УДК 004.051
І. В. Турченко, канд. техн. наук

КОРЕКТНЕ ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ РОЗПАРАЛЕЛЕННЯ ПРИ ВИКОРИСТАННІ БІБЛІОТЕКИ ВІЗУАЛІЗАЦІЇ MPE

Анотація. *Розглянуто грубозернистий паралельний алгоритм навчання модульних нейронних мереж з динамічним розподілом на процесори паралельного комп'ютера. Розпаралелення проведено на високопродуктивному комп'ютері Origin 300 з використанням технології MPI (Message Passing Interface). При оцінці ефективності даного паралельного алгоритму шляхом візуалізації процесу розпаралелення за допомогою бібліотеки MPE (MPI Parallel Environment) виявлено недолік бібліотеки, пов'язаний з обчисленням астрономічного, а не процесорного часу виконання паралельної програми. Здійснено відповідну модифікацію бібліотеки візуалізації MPE, що усуває даний недолік і дає змогу коректно визначати ефективність розпаралелення паралельної програми.*

Ключові слова: *оцінка ефективності, процесорний час, динамічний розподіл, грубозернисте розпаралелення, нейронні мережі, MPI, MPE*

I. V. Turchenko, PhD.

CORECT DEFINITION OF PARALLELIZATION EFFICIENCY USING MPE VISUALIZATION LIBRARY

Abstract. *The coarse-grain parallel algorithm of modular neural networks training with dynamic mapping onto processors of parallel computer is described in this paper. Parallelization of the algorithm is done on parallel computer Origin 300 using MPI (Message Passing Interface) technology. Efficiency estimation of this parallel algorithm using MPE (MPI Parallel Environment) library has showed its disadvantage connected with calculation of a wall, not a processor computational time of parallel routine. The appropriate modification of the MPE library allows correctly define of a parallelization efficiency of a parallel routine.*

Keywords: *efficiency analysis, processor time, dynamic mapping, coarse-grain parallelization, neural networks, MPI, MPE*

И. В. Турченко, канд. техн. наук

КОРРЕКТНОЕ ОПРЕДЕЛЕНИЕ ЭФФЕКТИВНОСТИ РАСПАРАЛЛЕЛИВАНИЯ ПРИ ИСПОЛЬЗОВАНИИ БИБЛИОТЕКИ ВИЗУАЛИЗАЦИИ MPE

Аннотация. *Рассмотрено крупнозернистый параллельный алгоритм обучения модульных нейронных сетей с динамическим распределением на процессоры параллельного компьютера. Распараллеливание выполнено на высокопроизводительном компьютере Origin 300 с использованием технологии MPI (Message Passing Interface). При оценке эффективности данного параллельного алгоритма путем визуализации процесса распараллеливания с помощью библиотеки MPE (MPI Parallel Environment) обнаружен недостаток библиотеки, связанный с вычислением астрономического, а не процессорного времени выполнения параллельной программы. Выполнено необходимую модификацию библиотеки визуализации MPE, которая устраняет этот недостаток и позволяет корректно определять эффективность распараллеливания параллельной программы.*

Ключевые слова: *оценка эффективности, процессорное время, динамическое распределение, крупнозернистое распределение, нейронные сети, MPI, MPE*

Вступ. Паралельні обчислення мають велике значення у багатьох сферах комп'ютерних застосувань. Сучасні персональні комп'ютери з багатоядерними процесорами обумовлюють розробку ефективних паралельних програм. Концепція паралельної обробки полягає в розподілі завдань, що виконувалася послідовно, на багато процесорів з метою зменшення загального часу виконання. Проте розподіл завдань на багато процесорів може бути неефективним через додаткові обчислювальні затрати на синхронізацію, комунікацію та нерівномірне завантаження процесорів/ядер. Як результат, питання ефективного розпаралелення прикладних задач все ще залишаються актуальними [1].

Вказані питання доцільно розглядати в кожному конкретному випадку окремо, тому що прикладна задача, для якої розробляється алгоритм розпаралелення, накладає суттєві обмеження на виконання цього паралельного алгоритму, так само, як і архітектура використовуваної паралельної обчислювальної системи. Відтак питання ефективного розпаралелення включають в себе визначення ефективності паралельних

алгоритмів, паралельних архітектур, паралельних мов програмування та аналіз продуктивності паралельних програм [2–3].

