УДК 004.89

О.В. Олещук, О.Є. Попель, М.Б. Копитчук
Одеський національний політехнічний університет

МОДЕЛЮВАННЯ ПОВНОЗВ'ЯЗНОЇ НЕЙРОННОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

Розглянуто задачу істотного підвищення продуктивності обчислювальних систем за рахунок використання сучасних апаратних засобів, таких як графічний процесор загального призначення. Описано відповідну програмну технологію CUDA і проаналізовано її ключові особливості, які суттєво впливають на продуктивність. На основі проведеного аналізу вибрана модель нейронної мережі та описано підхід до її реалізації. Наведено порівняльний аналіз реалізацій нейронної мережі на центральному та графічному процесорі, а також вплив деяких параметрів мережі на продуктивність.

Ключові слова: нейронні мережі, повнозв'язна нейронна мережа, GPU загального призначення, технологія CUDA.

There is considered the problem of significant improvement of computing systems performance by using modern hardware such as a general purpose graphics processing units. An appropriate software technology CUDA is considered and its key features that significantly affect performance are analyzed. Based on the analysis a neural network model is selected and is described an approach to its implementation. We give a comparative analysis of neural network implementations on central and graphics processors, as well as the influence of several parameters on network performance.

Key words: neural networks, fully connected neural network, general purpose GPU, CUDA technology.

Постановка проблеми

У наш час інформаційні технології активно впроваджують в побутову техніку та промислові інформаційно-вимірювальні системи. При цьому поряд з датчиками, що вимірюють температуру або тиск, реєструють факт контакту із зовнішнім об'єктом і формують в результаті одиничні скалярні сигнали, задіються і реєстратори аудіо- і відеосигналів. Це, з одного боку, значно розширює потенційні можливості пристроїв, а з іншого – ускладнює використовуваний математичний апарат і відповідно істотно підвищує вимоги до обчислювальних можливостей пристрою.

Одним з найперспективніших математичних апаратів є штучні нейронні мережі. Як відомо, нейромережеві технології дозволяють успішно розв'язувати деякі класи задач, до яких слабко застосовується звичайний логічний апарат. Це задачі розпізнавання образів, багатофакторний аналіз, експертні системи тощо. Однак досі сфера застосування нейронних мереж була багато в чому обмежена через їх високу обчислювальну складність. Сучасний стан передових сфер промисловості висуває вимогу до вирішення подібних завдань в режимі реального часу. Одним із прикладів, найбільш актуальних і перспективних сьогодні, є системи навігації та автоматизованого керування транспортними засобами. Побудова подібних систем потребує багатократного підвищення продуктивності нейронних мереж і водночас дає змогу використовувати тільки такі апаратні засоби, які можуть бути задіяні як компоненти вбудованих обчислювальних систем.

Аналіз джерел

Одним зі способів прискорення обчислювальних процесів є їх розпаралелювання [1] з використанням технології GPGPU (General purpose graphics processing units – графічного процесора загального призначення). GPGPU – це технологія використання графічного процесора відеокарти,

що має зазвичай справу тільки з обчисленнями для комп'ютерної графіки, для виконання розрахунків, які, як правило, виконує центральний процесор. Це стало можливим завдяки додаванню програмованих шейдерних блоків і підвищеній арифметичній точності растрових конвеєрів, що дає змогу розробникам програмного забезпечення використовувати потокові процесори для неграфічних даних. У наш час час існує декілька моделей програмування GPU і відповідних програмних технологій, які забезпечують прямий доступ до апаратних можливостей відеокарт. Однією з них є технологія CUDA [2]. CUDA – це програмно-апаратна архітектура, яка дає змогу виконувати обчислення з використанням графічних процесорів NVIDIA, що підтримують технологію GPGPU.

Як предметні області, де вже з успіхом застосовується технологія CUDA, називають такі галузі, як медицина, фінанси, геологія [2]. Тут вирішуються такі завдання, як аналіз ринку в режимі реального часу, формування зображень глибинних шарів на основі сейсмологічних даних, формування тривимірного зображення тіла людини під час ультразвукового дослідження. Природно, що технологія CUDA застосовується і в IT-сфері. Прикладом подібних завдань є вельми ресурсомістка задача перекодування відео з одного формату в інший. За заявами розробників відповідних програмних продуктів, використання технології CUDA дає змогу добитися приросту продуктивності під час розв'язання згаданих задач на один-два порядки. Такий стрибок продуктивності відкриває принципово нові можливості, оскільки переводить завдання з розряду вирішуваних в умовах відкладеного часу в задачі реального часу.

