УДК 004.8, 004.051

*А.Ю. Дорошенко, О.С. Косань*

# ОПТИМІЗАЦІЯ ПАРАЛЕЛЬНОГО ПЕРЦЕПТРОНА ДЛЯ ЦЕНТРАЛЬНИХ ПРОЦЕСОРІВ

Розглядається проблема ефективної реалізації багатошарового перцептрона для центральних процесорів класу i686. Запропоновано методику оптимізації за швидкодією процедур обчислення і навчання перцептрона на цих процесорах із застосуванням багатопоточності та використання паралельних інструкцій.

## Вступ

На сьогодні актуальними залишаються швидкісні обчислення нейромереж (НМ) у задачах класифікації та розпізнавання зображень великого розміру, що використовуються, зокрема, в задачах комп'ютерного зору. Для ефективного функціонування нейронних мереж існує можливість створити спеціалізовані рішення для різнотипних процесорів з найширшим використанням їх можливостей.

Нейронна мережа типу багатошарового перцептрона складається з елементарних структурних одиниць, що називаються нейронами [1, 2] (рис. 1).

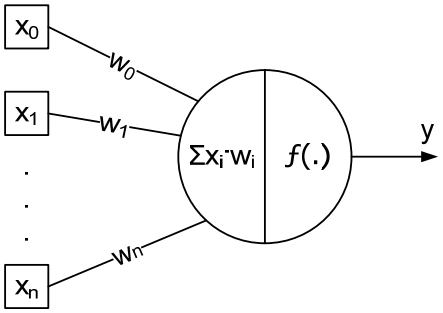


Рис. 1. Окремий нейрон нейромережі

Кожен нейрон має декілька входів ($x_1,..,x_n$), на які подаються дійсні значення. Кожна вхідна величина домножується на відповідну їй величину вагового коефіцієнта ($w_1,..,w_n$). Далі кожен частинний добуток подається на суматор і на обрану активаційну функцію нейрона. У цій роботі використана активаційна функція – однополярний сигмоїд, який був обраний через високу обчислювальну простоту його похідної, що обчислюється, наприклад, як значення виразу $f(sum) \cdot (1 - f(sum))$. В реалізації активаційна функція може бути замінена на біполярну (рис. 2) для кращої збіжності навчання [1].

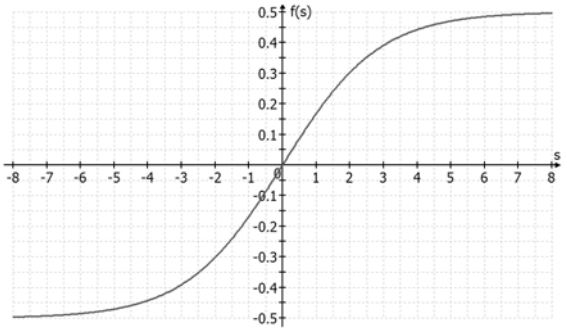


Рис. 2. Біполярна активаційна функція нейрона – сигмоїд

Існує велика кількість програмних реалізацій нейромереж для роботи на центральних процесорах, але більшість з них не пристосовані для виконання в багатопоточному режимі, або не оптимізовувались спеціально для високої ефективності. Частина з розглянутих існуючих бібліотек для роботи з нейронними мережами не повністю використовують обчислювальні потужності центральних процесорів, такі як SIMD-інструкції. У табл. приведені основні переваги та недоліки оглянутих бібліотек. У цій роботі проводиться дослідження ефективності виконання задачі обчислення багатошарового перцептрона на мультиядерних процесорах та пропонуються рекомендації щодо методики їх оптимального використання. Для досягнення максимальної гнучкості рішення, задача розглядається на найнижчому рівні розпаралелювання, без використання сторонніх бібліотек, таких як OpenMP та Intel TBB [3].

