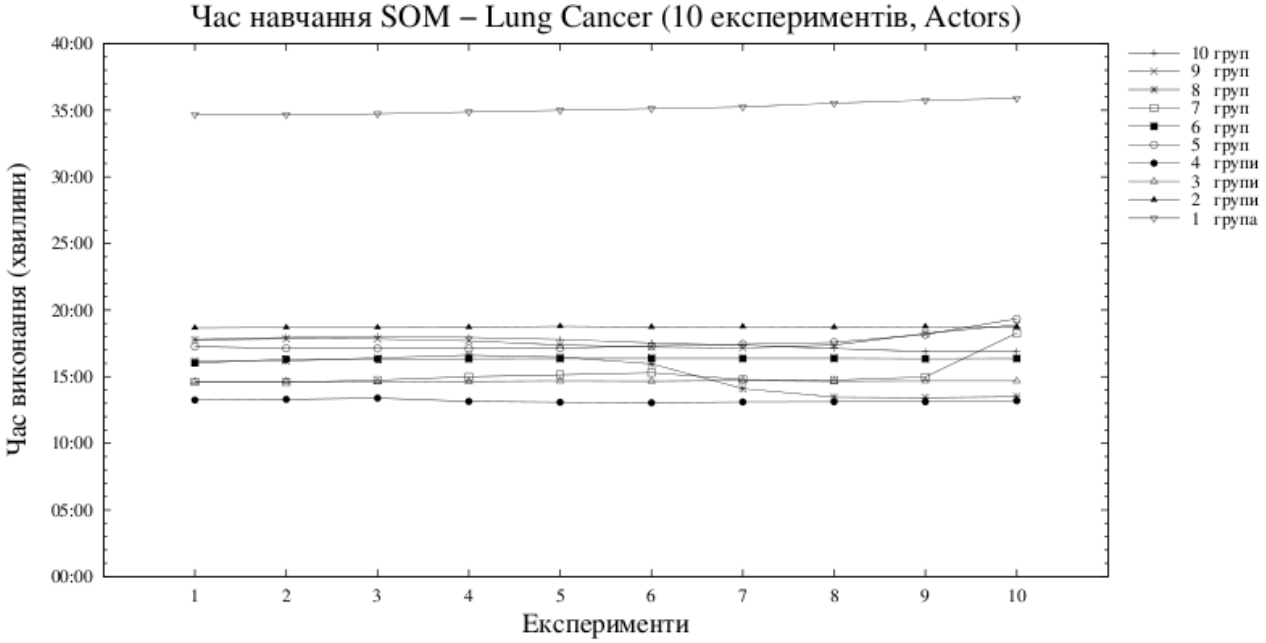


Рис. 3. Специфіка виконання експериментів на наборі даних Lung Cancer: Актори

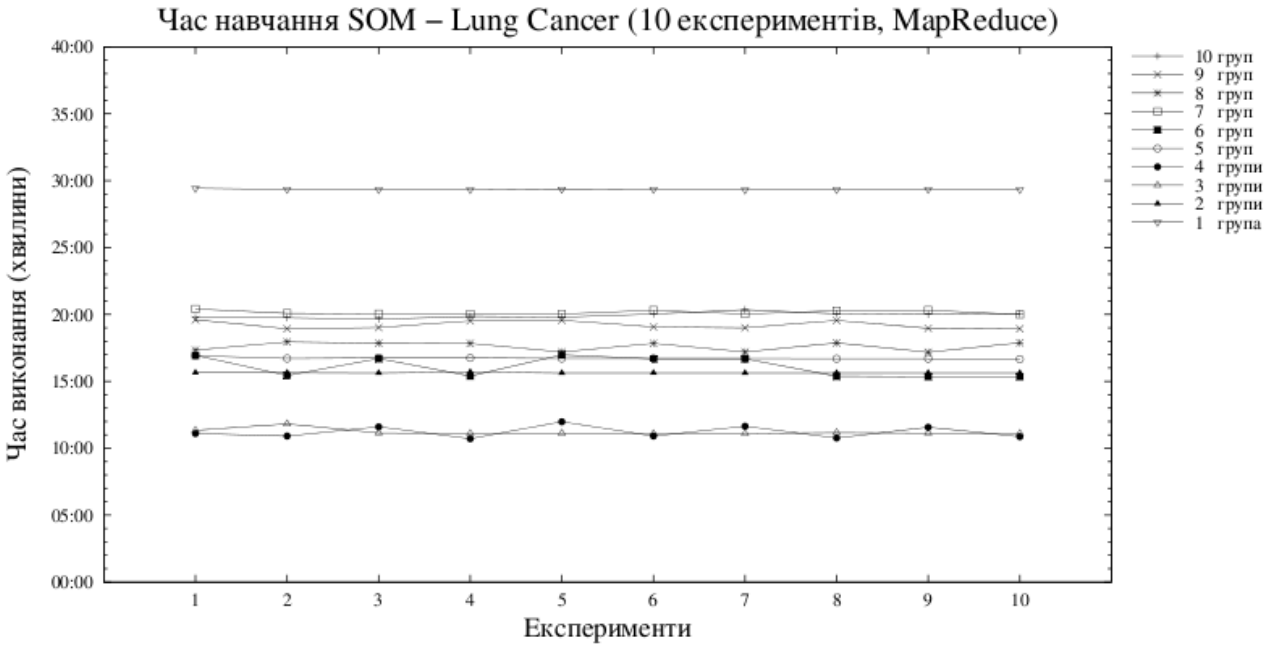


Рис. 4. Деталі виконання експериментів
на наборі даних Lung Cancer: MapReduce

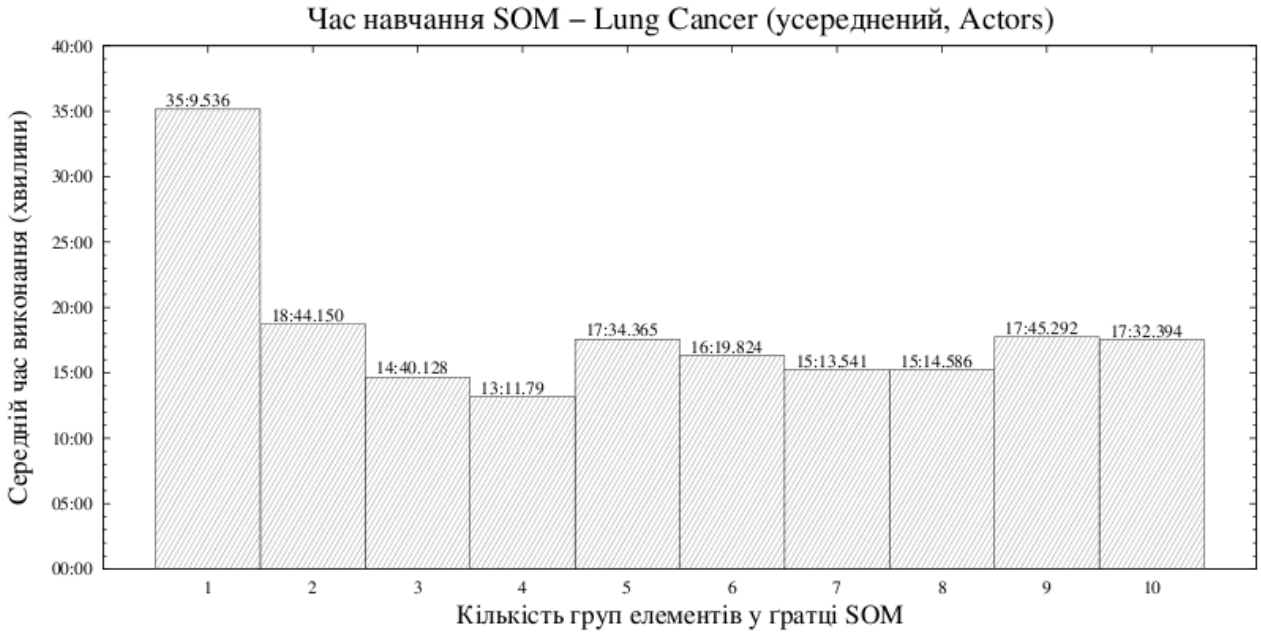


Рис. 5. Середній час виконання експериментів
на наборі даних Lung Cancer: Актори

Однозначним є приріст у швидкодії зі збільшенням груп елементів карти. На рис. 5 та 6 подано гістограми усередненого часу виконання експериментів у разі декомпозиції карт на різні кількості груп.

Зауважимо, що реалізація навчання декомпонованої карти на основі MapReduce забезпечила вищу продуктивність порівняно із акторною моделлю, якщо кількість груп не перевищувала кількості ядер процесора. Для реалізації на основі MapReduce один системний потік використовувався для синхронізації решти. Саме тому не відбулося істотних змін між трьома та чотирма групами.

Для того, щоб спростити порівняння ефективності навчання для різних парадигм реалізації багатопотоковості, на рис. 7 подано графік, який поєднує усереднені результати.