Разом із системним програмним забезпеченням, що дозволяє задіяти можливості графічного ядра, платформа CUDA надає ряд бібліотек, де реалізовано той чи інший математичний апарат. Серед подібних бібліотек [2] можна відзначити реалізацію лінійної алгебри і перетворень Фур'є. На жаль, досі не існує стандартних рішень, що реалізують на платформі CUDA штучні нейронні мережі. Однією з причин цього є велика різноманітність видів нейронних мереж [3], а також деякі обмеження, накладені технологією CUDA. Однак вже існують реалізації окремих видів нейронних мереж, наприклад, згортальної нейронної мережі [4].

Мета

Мета цієї статті – ознайомлення з основами використання графічного ядра для неграфічних обчислень з використанням технології CUDA, виявлення основних особливостей і обмежень вибраної інформаційної технології і побудова з урахуванням виявлених особливостей повнозв'язної нейронної мережі.

Загальні відомості про технологію CUDA

Обчислювальна архітектура CUDA основана на концепції "одна команда на множину даних" (Single Instruction Multiple Data, SIMD) і понятті мультипроцесора. Концепція SIMD припускає, що одна інструкція дає змогу одночасно обробити величезну кількість даних. Мультипроцесор – це багатоядерний SIMD-процесор, який дозволяє в кожний певний момент часу виконувати на всіх своїх ядрах тільки одну інструкцію.

Такий графічний пристрій дає можливість ефективніше виконувати обчислення на графічному процесорі, ніж на центральному, за умови, що задача допускає поділ на багато потоків. Як і під час роботи на CPU, одним з вузьких місць в обчислювальній системі є звернення до пам'яті. На GPU повільні звернення до пам'яті приховують, використовуючи паралельні обчислення. Поки одні завдання очікують дані, працюють інші, готові до обчислень. Це один з основних принципів CUDA, що дозволяє значно підвищити продуктивність загалом.

Структуру GPU показано на рис. 1. Верхній рівень ядра GPU складається з блоків, які групуються у сітку (грід, grid). Будь-який блок, своєю чергою, складається з ниток (потоків, threads), які є безпосередніми виконавцями обчислень. Нитки в блоці сформовані у вигляді тривимірного масиву. Відповідно в разі використання технології CUDA виникає проблема коректного розбиття на потоки для отримання максимальної продуктивності.

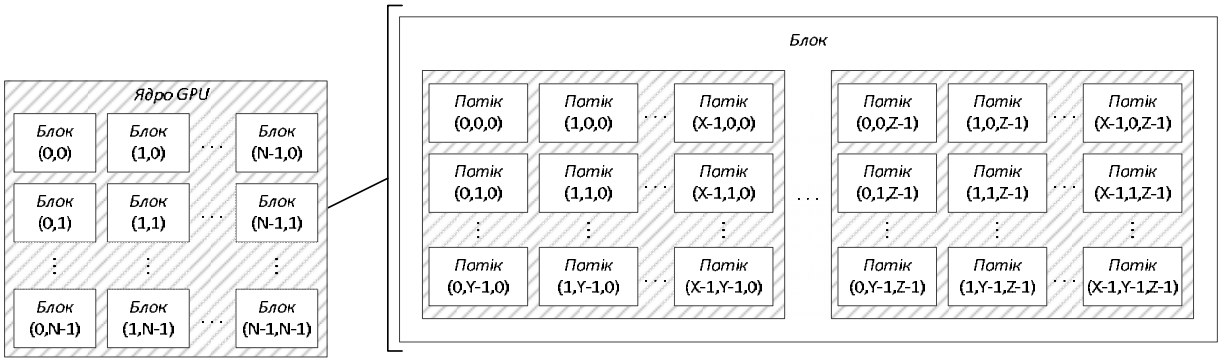

Рис. 1. Обчислювальний пристрій GPU

Особливості архітектури CUDA

Програмна модель архітектури CUDA має такі обмеження:

а) GPU в CUDA – це не самостійний пристрій, а співпроцесор. Постановку завдань, виділення пам'яті, передачу даних між оперативною пам'яттю і пам'яттю GPU виконує хостовий CPU за допомогою драйвера. З цього, зокрема, випливає, що GPU не можуть спілкуватися один з одним або в процесі виконання довантажувати дані з оперативної пам'яті;

б) оскільки виділенням пам'яті повинен обов'язково керувати хост, то в графічному ядрі може використовуватися лише статичний розподіл пам'яті;

в) у GPU немає стека, тому параметри викликів і локальні змінні підпрограм розміщуються в регістрах. Крім того, розміщення параметрів відбувається статично на етапі компіляції – один раз для кожного виклику. З цього випливає факт того, що рекурсія не підтримується;

г) у CUDA не існує понять "код помилки" або "виключення", тому виявлення і формування виняткових станів доводиться реалізовувати вручну;

д) CUDA розглядає три рівні ієрархії пам'яті, кожен з яких має певні інструкції читання/запису, тому покажчики не є прозорими. Крім того, регістрова пам'ять взагалі не може бути адресована динамічно сформованим покажчиком. Знехтувати цим і розміщувати всі дані в глобальній пам'яті недоцільно, оскільки це серйозно погіршить продуктивність.

Обмеження CUDA-пристроїв, які зумовлені способом організації пам'яті

При використанні CUDA-пристроїв необхідно обмінюватися інформацією, яка є як вхідними даними, так і вихідними. Сам CUDA-пристрій може працювати тільки зі своєю пам'яттю. Відповідальність за процес обміну інформацією між пам'яттю CUDA-пристрою і оперативною пам'яттю комп'ютера повністю покладена на CPU. Центральному процесору необхідно до запуску обчислень на CUDA-пристрої зарезервувати необхідні обсяги пам'яті й внести в них вхідні дані, якщо це необхідно. Як тільки CUDA-пристрій закінчить обчислення, то CPU повинен скопіювати вихідні дані з CUDA-пристрою в оперативну пам'ять комп'ютера і звільнити виділені на CUDA-пристрої обсяги пам'яті.

Модель пам'яті CUDA-пристрою показано на рис. 2. CUDA підтримує такі види пам'яті: реєстрову, локальну, загальну, константну, текстурну, глобальну. Для двостороннього обміну інформацією можна використовувати глобальну пам'ять, а для передачі вхідних даних також – константну або текстурну пам'яті. Всі ці типи пам'яті мають досить малу швидкість доступу, тому необхідно використовувати цю пам'ять якомога менше. Реєстрова і локальна пам'ять належить кожному з потоків. До цих типів пам'яті може звертатися тільки потік, якому вони належать. Реєстрова пам'ять найшвидша, але має малий розмір. Локальна пам'ять – це частина глобальної пам'яті, тому швидкість доступу до неї мала. Області цих типів пам'яті розподіляються автоматично – якщо не вистачає регістрової пам'яті, то використовується локальна. Тому не рекомендується зберігати багато інформації в області потоку. Спільна пам'ять в рази швидша від глобальної і є спільною пам'яттю для потоків кожного з блоків.