Таблиця

| Назва бібліотеки | Багато поточність | Набір типів архітектур мереж | Платформи | Ліцензія та код | Швидкість |
|---|---|---|---|---|---|
| FANN [4] | Ні | Всі загальні | Будь-яка | LGPL, Відкритий | Висока |
| Encog [5][6] | Так | Основні | Java та .NET (Windows) | LGPL, Відкритий | Висока |
| Flood [7] | Ні | Перцептрон | Будь-яка | Відкритий | Висока |
| JOONE [8] | Так | Основні | Java | Відкритий | Середня |
| Neuroph [9] | Ні | Більшість загальних | Java | Відкритий | Низька |

## 1. Постановка задачі

Для ефективного розв'язання поставленої задачі необхідно врахувати особливості сучасних центральних процесорів, такі як багатопоточність та можливості SIMD-інструкцій для прискорення обчислення нейронної мережі. Додатково важливо враховувати фактори, що потенційно можуть вплинути на зниження ефективності обчислень, а саме: часті виконання умовних операторів, відсутність вирівнювання даних, неструктурованість та непослідовність даних, конкуренція за ресурси і т. і. Для цього необхідно проводити аналіз параметрів задачі, при яких використання тих чи інших засобів оптимізації призводять до прискорення виконання (або уповільнюють) програму.

Експериментальну задачу для дослідження ефективності виконаємо спочатку в вигляді спрощеної задачі обчислення одношарового перцептрона на декількох процесорах та з окремими алгоритмічними шляхами для однопоточного та багатопоточного режиму. Результати досліджень наведені далі.

1. Характерні низькі затрати на синхронізацію потоків при правильному групуванні задачі, але неможливо усунути недолік недостатньої пропускної здатності пам'яті для високих розмірностей задачі. Досягнуто прискорення від 160 % (середні розмірності) до 125 % (високі розмірності) за рахунок другого ядра. Очевидно, що при зростанні розмірності задачі, для коефіцієнтів нейронів виявилось недостатньо вільного місця кеш-пам'яті.

2. Створення потоків для малорозмірної задачі (менше 210 нейронів на шар) (32x32) не доцільно через затрати часу операційної системи (ОС). Для ОС Windows створення потоку займає більше 2 мкс, активізація та зміна параметрів також вимагають частих затримок під час переключення в режим ядра.

3. Повторний запуск потоку обчислення дозволяє зберегти дані кеш-пам'яті та знизити звернення до ОЗП. З урахуванням цього, доцільно використовувати пул потоків.

4. Відмова від планування потоків ОС призводить до незначного (~3 %) підвищення продуктивності обчислень. З іншого боку, жорстка прив'язка потоку до процесора (за рахунок Affinity Mask) не доцільна на дрібних задачах з урахуванням часових витрат на її здійснення. Також жорстка прив'язка призводить до зниження надійності програми та стійкості результату – у випадку блокування одного ядра, програма може перейти в тривалий цикл очікування.

5. Використання суперскалярних «широких» інструкцій розряду Streaming SIMD Extensions (SSE) призводить до прискорення на 20 – 300 %, але потребує спеціальних методик вибірки та збереження даних, що вимагає створення альтернативного алгоритмічного кодового шляху). Використання при компіляції додатково SSE2 та SSE3 не вплинуло на показники швидкодії. Для SSE вкрай необхідне вирівнювання та послідовність подачі даних, що

виключає використання об'єктної моделі у коді обчислення програми.

6. Зберігання проміжного варіанта обчислення похідної сигмоїди для його повторного використання не завжди вигідніше його повторного розрахунку, оскільки використовується проста функція активації, та похідна що залежить від неї.

7. Використання табличних функцій (для сигмоїди) у даній задачі виправдалося тільки на процесорі Xeon з HyperThreading та дало приріст у 10 – 15 %, в усіх інших випадках дало гірший результат ніж розрахунок точного значення сигмоїди. Причина може полягати в перенавантаженості шини пам'яті та критичності кеш пам’яті для корисних даних. Виключення на серверному процесорі можна пояснити його зниженими арифметичними можливостями.

8. Відсутність вирівнювання даних унеможливлює ефективну роботу програми, оскільки вона необхідна не тільки для виконання команд SSE, але і для реалізації ефективної синхронізації (spinlock).