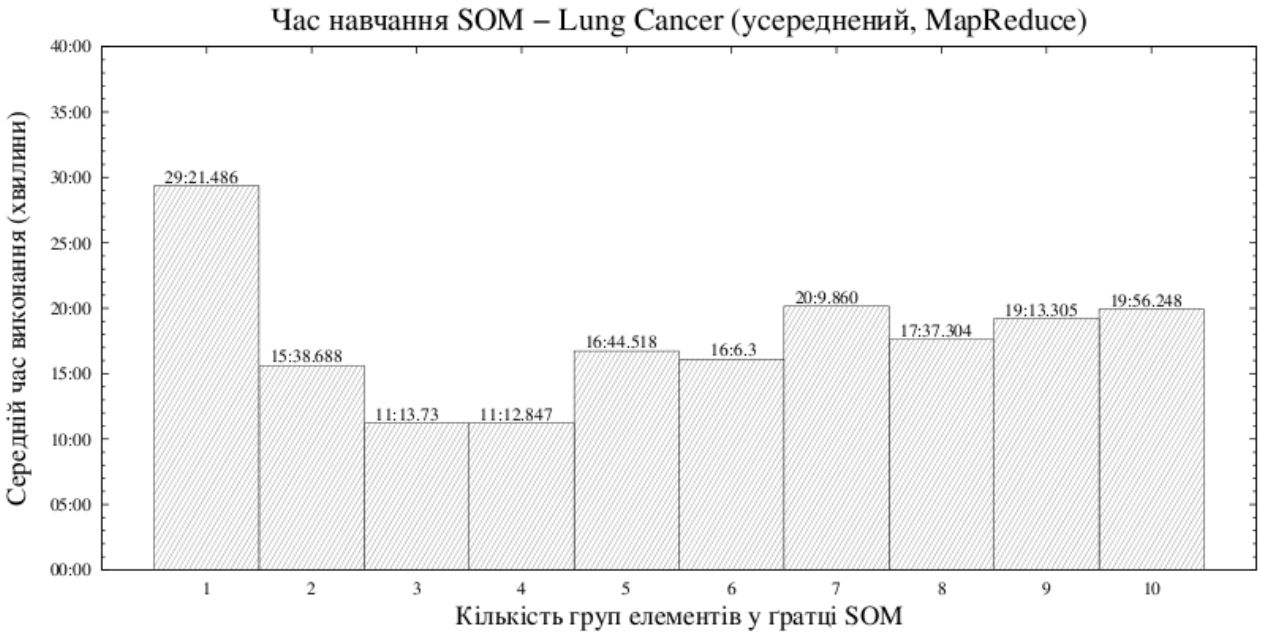


Рис. 6. Середній час виконання експериментів на наборі даних Lung Cancer: MapReduce

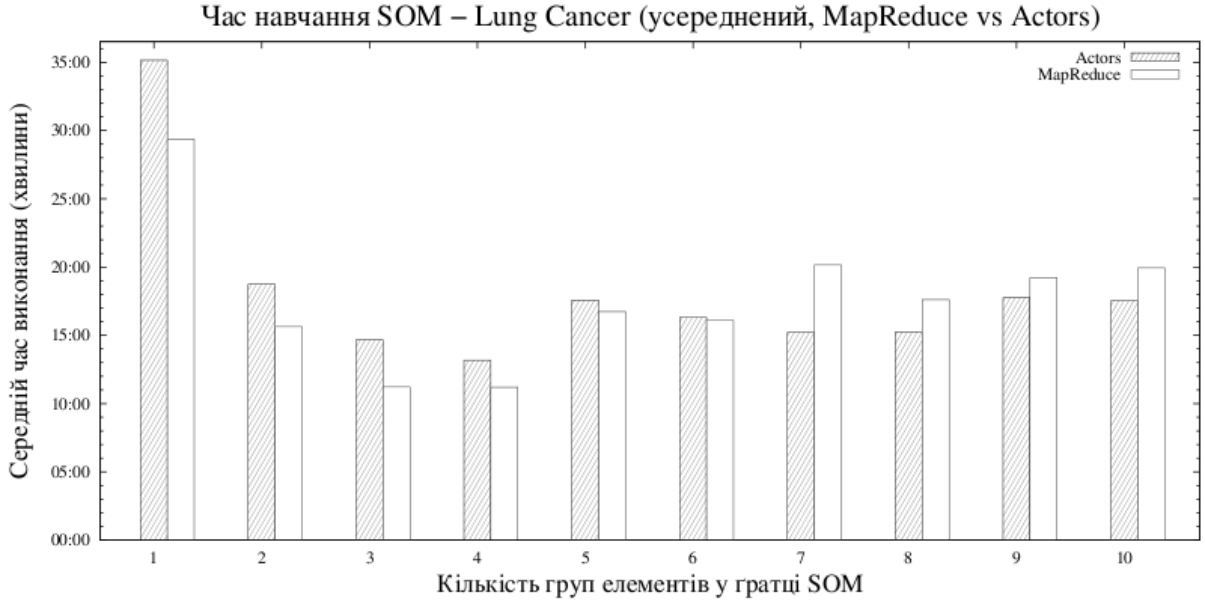


Рис. 7. Порівняння середнього часу виконання експериментів на наборі даних Lung Cancer для акторної моделі та MapReduce