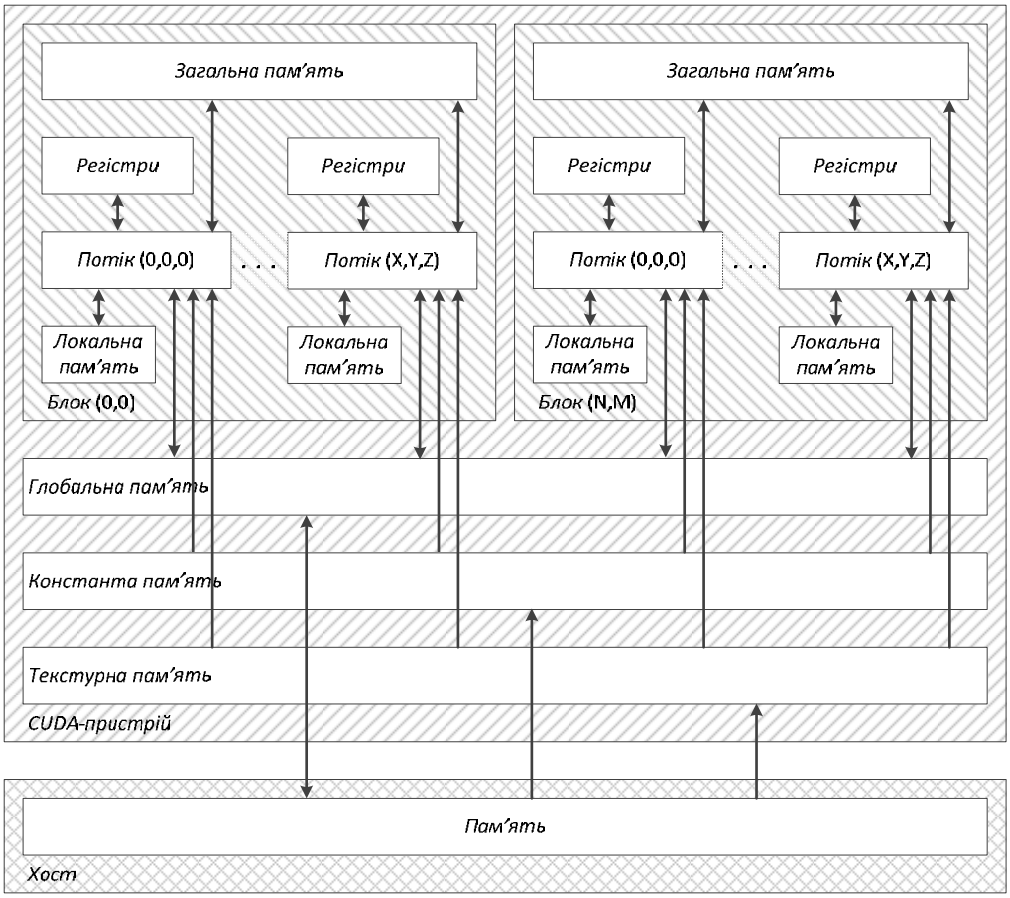

Рис. 2. Модель пам'яті

Розробляючи модулі, які виконуватимуться на CUDA-обчислювачах, необхідно дуже обережно працювати з глобальним, константним і текстурним типами пам'яті. Будь-якому програмному коду, призначеному для виконання на CUDA-пристрої, необхідно, як мінімум, використовувати глобальну пам'ять, тому що тільки тут можна зберігати вихідні дані.

Для реалізації будь-якої задачі з використанням технології CUDA необхідно коректно розбити її на підзадачі. Великі обсяги інформації, які потребують обробки, необхідно розділяти на блоки й обробляти послідовно. Окремим питанням є розмір інформації, з яким повинен працювати потік. Загалом потрібно керуватися принципом: потік не повинен обробляти інформацію, яку швидше можна обробити декількома потоками. Якщо значній частині потоків необхідно працювати з деяким упорядкованим обсягом інформації, то вигідніше розподілити між потоками ці обсяги інформації, зчитати їх відразу за один або декілька етапів і записати в загальну пам'ять, і надалі працювати з інформацією, яка була скопійована, із загальної пам'яті. За умови, що загальна пам'ять більш ніж на порядок швидша від глобальної, то одне копіювання з глобальної пам'яті і кілька зчитувань із загальної здійснюватимуться швидше від декількох зчитувань з глобальної.

Одним з ключових моментів при роботі з CUDA-пристроями є необхідність у вирівнюванні розмірів використовуваних типів даних. Вирівнювання типу даних дозволяє скомпілювати запит в глобальну пам'ять в одну команду GPU, інакше компілятор згенерує додатковий код, який може значно знизити продуктивність. Для оптимальної продуктивності тип даних повинен мати розмірність 4, 8 або 16 байтів.