Порівняння ефективності обчислень задачі на різних конфігураціях показано на рис. 3. Складність задачі в даному випадку – розмір сторони матриці вхідного зображення (наприклад, останнє значення становить 14×14 = 196 входів НМ). З графіка видно, що найбільший час обчислення був отриманий в однопоточній програмі. Використання SSE значно знижує затрати часу на обчислення. Використання SSE в декількох потоках одночасно виявилось ефективним, але витрати на створення потоків на таких розмірностях призвели до того, що багатопоточна програма працювала загалом повільніше, ніж однопоточна. При використанні пулу потоків вдалось компенсувати ці витрати.

Рішення проблеми підбору параметрів може заключатись у створенні профілю продуктивності процесора для задачі з різними коефіцієнтами на базі попереднього (тестового) обрахунку задачі з різними параметрами. Під час тесту продуктивності визначаються моменти, коли включення тих чи інших технологій призводить до зростання продуктивності.

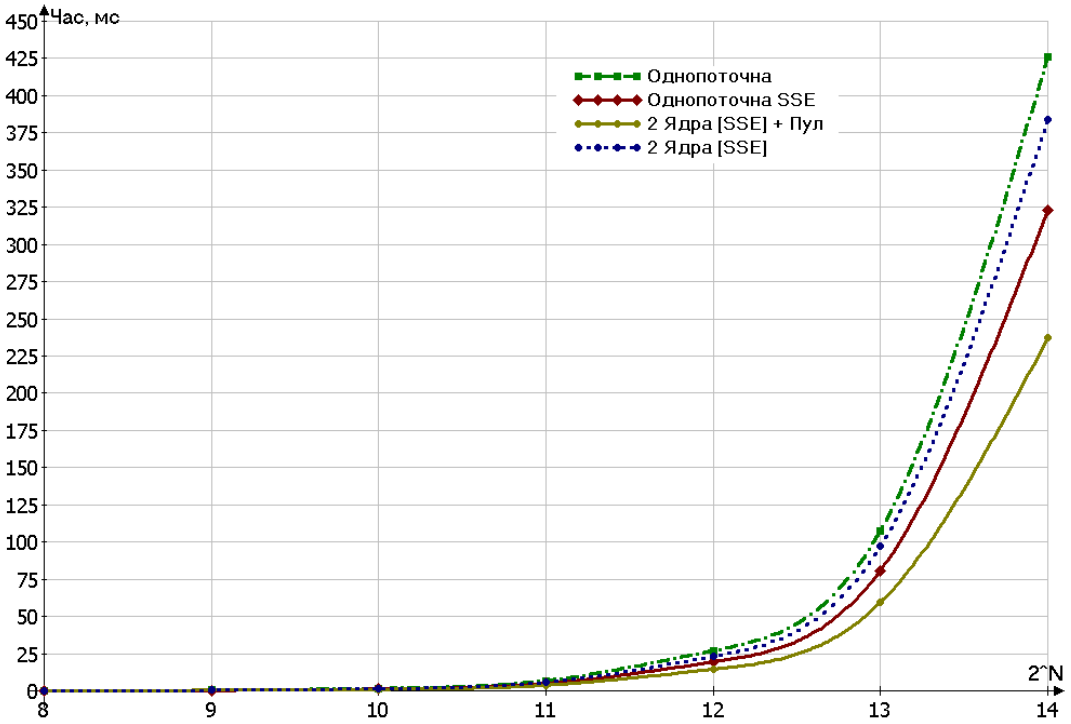


Рис. 3. Залежність часу обчислення тестової задачі розрахунку одношарового перцептрона від складності задачі

В даній реалізації встановлені жорсткі пороги включення додаткового потоку (не менше 64-х нейронів). На рис. 4 показано схематичне зображення задачі для двох ядер.

На вході довільна кількість вхідних нейронів (доповнена до кратної 4 для вирівнювання даних), на виході – довільна кількість вихідних нейронів.