Аналіз продуктивності паралельних програм зручно здійснювати за допомогою різноманітних пакетів візуалізації виконання, що спрощують складний процес знаходження «вузьких місць» паралельної програми. Відомими технологіями та підходами, що досліджуються зараз у цій області, є NetVampire [4], Prober [5], TAU [6] та інші [7]. Такі пакети, як NetVampire і TAU, є частково на платній основі, крім того, їх використання доцільне для аналізу продуктивності складних паралельних програм. На відміну від вищеописаних пакетів, бібліотека MPE [8] є стандартним засобом, її легко застосовувати, вона «по-замовчуванню» входить в MPI (Message Passing Interface) – дистрибутив MPICH [9].

Метою даної статті є оцінка ефективності паралельної програми шляхом візуалізації процесу розпаралелення на прикладі паралельного алгоритму динамічного розподілу набору інтегруючих історичні дані нейронних мереж [10] за допомогою бібліотеки MPE. Метод інтеграції історичних даних є складовою частиною процесу прогнозування дрейфу сенсорів за допомогою нейронних мереж (НМ). Перевагою запропонованого підходу є те, що на основі малої кількості даних для навчання НМ він дає змогу підвищити точність прогнозу дрейфу сенсорів у 3–5 разів при збільшенні міжкалібрувального інтервалу у 6–12 разів [11]. Проте цей метод вимагає значних обчислювальних ресурсів для навчання інтегруючих історичні дані нейронних мереж (ІІДНМ), зокрема близько 40 хв для одного каналу збору даних.

В статті описано розробку динамічного алгоритму розпаралелення ІІДНМ, наведено результати експериментальних досліджень алгоритму, досліджуються підходи до оцінки ефективності паралельного алгоритму та наводяться результати відповідних експериментальних досліджень.

Грубозернистий паралельний алгоритм навчання ІІДНМ з динамічним розподілом. Як показано в роботі [10], для інтеграції історичних даних використовуються M інтегруючих історичні дані нейронних мереж ІІДНМ. В цих роботах також показано, що доцільно вибрати грубозернистий рівень розпаралелення, коли кожен модуль з M нейронних мереж виконується на окремому процесорі паралельного комп'ютера. Паралельний алгоритм побудовано за схемою "централізованого" планування (рис. 1). Функції розподілу завдань між процесорами виконує головний процесор *Root*, а обчислення ІІДНМ виконують інші наявні процесори паралельної машини, визначені користувачем при завантаженні паралельної програми на виконання. Процесори, що здійснюють обчислення ІІДНМ, при описі алгоритму нижче названі "обчислюючими" процесорами. Організація розподілу здійснена за допомогою функцій передавання/приймання повідомлень бібліотеки організації паралельних обчислень MPI *MPI_Send()* та *MPI_Recv()* [12].

Послідовна частина алгоритму включає в себе виконання двох операцій: зчитування вхідних файлів з даними про сенсорний дрейф та визначення порядкового номера ІІДНМ в залежності від її вхідних даних. Проте на рис. 1 ці процедури показано окремо в складі кожного алгоритму з метою спрощення їх опису, паралельні частини алгоритмів починаються після виконання інструкції *MPI_Init()*. Індекс $<cn>$ служить порядковим номером поточної ІІДНМ. Динамічний розподіл полягає в передаванні та прийманні цього номера $<cn>$, тому що кожен процесор, як головний, так і обчислюючий, має всі відповідні дані про ІІДНМ у своїй локальній пам'яті. Це значно спрощує організацію взаємодії між паралельно працюючими процесорами шляхом обміну повідомленнями, які містять лише один елемент даних – індекс $<cn>$.