Куди більший приріст продуктивності можна отримати під час об'єднання більшої кількості запитів у глобальну пам'ять в один, званий транзакцією або об'єднаним доступом до глобальної пам'яті. Для об'єднання запитів у транзакцію необхідно дотримуватися таких умов. По-перше, потоки одного варпа повинні звертатися або до 32-бітових слів, даючи в результаті один 64-байтовий блок, або до 64-бітових слів, даючи один 128-байтовий блок. Якщо використовується звер-

нення до 128-бітових слів, то в результаті буде виконано дві транзакції, кожна з яких поверне по 128 байтів інформації. По-друге, потоки одного варпа повинні звертатися до комірок пам'яті послідовно, кожному наступному потоку повинно відповідати наступне слово в пам'яті, і всі слова повинні бути в межах блока пам'яті, до якого виконується доступ. Але при цьому допускається, щоб деякі потоки взагалі не зверталися до відповідних слів. Другу умову можна інакше переформулювати як необхідність вирівнювання адреси початку пам'яті до розміру блока транзакції. Поняття варп (warp) у цьому випадку означає пул з 32 потоків, у межах якого відбувається паралельна робота. Причому звернення до пам'яті відбувається в межах половини варпа (half-warp), тобто спочатку до пам'яті звертаються перші 16 потоків, а потім другі.

Побудова нейронної мережі з використанням технології CUDA

З урахуванням розглянутих вище особливостей архітектури CUDA виберемо і запропонуємо спосіб реалізації нейронної мережі на графічному ядрі. Як модельовану нейронну мережу візьмемо повнозв'язний персептрон Розенблатта [3] з одним виходом (рис. 3). Такий вибір частково продиктований максимальною простотою мережі цього виду з погляду алгоритму її роботи, а частково – значною обчислювальною складністю, пов'язаною як з обчисленням вихідного сигналу, так і з навчанням мережі. Однією з ключових особливостей вибору виду мережі як найпридатнішої для реалізації на графічному ядрі є її повнозв'язність, тобто наявність зв'язків кожного нейрона *i*-го шару з кожним нейроном (*i* + 1)-го шару для всіх *i*. Повнозв'язність забезпечує регулярність структури мережі, за рахунок чого можна обчислювати розташування вагових коефіцієнтів у масиві без використання додаткового індексного масиву, що задає зв'язки між нейронами. Це істотно знижує кількість звернень до пам'яті, що є одним з вузьких місць в графічному ядрі, а значить, істотно підвищує продуктивність. Повнозв'язність також означає, що потрібно виконувати значну кількість обчислень за доволі невеликої кількості вхідних і вихідних даних. Зокрема, при прямому поширенні сигналу для кожної пари сусідніх шарів потрібно буде виконати приблизно Ni · N(i +1) обчислювальних операцій, де N_i і $N_{(i+1)}$ – кількість нейронів в *i*-му і (*i* +1) - му шарах відповідно.

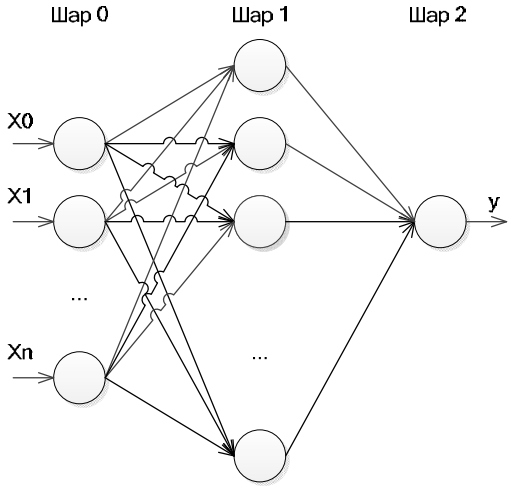

Рис. 3. Модель реалізованої нейронної мережі

Наступною особливістю модельованої нейронної мережі є вимоги до кількості нейронів у кожному шарі. Ці вимоги визначаються розглянутою раніше особливістю архітектури CUDA, що забезпечує під час виконання ряду умов об'єднання в транзакції операцій читання/запису з глобальної пам'яті. Для того, щоб забезпечити умову кратності 128 адрес комірки пам'яті, до якої здійснюється звернення з нульового потоку, необхідно, щоб і кількість нейронів у кожному шарі також була кратною 128. Це пов'язано з тим, що розпаралелювання обчислень в нейронній мережі можливе тільки в межах одного шару, і з тим, що якщо кількість нейронів у шарі перевищує максимально можливу кількість потоків, то доведеться розбивати шар на ряд блоків, обчислюючи їх окремо. У результаті, якщо вимогу кратності кількість нейронів не дотримано, у блоках, відмін-