Мережа повнозв'язна, всі вагові коефіцієнти для всієї мережі виділені в динамічній пам'яті за один раз за попереднім розрахунком їх ємкості з урахуванням вирівнювання. З рис. 4 також видно, що для кожного потоку відводиться кількість нейронів, кратна 4 (вимога SSE). Розподілення нейронів кожного шару між потоками проводиться рівномірно, у випадку неможливості поділити рівно, перевага надається не першому потоку, оскільки він додатково завантажений синхронізацією усіх потоків.

Математично процес обчислення вихідних значень кожного шару буде представляти добуток вектора вихідних значень попереднього шару $X$ та матриці вагових коефіцієнтів $W$ для кожного нейрона цього шару.

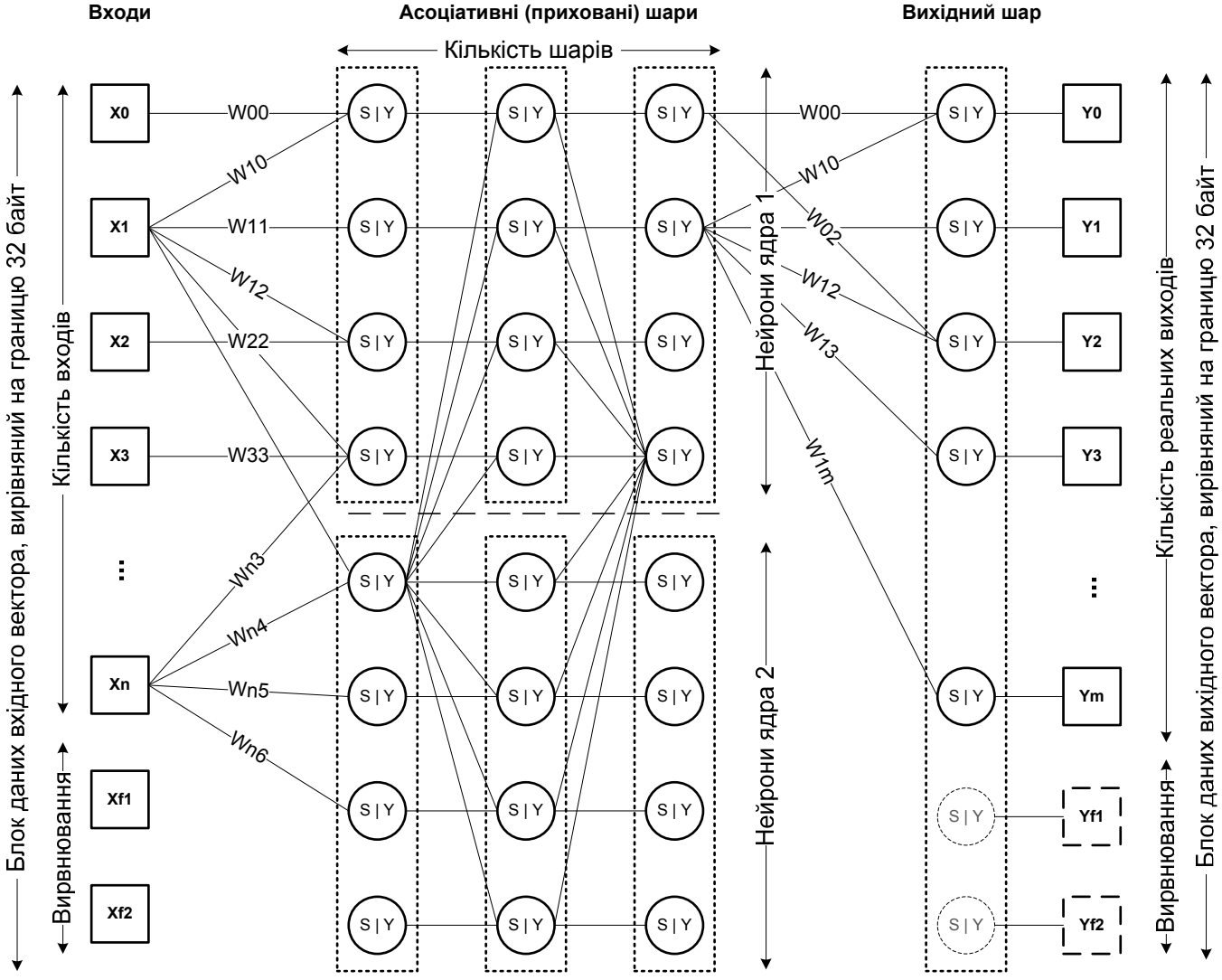


Рис. 4. Програмне розбиття задачі для декількох ядер
(показані не всі зв'язки між нейронами)

Такий метод розпаралелювання за рядками (рис. 5) може бути виправданий, оскільки таким чином для кожного з $k$ процесорів утворюється максимально довга послідовність безперервних даних, довжиною $n$, та мінімальна кількість синхронізуючих операцій (максимальна незалежність даних), необхідна по закінченню $q$ рядків. Також, таке розділення потребує повного дублювання вектора $X$ для кожного з ядер.