Головний процесор *Root* (рис. 1, а) спочатку здійснює початковий розподіл M ІІДНМ на M процесорах, тому що кількість паралельних процесорів, на яких виконується програма, відома відразу після її завантаження на виконання. Потім *Root* виконує головний цикл розподільчих функцій. Умовою циклу є перевірка, чи всі ІІДНМ вже розподілено на виконання на

обчислюючих процесорах. У випадку, якщо всі ІІДНМ розподілено на виконання, то *Root* очікує повідомлення від будь-якого обчислюючого процесора. Це реалізовано за допомогою функції *MPI_Recv()* з параметром *MPI_ANY_SOURCE*. При фізичному отриманні будь-якого вхідного повідомлення *Root* визначає номер процесора $<cp>$ (процесор-відправник повідомлення) та порядковий номер ІІДНМ $<cn>$ (цей номер міститься в повідомленні), що закінчила навчатись на даному процесорові. Також в цьому повідомленні *Root* приймає результат інтеграції історичних даних, обчислений ІІДНМ з номером $<cn>$ на процесорі $<cp>$ і зберігає цей результат у відповідній комірці матриці результатів $ComM$.

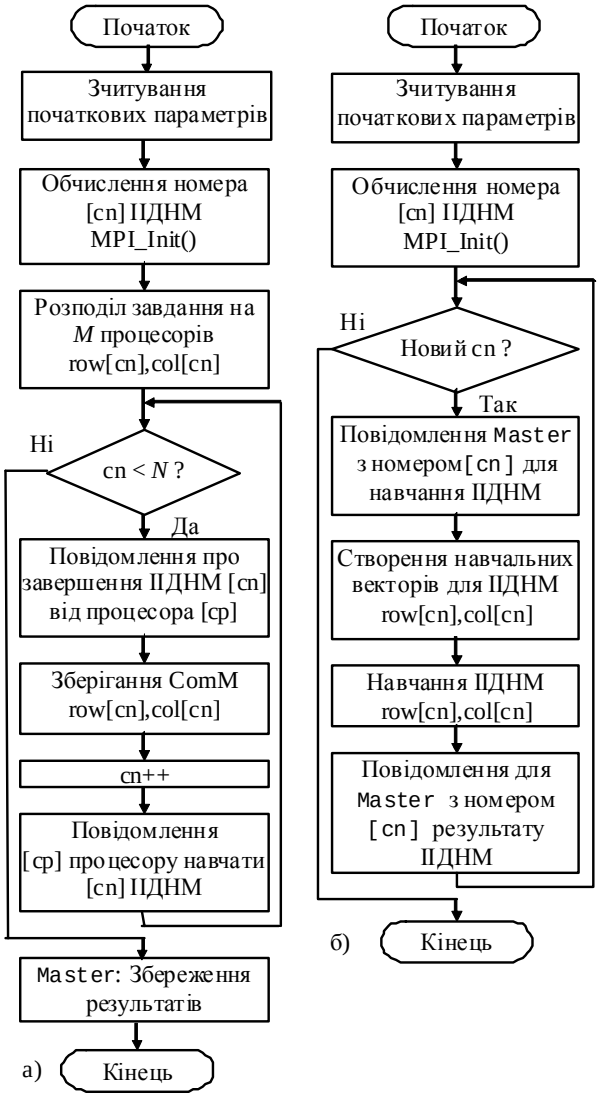
Далі головний процесор *Root* вибирає ІІДНМ з наступним порядковим номером ($cn++$) і посилає повідомлення навчати цю ІІДНМ процесору $<cp>$. При розподілі всіх ІІДНМ та отриманні повідомлень про закінчення навчання від усіх обчислюючих процесорів, *Root* зберігає отриману матрицю результатів $ComM$, що знаходиться в його локальній пам'яті, у вихідному файлі результатів роботи паралельного алгоритму. Доцільно відмітити, що рішення про закінчення свого функціонування приймається *Root* на основі аналізу номера $<cn>$, який надходить від обчислюючих процесорів, а не власної копії цієї змінної, розміщеної в локальній пам'яті *Root*. Це забезпечує гарантовано вірне завершення роботи паралельного алгоритму у випадку отримання *Root* повідомлення про завершення навчання останньої ІІДНМ від обчислюючого процесора.