Результат для набору даних Iris Plants

На рис. 8 та 9 показано графіки залежності часу тривалості навчання від кількості груп декомпозиції карти для набору даних Iris Plants. Так само, як і у попередньому випадку, кількість груп декомпозиції збільшено від однієї до десяти. Із графіків на рис. 8 та 9 видно, що залежність часу виконання навчання від кількості груп є істотнішою, ніж для набору даних Lung Cancer. Навіть у випадку акторної моделі розкид значень часу виконання є більшим, ніж у попередньому випадку. Така ситуація зумовлена менш інтенсивним використанням процесора у випадку даних Iris Plants з огляду на значно меншу розмірність вхідного простору. Отже, більше часу витрачалося на синхронізацію потоків, що відповідають за навчання.

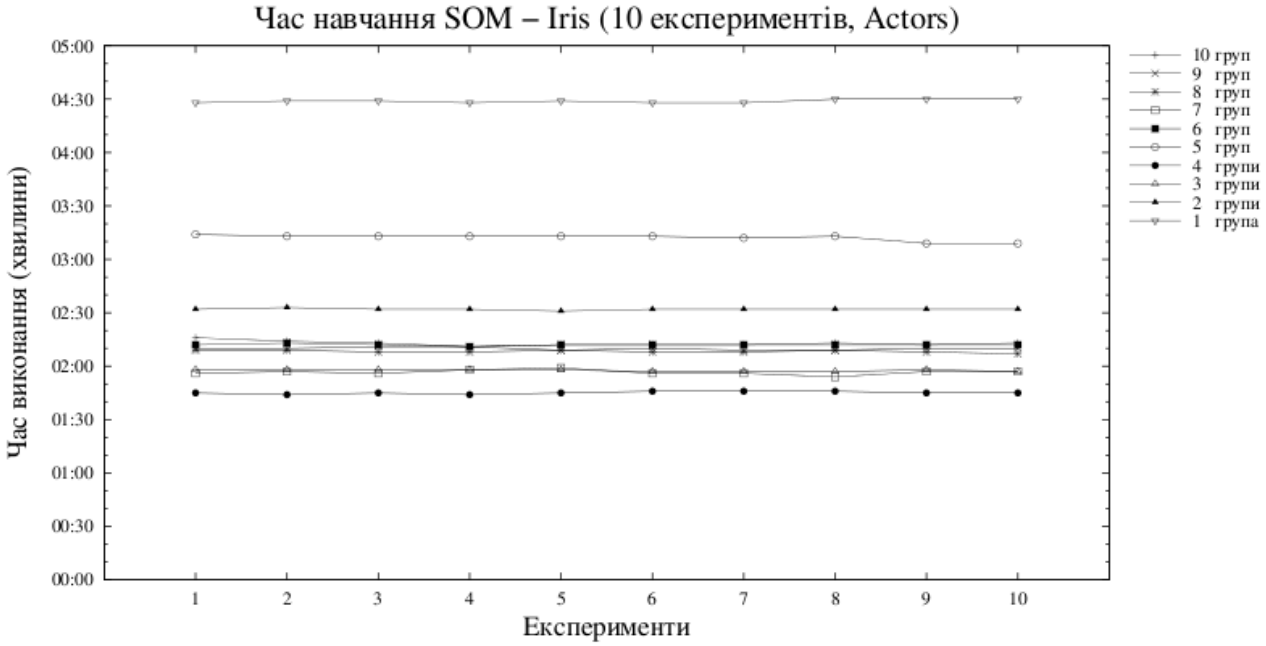


Рис. 8. Особливості виконання експериментів на наборі даних Iris Plants: Актори

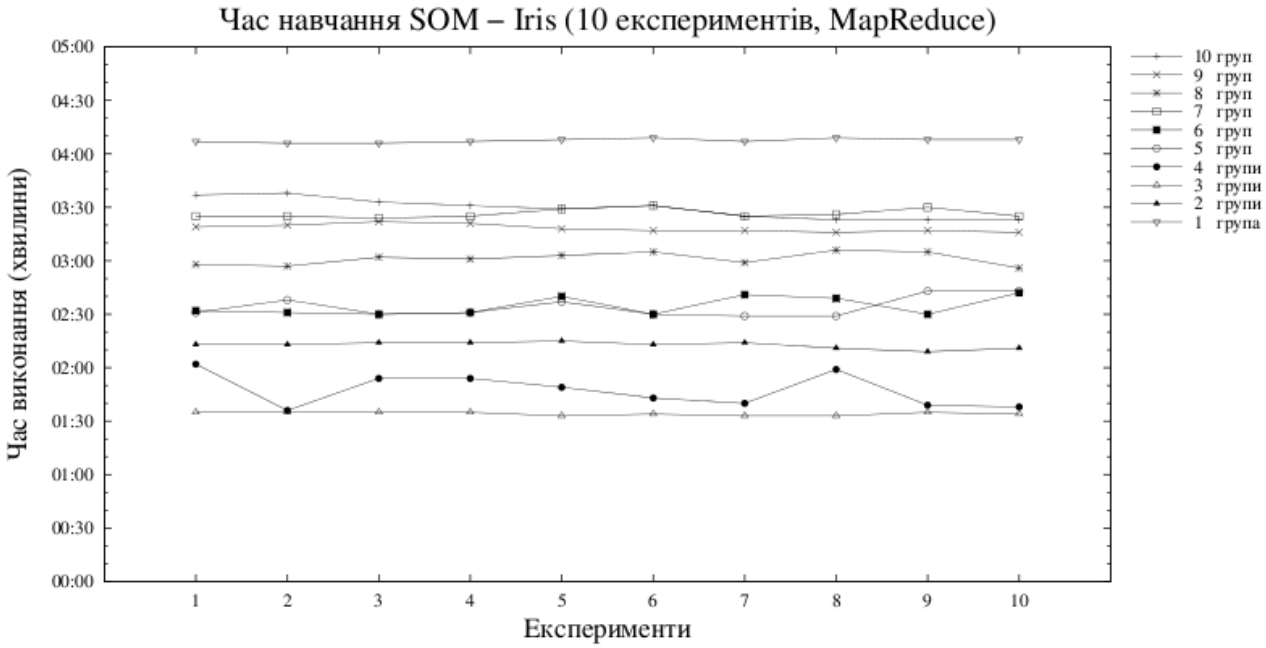


Рис. 9. Особливості виконання експериментів на наборі даних Iris Plants: MapReduce