Оскільки вектор $X$, при збільшенні кількості входів зростає лінійно, а матриця вагових коефіцієнтів – квадратично, то обраний метод розділення буде мати мінімальний негативний вплив на ефективність обчислень за умови, що весь вектор $X$ повністю розміщений в кеш пам'яті даних. В іншому випадку доцільно використовувати блочне розбиття або розбиття за стовпцями матриці $W$.

Процес навчання нейронної мережі методом зворотного розповсюдження помилки включає в себе етап обчислення і корекції вагових коефіцієнтів мережі. Для прискорення останнього, доцільно створити варіант функції обчислення, що розраховує також і похідні виходів.

Для фантомних вхідних та вихідних нейронів (що створюються для вирівнювання), виділяється додаткова пам'ять у масиві вагових коефіцієнтів, але участі в розрахунку вони не приймають (крім варіанта де використовується SSE, тоді можна обрати використання надлишкових нейронів).

У даній реалізації для синхронізації використовується механізм подій (об'єкт Event для синхронізації в ОС Windows [10]). Оскільки одиниці рядка мають однакову обчислювальну ємність, то ефективність обчислення буде залежати від одночасності початку та закінчення розрахунку.

Для оптимізації SSE використовується ручний розподіл змінних вектора суми $s$ ($s_0..s_3$) – оптимізатор компілятора самостійно додає інструкції SSE де це можливо. Арифметичні операції перегруповані так, щоб мати мінімальну залежність операндів.