Першою операцією любого обчислюючого процесора (рис. 1, б) є перевірка наявності нового повідомлення від *Root* за допомогою виклику функції *MPI_Probe()*. Якщо таке повідомлення прийшло, то обчислюючий процесор зчитує це повідомлення і отримує порядковий номер ІІДНМ $<cn>$, яку *Root* визначив йому для навчання. Після цього обчислюючий процесор виконує три операції по навчанню ІІДНМ:

– формує вибірки навчання на основі значень $row[cn]$ і $col[cn]$ для своєї версії ІІДНМ на основі послідовності кроків, описаних в [13];

– навчає ІІДНМ, в основі якої лежить модель багаторівневого персептрону та алгоритм зворотного поширення помилки;

– здійснює процедуру інтеграції історичних даних шляхом обчислення ІІДНМ вектора прогнозу і надсилає отримане значення разом з номером $<cn>$ ІІДНМ до процесора *Root*.



а б

Рис. 1. Грубозернисті паралельні алгоритмів процесорів *Master (a) і Slave* (б) навчання ІІДНМ динамічним розподілом

Розроблений паралельний алгоритм з динамічним розподілом використовує не точки синхронізації між гілками паралельної програми, а лише функції передавання та приймання повідомлень. Тому втрати часу, що призводять до зменшення ефективності цього алгоритму, можуть виникати за рахунок затримок, обумовлених технічною реалізацією операцій передавання/приймання повідомлень на конкретному паралельному комп'ютері.

При проведенні експериментальних досліджень вимірювався час паралельного навчання 50 модулів ІІДНМ. Значення часу паралельного навчання ІІДНМ замірювалось при виконанні паралельної програми на 2, 4 і 8 обчислюючих процесорах, один додатковий процесор виконував функції *Root*. Для розробки як послідовної, так і паралельної програми використано мову програмування С, паралельна версія реалізована з використанням бібліотеки паралельного виконання завдань MPI [12] та бібліотеки візуалізації MPE [8]. Експериментальні дослідження здійснювалися на паралельному комп'ютері Origin300, що містить 8 RISC-процесорів MIPS R14000 з тактовою частотою 500MHz та 4 Гб загальної оперативної пам'яті. Кожен процесор має 2 Мб кеш-пам'яті. Origin300 працює під керуванням ОС UNIX (IRIX64 6.5). Час виконання, прискорення та ефективність виконання паралельного завдання представлено в таблиці та на рис. 2.

Як видно з представлених результатів, прискорення виконання паралельної програми є нелінійним, ефективність розпаралелення падає як зі збільшенням середньоквадратичної помилки навчання SSE, так і зі зростанням кількості паралельно працюючих процесорів.

1. Час виконання паралельної програми, сек

Процесор	Кількість процесорів			
	1	2	4	8
$SSE=10^{-3}$	23,41	12,07	6,14	3,16
$SSE=10^{-4}$	173,14	88,70	45,22	25,23
$SSE=10^{-5}$	637,70	326,00	234,70	219,76
$SSE=10^{-7}$	1114,53	570,70	304,33	282,93