Схоже, як для випадку даних Lung Cancer, приріст у швидкодії у разі зростання груп елементів карти, більший у MapReduce. Такий виграш є мінімальним, і вже для чотирьох груп акторна модель забезпечила кращий результат. На рис. 10 та 11 подано гістограму усередненого часу виконання експериментів для декомпозиції карт на різні кількості груп.

Щоб спростити порівняння ефективності навчання для різних парадигм реалізації багатопотоковості, на рис. 12 подано графік, який поєднує усереднені результати. Легко переконатися, що акторна парадигма розпаралелення обчислень забезпечила несуттєво гірший результат порівняно із MapReduce. Водночас, ця парадигма уможливлює істотно простіший спосіб програмної реалізації алгоритму навчання декомпонованої карти Кохонена, що важливо враховувати, створюючи складні програмні комплекси.

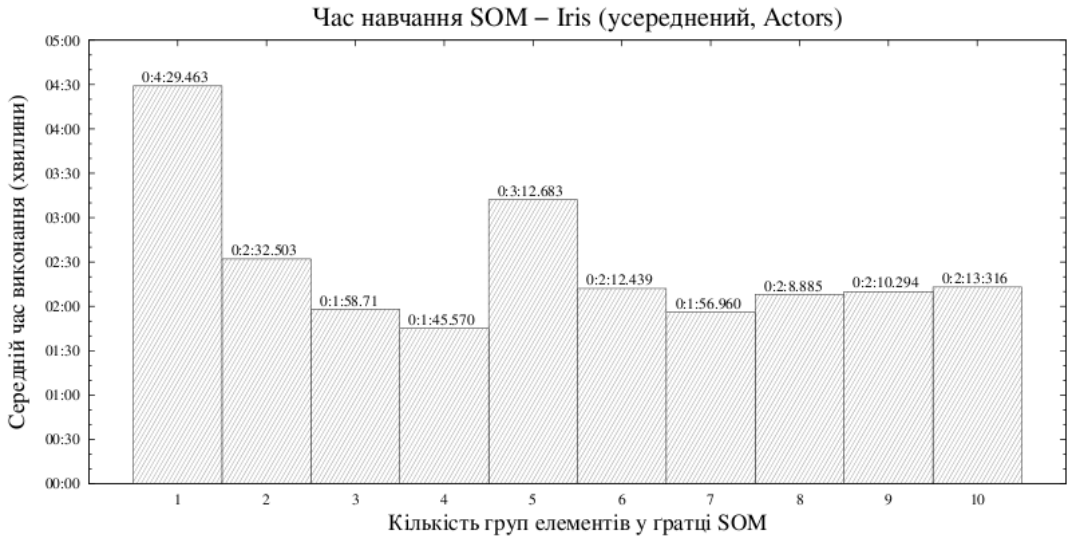


Рис. 10. Середній час виконання експериментів на наборі даних Iris Plants: Актори