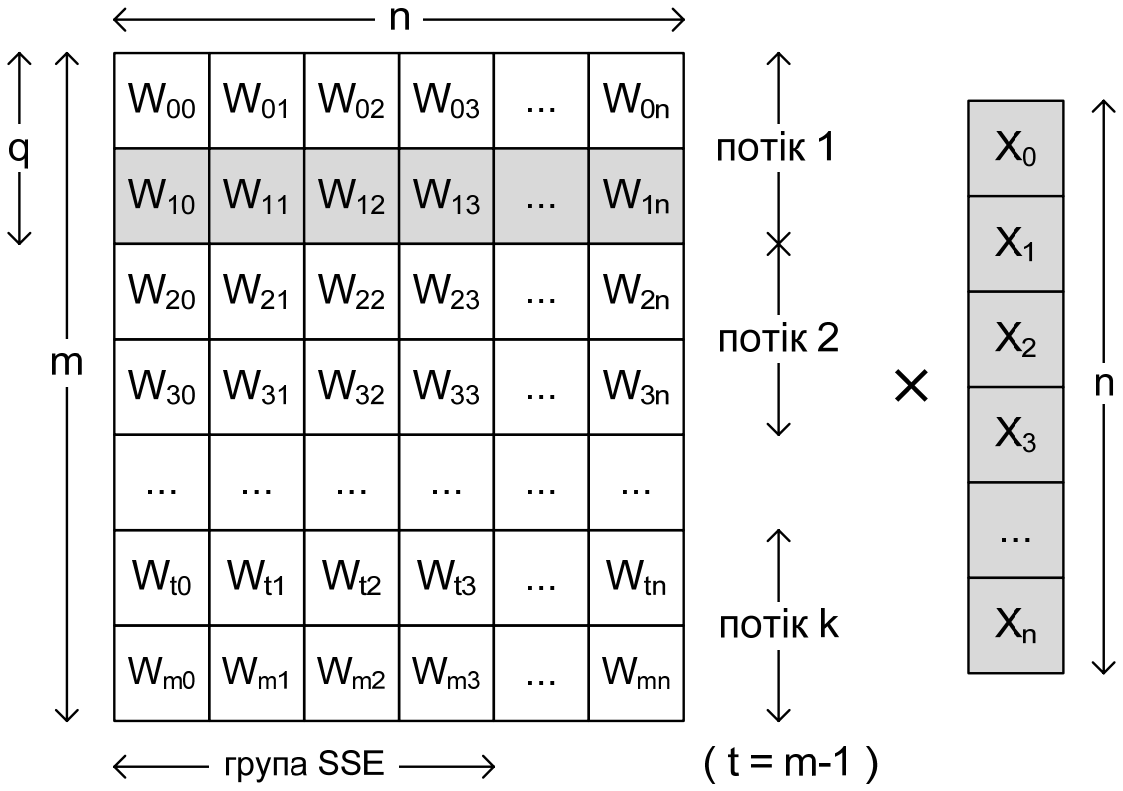


Рис. 5. Представлення даних одного шару та їх розділення між потоками

## 2. Результати експериментів

Обчислення здійснювались на системі з наступною конфігурацією:

процесор – AMD Phenom FX-5000 (2.42 ГГц, 4 ядра, 2 Мб L2 кеш, 6 Мб L3 кеш), обсяг оперативної пам'яті – 2 Гб; ОС – Windows XP SP3;

Результат швидкості обчислень для різних задач показано на рис. 6. З якого видно, що однопоточна програма, яка використовує SSE має значну перевагу над простою однопоточною програмою. Водночас, багатопоточна програма, що використовує SSE, має низку перевагу від додаткових ядер, що може бути пояснено обмеженою пропускною здатністю пам'яті [11]. Код експерименту розміщений у [12].

Ураховуючи результати експериментів можна запропонувати ряд рекомендацій:

- попередньо проводити тест розпаралелювання задачі у спрощеному вигляді;

- у залежності від того, чи буде програма конкурувати за ресурси з іншими програмами, а також беручи до уваги енергозбереження, обираємо спосіб синхронізації потоків – можливостями операційної системи (мьютекси, семафори та події), чи спін-лок (англ. spin-lock) синхронізацію вбудованими (intrinsic) функціями. Як правило, спін-лок синхронізація більш ефективна за швидкодією [10, 13], оскільки не вимагатиме виклику функцій ядра, але більш енерговитратна, оскільки ядра очікують сигнал синхронізації у циклі;

- при розділенні задач для кожного потоку необхідно розділяти оброблювані дані так, щоб кожен потік зберігав максимальну кількість даних в кеш пам'яті, та не заміщував їх при кожній операції;

- використовувати інструкції SSE та інші, в тих випадках, коли можна гарантувати кратність вхідних даних до розміру «широкого» регістру. У випадках, коли розмірності не збігаються, необхідно про-