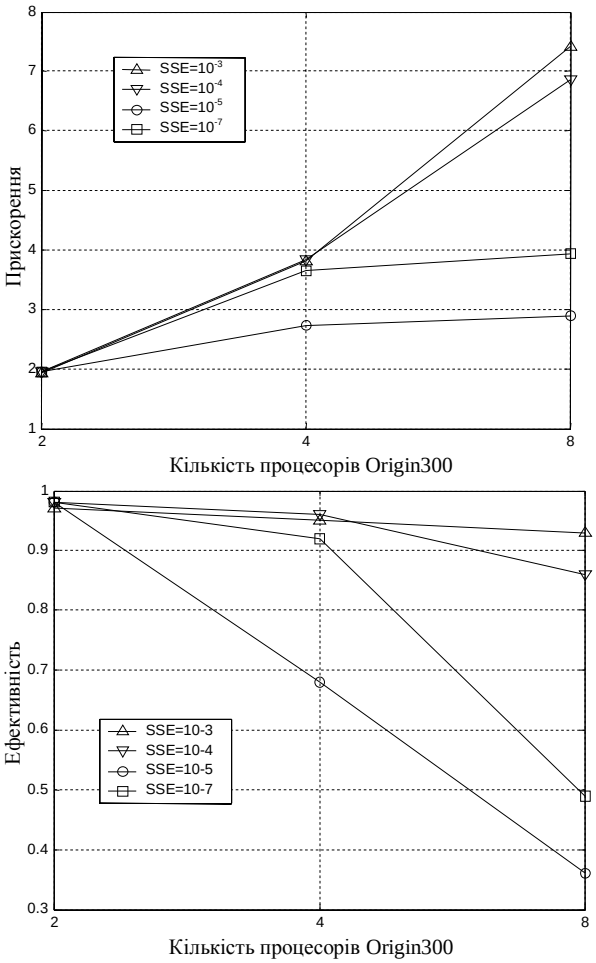


Рис. 2. Прискорення та ефективність виконання паралельного завдання

Аналіз ефективності розпаралелення. Результати експериментів, наведені в попередньому розділі, показують, що зі збільшенням середньоквадратичної помилки навчання мережі та збільшенням кількості використовуваних процесорів прискорення стає нелінійним, а ефективність розпаралелення зменшується. З урахуванням використання механізму передавання повідомлень між паралельно виконуваними задачами (а не використання механізму синхронізації), встановлено, що зменшення ефективності може бути спричинене:

– неточністю вимірювання часу виконання паралельної програми;

– затримкою в комунікаціях між паралельно працюючими програмами;

– нерівномірністю завантаження процесорів паралельного комп'ютера.

Для дослідження зменшення ефективності було використано бібліотеку візуалізації

MPE [8]. Для аналізу, вибрано дослід паралельного навчання 50 ІІДНМ на 8 процесорах Origin 300 при навчанні до $SSE=10^{-7}$ (рис. 3).

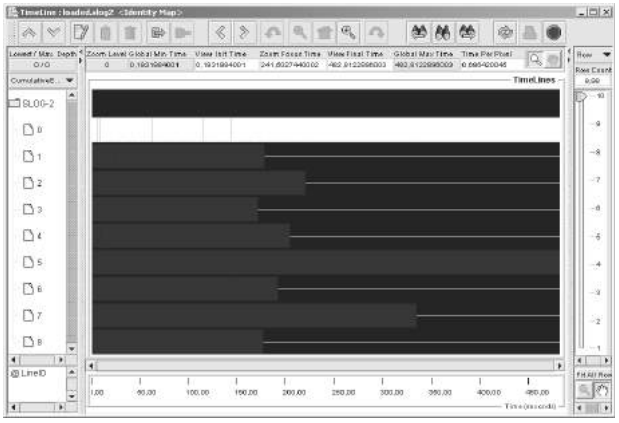


Рис. 3. Візуалізація навчання ІІДНМ з $SSE=10^{-7}$ на 8 процесорах Origin 300 з використанням астрономічного часу (по замовчуванню)

Темним кольором показано час роботи кожного процесора, світлим кольором – час очікування головного процесора *Root* до моменту збору даних від усіх 8 процесорів паралельного комп'ютера. Тонкими полосами темного кольору на світлому фоні полоси часу очікування процесора *Root* на рис. 3 показано час його роботи по отриманню результатів від обчислюючих процесорів та час комунікації. Результати візуалізації (рис. 3) показали, що загальний час виконання паралельної програми (див. поле *View Final Time* на рис. 3) набагато більший (482.81 сек) від отриманого часу виконання паралельної програми на 8 процесорах табл. 1 (282.93 сек, див. таблицю 4 і 5). Після серії проведених досліджень виявлено, що бібліотека MPE невірно обчислює час виконання паралельної програми. Зокрема, для обчислення часу роботи паралельної програми в самій програмі було використано функцію отримання процесорного часу *times()*, а MPE бібліотека заміряє реальний (астрономічний) час роботи паралельної програми, який залежить від завантаженості паралельного комп'ютера іншими системними та користувацькими завданнями. Тобто бібліотека MPE знаходить значення відрізка часу між двома подіями в минулому і майбутньому за допомогою функцій обчислення астрономічного часу. На довжину цього відрізка часу впливають усі прикладні та системні завдання та процеси, що виконує паралельний комп'ютер. Тому, чим більше паралельних завдань виконує паралельний комп'ютер, тим довший відрізок часу буде обчислений пакетом MPE.