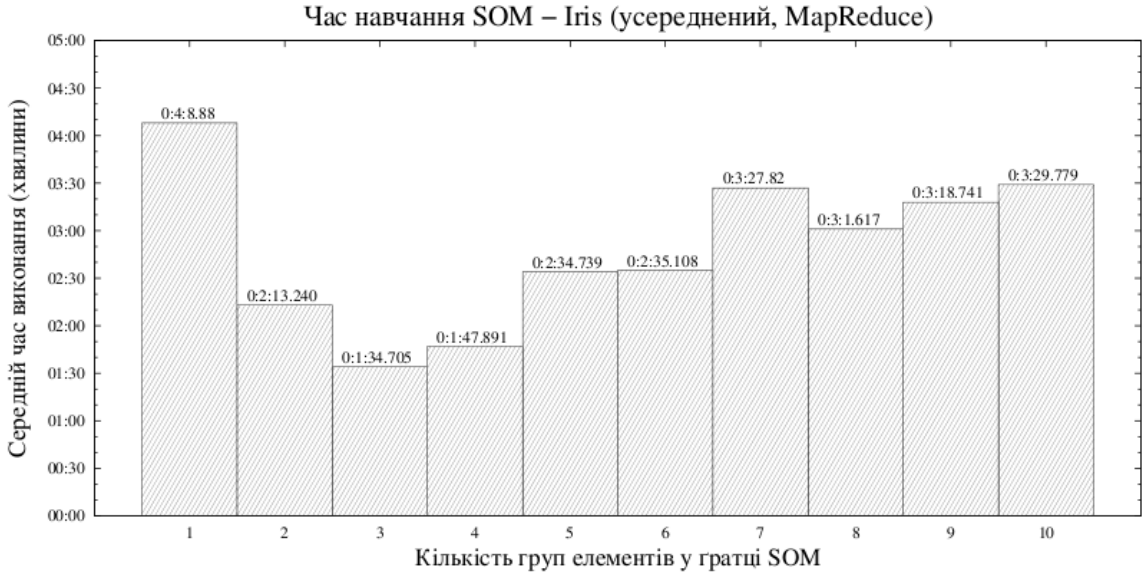


Рис. 11. Середній час виконання експериментів на наборі даних Iris Plants : MapReduce

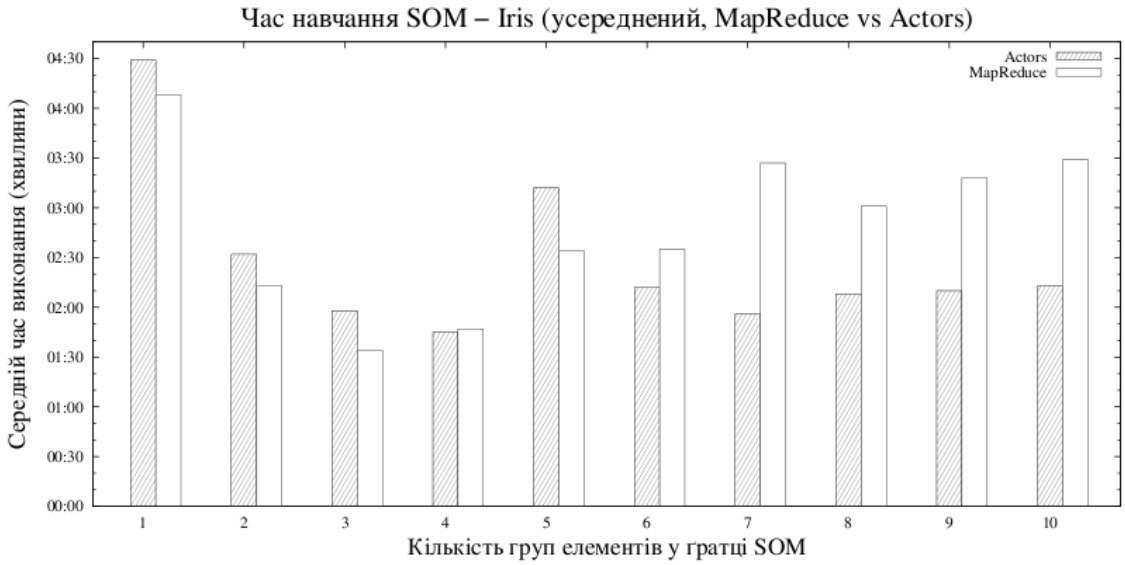


Рис. 12. Порівняння середнього часу виконання експериментів на наборі даних Iris Plants для акторної моделі та MapReduce