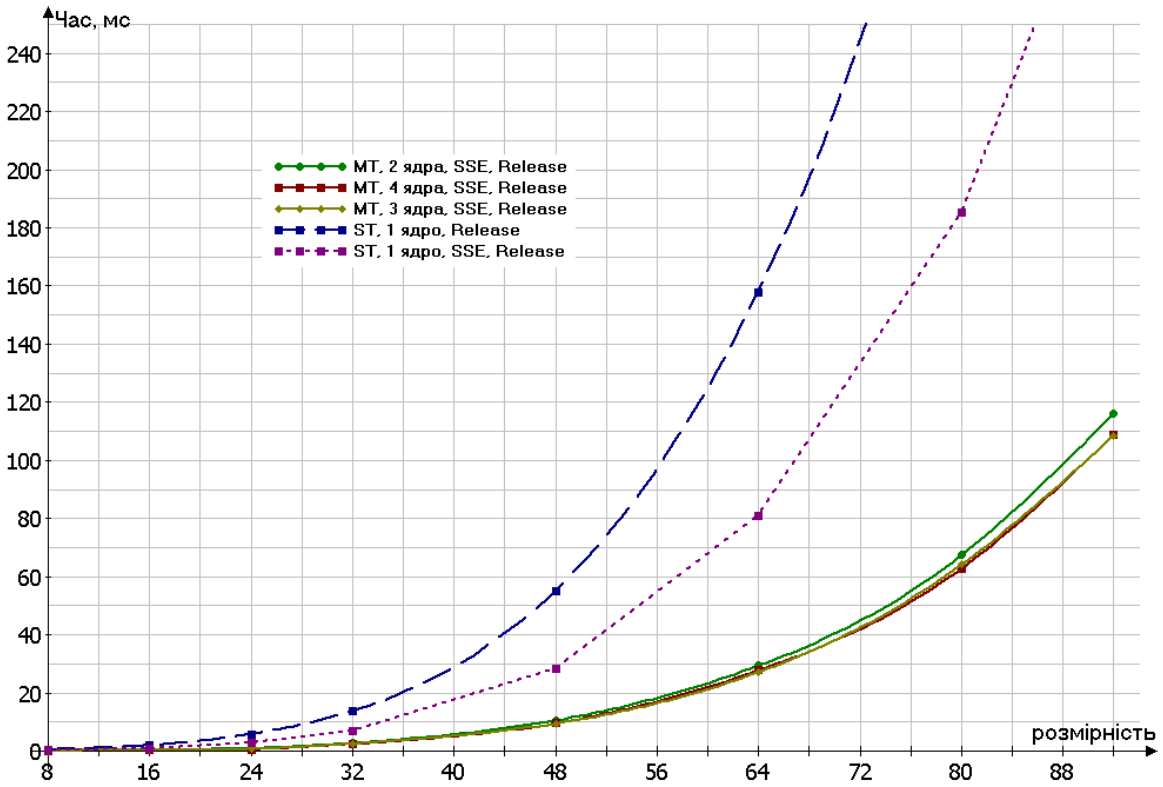


Рис. 6. Залежність часу обчислення задачі розрахунку багатошарового перцептрона від складності задачі

вести доповнення (до 32 байт, у випадку SSE) та вирівнювання вхідних даних (на адресу кратну 32 байтам), при чому забезпечити відсутність впливу від обчислення вирівнювальних даних (найчастіше досягається зануленням);

- розгортати цикли, в тому числі частково, коли за рахунок цього можливо зменшити витрати на повторне обчислення, або значно перегрупувати дані та порядок операцій над ними. При цьому необхідно враховувати, що надмірне розгортання циклу (більше 16 – 32 ітерацій) призведе до збільшення розміру коду програми, внаслідок чого може знизитись її вмістимість у кеші інструкцій і, як наслідок, знизиться швидкодія;

- у випадку, коли не використовуються асемблерні вставки, для досягнення застосування компілятором широких команд та паралельних конвеєрів сучасних процесорів у результаті оптимізації, необхідно групувати порядок операцій над даними так, щоб команди з взаємонезалежними операндами йшли підряд, а їх дані знаходились згрупованими та йшли послідовно, що частково протирічить концепції ООП, де об'єкт зберігає самодостатність, та зберігає різнорідні дані поряд;

- для зниження фрагментованості пам'яті доцільно виділяти пам'ять для всіх динамічних буферів задачі, єдиним системним викликом, коли це можливо;

- використовувати вбудовувані функції для невеликих операцій, та вбудовані (intrinsic) функції для зниження витрат на системний виклик;

- при обробці великих обсягів даних використовувати табличні функції тільки тоді, коли її значення має значну арифметичну складність (більше 5 – 10 арифметичних операцій, що може бути досягнуто шляхом ускладнення активаційної функції), оскільки таблиця значень буде конкурувати з даними за спільні ресурси – кеш пам'ять та пропускну здатність шини пам'яті.