З проведеного вище аналізу очевидно, що невірне обчислення часу виконання паралельної програми на процесорах паралельного комп'ютера призводить до невірного обчислення прискорення та ефективності розпаралелення. Уникнути цього недоліку можливо шляхом обчислення процесорного часу виконання кожного паралельного завдання. Під процесорним часом будемо розуміти дійсний час виконання завдання одним із процесорів паралельного комп'ютера без урахування часу, що він витрачає на виконання інших завдань паралельного комп'ютера.

Виконати поставлене завдання можливо шляхом модифікації існуючої бібліотеки візуалізації MPE. При модифікації бібліотеки MPE функцію вимірювання астрономічного часу цієї бібліотеки *CLOG_timestamp()* замінено на функцію вимірювання процесорного часу *times()*. Експериментальні дослідження модифікованої бібліотеки MPE виконувалися на спеціально завантаженому іншими завданнями паралельному комп'ютері Origin 300. Результати використання модифікованої бібліотеки MPE (рис. 4) відповідають рис. 3 по формі, але в той же час практично аналогічні результатам розпаралелення (див. таблицю). Зокрема, загальний час роботи паралельної програми при використанні модифікованої бібліотеки (див. поле *View Final Time* на рис. 5) складає 280.41 сек, що практично відповідає часу розпаралелення завдання (282.93 сек) на 8 процесорах (див. таблицю). Мала різниця між цими двома значення може бути пояснена похибкою вимірювання процесорного часу модифікованою бібліотекою MPE. Таким чином, використання модифікованої бібліотеки MPE на основі обчислення процесорного часу виконання паралельних завдань дає змогу ігнорувати вплив інших системних та користувацьких завдань, що паралельно виконуються на високопродуктивному комп'ютері Origin 300.

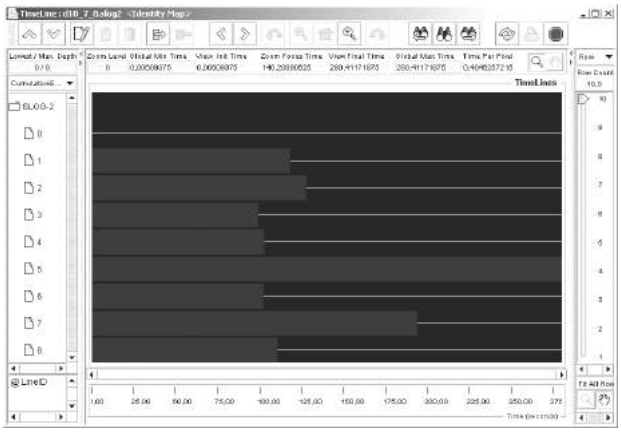


Рис. 4. Візуалізація навчання ІІДНМ з SSE=10^{-7} на 8 процесорах Origin 300 з використанням процесорного часу

За допомогою модифікованої бібліотеки візуалізації MPE доцільно дослідити причини зменшення ефективності розпаралелення даного завдання. Для цих досліджень використано формат *slog* (scalable logging) і програму jumpshot-4 [14]. Jumpshot-4 реалізовано на мові програмування Java і вимагає наявності пакету j2sdk (Software Development Kit), що має бути встановлений на комп'ютері для здійснення візуалізації. Ця програма дозволяє збільшити крок візуалізації при перегляді часу виконання паралельної програми і показати значно менші відрізки часу, ніж є в базових рисунках 3 чи 4. Так, на рис. 5 для 1-го обчислюючого процесора темним кольором показано час обчислення ним ІІДНМ, а світлим кольором – час комунікації з головним процесором *Root*. З рис. 5 видно, що час комунікації не перевищує 0,08 сек. У відповідності з алгоритмом рис. 1 за цей час головний процесор *Root* встигає прийняти від 1-го обчислюючого процесора результати по навчанню поточної ІІДНМ, записати їх у свою пам'ять, і направити на виконання 1-му процесорові наступну ІІДНМ для її навчання. Низькі втрати часу на комунікацію між *Root* та обчислюючими процесорами пітверджуються тим, що темні смуги на рисунках 3 і 4 (час обчислення) практично не перериваються світлими смугами (час комунікації). Тому витрати на комунікацію практично не впливають на зменшення ефективності розпаралелення даного завдання.