Висновки

У дослідженні здійснено порівняльний аналіз двох найпоширеніших парадигм реалізації багатопотокового програмного забезпечення – MapReduce та акторної моделі. Це дослідження дало змогу виявити простоту використання моделей і проаналізувати приріст швидкодії процесів навчання. Із поданих результатів можна зробити висновок, що акторна модель є стійкішою до налаштувань і середовища виконання. Водночас, для даних великої розмірності істотно ефективнішим виявився алгоритм на основі MapReduce. Важливим при цьому є аналіз закону Амдагла (англ. Amdahl's law) стосовно реалізованих алгоритмів (див. рис. 13).

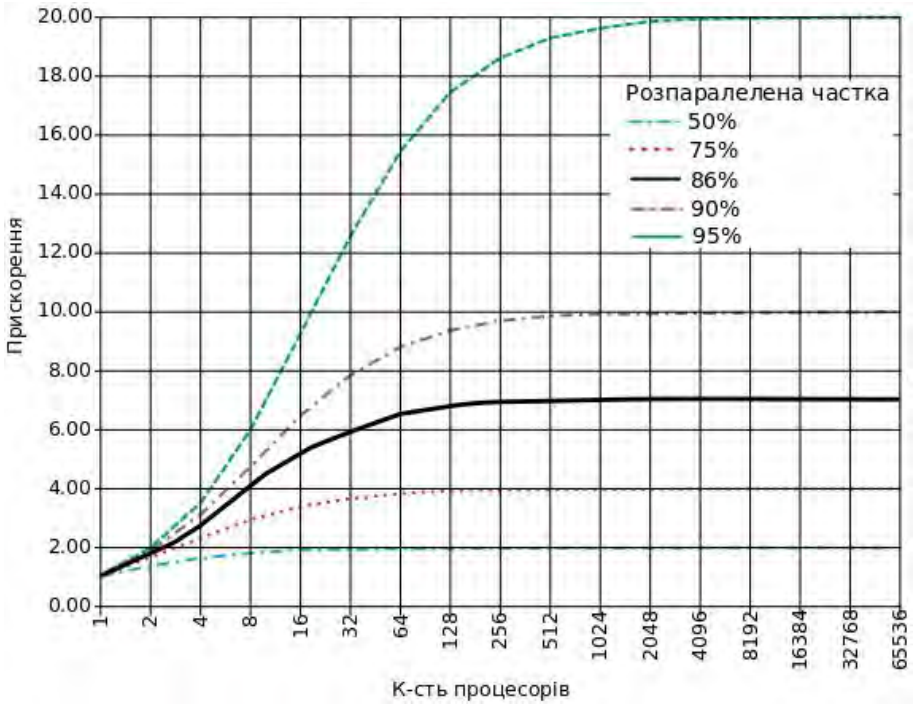


Рис. 13. Графіки залежності прискорення програми від частки розпаралелення і кількості процесорів

Цей закон стверджує, що якщо P є часткою програми, яку можна паралелізувати, а, відповідно, $(1-P)$ є часткою, яку неможливо паралелізувати (залишається послідовною), то максимальне прискорення, яке можна отримати, використовуючи N процесорів, обчислюється співвідношенням $1/\left((1-P)+P/N\right)$. Емпірично значення P можна знайти за співвідношенням $P_{est}=\left(1/SU-1\right)/\left(1/NP-1\right)$, де SU – емпірично встановлений коефіцієнт прискорення на NP процесорах. Для виконаних експериментів отримано $P_{est}=\left(1/2.8-1\right)/\left(1/4-1\right)\approx 0.86$. Отже, максимальний приріст швидкодії за умови нескінченної кількості процесорів становить $\lim\limits_{N\to\infty}\left(1/\left((1-0.86)+0.86/N\right)\right)\approx 7.14$. За графіком на рис. 13, який відповідає частці 86 %, можна визначити загальну кількість ядер, які доцільно використовувати для моделювання даних на основі карт Кохонена із розробленим алгоритмом навчання.