них від нульового, спостерігатиметься зміщення адреси від значення, кратного 128, а значить, і різке падіння продуктивності. У багатьох випадках вимогу кратності можна пом'якшити за рахунок того, що дані, до яких відбувається звертання, займають не один байт, а 2, 4 або 8. Наприклад, для зберігання вагових коефіцієнтів доцільно використовувати дійсний тип даних одинарної точності, що займає чотири байти. Тоді кількість нейронів має бути кратна 128/4 = 32.

У разі, якщо кількість нейронів у вхідному шарі не відповідає вимозі кратності, є сенс додати відсутні фіктивні нейрони, подавши на їх вхід довільні константні значення. Незважаючи на зростання кількості потрібних обчислень, загальна продуктивність нейронної мережі підвищиться за рахунок об'єднання в транзакції звернень до пам'яті. Кількість нейронів у прихованому шарі, як правило, підбирають дослідно, вона може варіюватися в доволі широких межах зі збереженням властивостей нейронної мережі, тому вимогу кратності в цьому випадку забезпечити достатньо легко. Що стосується вихідного шару, то тут кількість нейронів, як правило, досить мала і визначається специфікою задачі. Додавати значну кількість фіктивних вихідних нейронів нерідко недоцільно, тому необхідно буде побудувати нейронну мережу з використанням технології CUDA так, щоб для вихідного шару зняти вимогу кратності. І для єдиного шару це цілком розв'язувана задача, причому із забезпеченням об'єднання в транзакції звернень до пам'яті. Це можливо завдяки тому, що в попередньому шарі кількість нейронів кратна до 128, а значить, кратною буде і кількість зв'язків між нейронами вихідного і прихованого шару.

З урахуванням розглянутих вище вимог до моделі нейронної мережі пропонуємо спосіб її розбиття на блоки. Для розрахунку поширення сигналу від вхідного шару до прихованого кожен потік обраховує окремий нейрон. Оскільки кожному потоку знадобляться вхідні значення $X = \{x_0, x_1, \ldots, x_n\}$ і оскільки їх кількість порівняно мала, то для прискорення доступу до них ці значення копіюються з глобальної пам'яті в загальну. Якщо кількість нейронів у прихованому шарі перевищує кількість потоків, що одночасно працюють то кожен потік по черзі прорахує значення декількох нейронів з індексами $j,\ j+T,\ j+2T$ і т.д., де j – порядковий номер потоку; а T – загальна кількість потоків. З цього випливає, що кількість потоків слід вибирати так, щоб вона була кратною до кількості нейронів у прихованому шарі.

Оскільки у вихідному шарі кількість нейронів мала порівняно з кількістю потоків, а в такому спрощеному випадку вихідний нейрон всього один, то алгоритм, використовуваний для розрахунку нейронів прихованого шару, буде неефективним. У такому разі пропонується розпаралелювання обчислень. Кожен потік підсумовує значення декількох прихованих нейронів з урахуванням їх ваг. Індекси нейронів для кожного потоку такі самі, як і у випадку розрахунку нейронів прихованого шару. В результаті отримуємо T проміжних сум, які необхідно просто скласти між собою і над отриманим результатом виконати передатну функцію. Складання виконується попарно зі зменшенням кількості потоків, що одночасно працюють, вдвічі на кожному кроці. Оскільки CUDA насправді не надає можливості зупинити незадіяні потоки, то просто перевіряється порядковий номер потоку і, якщо він перевищує кількість потоків, необхідних у цей момент, то ніяких дій він не виконує.

Для навчання мережі використано стандартний метод зворотного поширення помилки [3]. Як складову частину він використовує розглянутий раніше розрахунок мережі в прямому напрямку, після чого отримане вихідне значення порівнюють з бажаним і, починаючи з вихідного шару, вносять корективи у ваги синапсів у напрямку похідної і залежно від величини помилки і швидкості навчання.