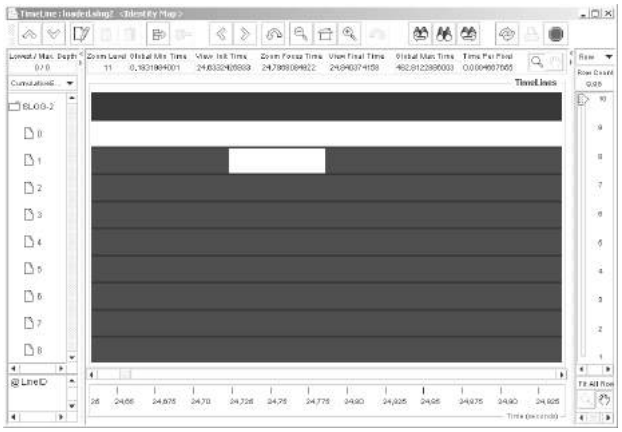


Рис. 5. Час комунікації між *Master* і *Slave* процесором не перевищує 0.08 сек

Аналіз результатів візуалізації на рисунках 3 та 4 показує, що зниження ефективності паралельного алгоритму з динамічним розподілом відбувається за рахунок нерівномірності часу навчання кожної ІІДНМ. Аналіз часів виконання інших дослідів паралельного алгоритму з динамічним розподілом показав, що вибраний приклад (рисунках 3 чи 4 відображає загальну тенденцію розподілу часів навчання набору ІІДНМ між процесорами паралельного комп'ютера.

Висновки. В статті оцінено ефективність паралельного грубозернистого алгоритму навчання модульних нейронних мереж шляхом візуалізації процесу розпаралелення. Для візуалізації використана стандартна бібліотека MPE. При проведенні дослідів виправлено недолік бібліотеки MPE, що базується на візуалізації реального часу виконання паралельної програми. Цей недолік проявився при виконанні алгоритму на паралельному комп'ютері Origin 300. Модифікація стандартної бібліотеки MPE шляхом заміни функції вимірювання процесорного часу дала змогу точно здійснити візуалізацію роботи паралельної програми незалежно від стану завантаження паралельного комп'ютера іншими завданнями. Візуалізація показала, що зниження ефективності алгоритму динамічного розподілу модульних нейронних мереж на процесори паралельного комп'ютера відбувається не через комунікаційні втрати між паралельними процесорами, а через нерівномірність їх завантаження, що

пов'язане з нестаціонарністю часу навчання окремих модулів нейронних мереж.

Список використаної літератури

1. Trobec, R. Parallel Computing: Numerics, Applications, and Trends / R. Trobec, M. Vajteršic, P. Zinterhof // Springer, 2009. – 520 p.

2. Chang, L.-C. An efficient parallel algorithm for LISSOM neural network / L.-C. Chang, F.-J. Chang // Parallel Computing. – 2002. – Vol. 28. – No. 11. – P. 1611 – 1633.

3. Estévez, P. A. A scalable parallel algorithm for training a hierarchical mixture of neural experts / P. A. Estévez, H. Paugam-Moisy, D. Puzenat et al. // Parallel Computing. – 2002. – Vol. 28. – No. 6. – P. 861 – 891.

4. Vampir – a Tool for Performance Optimization [Електронний ресурс]. – Режим доступа: http://www.vampir.eu/ –15.01.2013.

5. Góes, L. F. W. Performance Analysis of Parallel Programs using Prober as a Single Aid Tool / L. F. W. Góes, L. E. S. Ramos, C. A. P. S. Martins // Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing. – 2002. – P. 204 – 211.

6. TAU Performance System® - a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, Java, Python [Електронний ресурс]. – Режим доступа: http://www.cs.uoregon.edu/research/tau/home.php – 15.01.2013.

7. Moore, S. Review of Performance Analysis Tools for MPI Parallel Programs / S. Moore, D. Cronk, K. S. London, J. Dongarra // Proceedings of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. – 2001. – P. 241 – 248.

8. MPE – a software package for Message Passing Interface programmers [Електронний ресурс] – Режим доступа: http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm#MPE – 15.01.2013.

9. MPICH – a high performance and widely portable implementation of the Message Passing Interface standard [Електронний ресурс]. – Режим доступа: http://www.mpich.org/ – 15.01.2013.