1. Kohonen T. Self-Organizing Maps / T. Kohonen. – Springer, 2001. – 521 p. 2. Ritter H. Neural Computation and Self-Organizing Maps: An introduction / H. Ritter, T. Martinetz, K. Schulten. – Addison Wasley, 1992. – 350 p. 3. Годыч О.В. Исследование эффективности алгоритмов обучения нейросетей Кохонена / О. В. Годыч, В. В. Пасичник, Ю.В. Никольский, Ю.Н. Щербина // Управляющие системы и машины. – К., 2006. – № 2. – С. 63–80. 4. Hodych O. Synthesis of self-organizing map and feedforward neural network for better forecasting / O. Hodych, Y. Shcherbyna, M.

Zylan // International Journal of Computing. – 2004. – Vol. 3, no. 3. – P. 68–75. 5. Годич О. Застосування штучної нейронної мережі типу SOM для розв'язування задачі діагностування / О. Годич, Ю. Нікольський, Ю. Щербина // Вісник Національного ун-ту "Львівська політехніка", Інформаційні системи та мережі. – 2002. – № 464. – С. 31–43. 6. Hodych O. SOM-based dynamic image segmentation for sign language training simulator / O. Hodych, K. Hushchyn, I. Nikolski, V. Pasichnyk, Y. Shcherbyna // Information Systems: Modeling, Development, and Integration / Ed. by W. van der Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski. – Berlin: Springer Berlin Heidelberg, 2009. – Vol. 20 of Lecture Notes in Business Information Processing. – P. 29–40. 7. Годич О. Українська жестова мова: комп'ютерно-лінгвістичний аспект: моногр. / О. Годич, М. Давидов, Ю. Нікольський, В. Пасічник, Ю. Щербина. – Львів: Літературна агенція "Піраміда", 2009. – 253 с. 8. Годич О. Навчання SOM методом нейронної міграції / О. Годич // Вісник Національного ун-ту "Львівська політехніка", Інформаційні системи та мережі. – 2004. – № 519. – С. 55–72. 9. Годич О. Кластеризація даних нейромережею ADD / О. Годич // Вісник Національного ун-ту "Львівська політехніка", Інформаційні системи та мережі. – 2005. – № 549. – С. 54–68. 10. Rauber A. parSOM: A Parallel Implementation of the Self-Organizing Map Exploiting Cache Effects: Making the SOM Fit for Interactive High-Performance Data Analysis / A. Rauber, P. Tomsich, D. Merkl // Neural Networks, IEEE - INNS - ENNS International Joint Conference on. – 2000. – Vol. 6. – P. 61–77. 11. García C. A speculative parallel algorithm for self-organizing maps / C. García M. Prieto, A. D. Pascual-Montano // PARCO. – 2005. – P. 615–622. 12. Hämäläinen Timo D. Parallel implementation of self-organizing maps / Timo D. Hämäläinen // Self-Organizing Neural Networks: Recent Advances and Applications/ Ed. by Udo Seiffert, Lakhmi C. Jain – Springer-Verlag New York, Inc., 2002. – P. 245–278. 13. Lawrence R. D. A Scalable Parallel Algorithm for Self-Organizing Maps with Applications to Sparse Data Mining Problmes / R. D. Lawrence, G.S. Almasi, H.E. Rushmeier – 1998. – 27 p. – [Електронний ресурс]: http://www.research.ibm.com/dar/papers/pdf/scalableSOM.pdf 14. Odersky M. Event-based programming without inversion of control / M. Odersky // In Proc. Joint Modular Languages Conference (2006), Springer LNCS. – Springer, 2006. – P. 4–22. 15. Odersky M. Actors that unify threads and events / M. Odersky // In International Conference on Coordination Models and Languages, LNCS. – Springer-Verlang, 2007. – P. 171–190. 16. Odersky M. Programming in Scala: A Comprehensive Step-by-step Guide / M. Odersky, L. Spoon, B. Venners. – 1st edition. – Artima Inc, 2008. – 776 p. 17. Wampler D. Programming Scala: Scalability = Functional Programming+ Objects / D. Wampler, A. Payne. – 1st edition. – O'Reilly Media, 2009. – 448 p. 18. Dean J. Mapreduce: simplified data processing on large clusters / J. Dean, S. Ghemawat // Commun. ACM. – 2008. – Vol. 51, no. 1. – P. 107–113. 19. Goetz B. Java Concurrency in Practice / B. Goetz, T. Peierls, J. Bloch. – Addison-Wesley Professional, 2006. – 384 p. 20. Asuncion A. UCI machine learning repository. – 2007. – [Електронний ресурс]: http://archive.ics.uci.edu/ml.