## Методика

Підсумовуючи вищевказане можна сформулювати наступні пункти методики оптимізації за ефективністю процедур обчислення та навчання багатошарового перцептрона на мультиядерних процесорах.

1. Створити для кожного ядра відповідну нитку обчислень, що бере на себе задачу розрахунку вихідного значення рівномірно розділених між ядрами і послідовно розміщених нейронів з кожного шару. Розділити рівномірно між ядрами нейрони кожного шару, при цьому розділення матриці вагових коефіцієнтів здійснювати за рядками. Всередині множити елементи рядка та вхідного вектора з використанням інструкцій SSE, при цьому доцільно розгортати цикл на 4 – 16 операцій (1 – 4 інструкції SSE). Для великих розмірностей синхронізацію потоків здійснювати методом spinlock, для великих – через механізм подій.

2. Для процедури навчання необхідне створення окремих потоків корекції коефіцієнтів, а також процедури обчислення з модифікованим алгоритмом, що обчислює необхідні значення похідних виходів кожного шару.

Дана методика може бути також використана для оптимізації нейронних мереж і для інших архітектур.

## Висновки

Досліджено та розроблено методику підвищення ефективності програмної реалізації багатошарового перцептрона на багатоядерному центральному процесорі. Для підвищення ефективності обчислень перевірено різнотипні засоби оптимізації, найефективнішим з яких виявилося використання SIMD-інструкцій. При комбінуванні всіх методів було виявлено, що для ефективного обчислення задачі великих розмірностей необхідна висока пропускна здатність пам'яті.

1. *Комашинский В.И., Смирнов Д.А.* Нейронные сети и их применение в системах управления и святи. – М.: Горячая линия-Телеком, 2002. – 94 с.
2. *Хайкин Саймон.* Нейронные сети: полный курс, 2-е изд., испр.: Пер. С англ. – М.: ООО Вильямс, 2006. – 1104 с.
3. *Rainders James.* Intel Threading Building Blocks. – Sebastopol: O'Reilly Media, Inc., 2007. – 305 с.
4. *FANN* http://leenissen.dk/fann/wp/

5. *Encog* http://www.heatonresearch.com/encog
6. Порівняльний тест нейромереж http://www.codeproject.com/KB/recipes/benchmark-neuroph-encog.aspx (3/6/2010)
7. *Flood* http://www.cimne.com/flood/
8. *JOONE* https://sourceforge.net/projects/joone/
9. *Neuroph* http://neuroph.sourceforge.net/
10. *Рихтер Джеффри, Кристоф Назар*. Windows via C/C++. Программирование на языке Visual C++. – СПб.: Питер, 2009. – 896 с.
11. *Breshears Clay.* The Art of Concurrency. – Sebastopol: O'Reilly Media, Inc., 2009. – 287 с.
12. *Код* бібліотеки та програми експерименту, [archive] URL http://endrobene.org/download.php?f=xnn_exp_2.zip (10 травня 2011)
13. *Herlihy Maurice, Shavit Nir.* The Art of Multiprocessor Programming, Morgan Kaufmann Publishers, 2008. – 508 с.

Отримано 19.05.2011

***Про авторів*:**

*Дорошенко Анатолій Юхимович*,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень,

*Косань Олег Станіславович*,
студент 5-го курсу магістратури.

***Місце роботи авторів:***

Інститут програмних систем
НАН України,
e-mail: dor@isofts.kiev.ua,

Національний технічний університет
України «КПІ»,
03056, Київ, проспект Перемоги, 37.
Тел.: 068 5993012,
e-mail: jscythe@gmail.com