10. Turchenko, V. Parallel Algorithm of Dynamic Mapping of Integrating Historical Data Neural Networks / V. Turchenko // Information Technologies and Systems. – 2004. – Vol. 7. – No. 1. – P. 45 – 52.

11. Sachenko, A. Instrumentation for Data Gathering / A. Sachenko, V. Kochan, V. Turchenko // IEEE Instrumentation and Measurement Magazine. – 2003. – Vol. 6. – No. 3. – P. 34 – 40.

12. A High Performance Message Passing Library [Електронний ресурс]. – Режим доступа: http://www.open-mpi.org/ –15.01.2013.

13. Турченко, В. Улучшенный метод интеграции исторических данных з использованием нейронных сетей / В. Турченко, В. Кочан, А. Саченко, Т. Лаопоулос // Датчики и системы. – 2002. – № 7 (38). – С. 35 – 38.

14. Performance Visualization for Parallel Programs [Електронний ресурс]. – Режим доступа: http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm#Jumpshot-4 – 15.01.2013.

Отримано 12.02.2013

References

1. Trobec, R. Parallel Computing: Numerics, Applications, and Trends, Springer / R. Trobec, M. Vajteršic, P. Zinterhof. – 2009, 520 p. [in English].

2. Chang, L. -C. An efficient parallel algorithm for LISSOM neural network, Parallel Computing / L.-C. Chang, F.-J. Chang. – 2002. – Vol. 28. – No. 11. – P. 1611 – 1633 [in English].

3. Estévez, P. A. A scalable parallel algorithm for training a hierarchical mixture of neural experts / P. A. Estévez, H. Paugam-Moisy, D. Puzenat et al. // Parallel Computing. – 2002. – Vol. 28. – No. 6. – P. 861 – 891 [in English].

4. Vampir – a Tool for Performance Optimization [Electronic resource]. –Access mode: http://www.vampir.eu/ – 15.01.2013 [in English].

5. Góes, L. F. W. Performance Analysis of Parallel Programs using Prober as a Single Aid Tool / L. F. W. Góes, L. E. S. Ramos, C. A. P. S. Martins // Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing. – 2002. – P. 204 – 211 [in English].

6. TAU Performance System® - a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, Java, Python [Electronic resource].– Access mode: http://www.cs.uoregon.edu/research/tau/home.php – 15.01.2013 [in English].

7. Moore, S. Review of Performance Analysis Tools for MPI Parallel Programs / S. Moore, D. Cronk, K. S. London, J. Dongarra // Proceedings of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. – 2001. – P. 241 – 248 [in English].

8. MPE – a software package for Message Passing Interface programmers [Electronic resource]. – Access mode: http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm#MPE – 15.01.2013 [in English].

9. MPICH - a high performance and widely portable implementation of the Message Passing Interface standard [Electronic resource]. – Access mode: http://www.mpich.org/ – 15.01.2013 [in English].

10. Turchenko, V. Parallel Algorithm of Dynamic Mapping of Integrating Historical Data Neural Networks / V. Turchenko // Information Technologies and Systems. – 2004. – Vol. 7. – No. 1. – P. 45 – 52 [in English].

11. Sachenko, A. Instrumentation for Data Gathering / A. Sachenko, V. Kochan, V. Turchenko // IEEE Instrumentation and Measurement Magazine. – 2003. – Vol. 6. – No. 3. – P. 34 – 40 [in English].

12. A High Performance Message Passing Library [Electronic resource]. – Access mode: http://www.open-mpi.org/ –15.01.2013 [in English].

13. Turchenko, V. Enchanced Method of Historical Data Integration Using Neural Networks / V. Turchenko, V. Kochan, A. Sachenko, Th. Laopoulos // Datchiki i Sistemy, Moscow, Russia. – 2002. – № 7 (38). – P. 35 – 38 [in Russian].

14. Performance Visualization for Parallel Programs [Electronic resource]. – Access mode:http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm#Jumpshot-4 –15.01.2013 [in English].

Турченко
Ірина Василівна,
канд. техн. наук, доц. каф. інформаційно обчислювальних систем та управління Тернопільського нац. економічного ун-ту,
e-mail: itu@tneu.edu.ua,
тел. 0352 475050#12321