Помилка у вихідному шарі обчислюється тільки в одному потоці, а інші потоки в цей час простоюють. Але оскільки це порівняно швидка обчислювальна операція, то втрати в продуктивності за рахунок цього не істотні. Якщо вихідних нейронів більше ніж один, то кількість потоків, що одночасно працюють, можна відповідно збільшити. А далі задіюються всі потоки для розрахунку коректив вагових коефіцієнтів. Для прихованого шару кожний потік обробляє по черзі кожен синапс з індексами j, j + T, j +2T і т.д. А для нейронів вхідного шару використовується додаткова оптимізація. Враховується той факт, що кількість синапсів, а значить, і кількість вагових коефіцієнтів у повнозв'язній мережі велика, отже, в загальній пам'яті їх розмістити не видається можливим. Проте кількість вхідних значень і навіть кількість нейронів у прихованому шарі досить

мала, щоб задіяти для їх зберігання загальну пам’ять. Ну, а оскільки для зміни вагових коефіцієнтів все одно необхідно зчитувати їх значення з глобальної пам’яті, а на наступному кроці навчання мережі обчислювати поширення сигналу в прямому напрямку, то ці два етапи доцільно об’єднати. Тобто, обчисливши нове значення вагового коефіцієнта, одразу ж розрахувати значення на відповідному виході. З урахуванням запропонованого об’єднання етапів дещо змінюється розподіл обчислень по потоках відносно розбиття розрахунків між вихідним і прихованим шаром. У такому разі кожний потік коригує вагові коефіцієнти всіх синапсів, пов’язаних з якимсь нейроном прихованого шару, одночасно підсумовуючи значення на його вході. Якщо кількість нейронів прихованого шару перевищує кількість потоків, кожен потік послідовно обробить подібним способом кілька нейронів. Вибір оброблюваних нейронів, як і у всіх попередніх випадках, диктується необхідністю дотримання умов послідовного звернення до глобальної пам’яті.

У тому випадку, якщо моделюватися буде багатошаровий персептрон, то обчислення при прямому проході буде повністю аналогічним до розглянутого. Але оскільки у ньому є кілька прихованих шарів, то необхідно буде просто повторити розрахунки для кожного шару, задіявши один із запропонованих алгоритмів. Кращим є алгоритм, використаний для розрахунку нейронів прихованого шару в класичному персептроні Розенблатта, оскільки в цьому алгоритмі постійно задіяні всі потоки. Використовувати алгоритм розрахунку нейронів вихідного шару доцільно тільки в тих випадках, коли кількість нейронів у деякому прихованому шарі мала.

Реалізація алгоритму зворотного поширення помилки для багатошарового персептрона повністю аналогічна, за винятком того, що можна буде запам’ятати вихідні значення для подальшого вхідного набору тільки для першого шару, тому перевага у продуктивності буде трохи менш істотною.

Аналіз продуктивності

Для порівняльного аналізу реалізовано моделі нейронної мережі з використанням тільки обчислювальної потужності центрального процесора і графічного ядра за допомогою технології CUDA. Представлена нейронна мережа містила три шари. Прихований шар містив 1024 нейрони, вихідний шар – один нейрон. Кількість нейронів у вхідному шарі змінювалась в діапазоні від 32 до 512. При обчисленні на графічному ядрі у всіх випадках було задіяно 512 потоків. Апаратна платформа для тестування: відеокарта NVidia GeForce GTX 560 Ti, двоядерний центральний процесор Intel Core 6320 1.86 GHz. Результати тестування роботи нейронної мережі в разі обчислення значень у прямому напрямку наведено в табл. 1.

Як видно, час обчислення на центральному процесорі істотно більший, причому він значно зростає зі збільшенням кількості входів. При використанні GPU, по-перше, спостерігається істотний приріст продуктивності, а по-друге, залежність від кількості вхідних значень дуже слабка. Це цілком узгоджується з теоретичними припущеннями, тому в разі зростання кількості входів головно збільшується кількість обчислювальних операцій, які виконуються на графічному ядрі, а обсяг даних, який передається між центральним процесором і ядром, зростає незначно.

Час обчислення вихідного значення

Кількість входів, шт.	Час, мкс		Приріст продуктивності, раз
	GPU	CPU	
32	29.6	1070	36.1
64	30.7	1970	64.2
128	30.3	4520	149.2
256	32.1	9800	305.3
512	34.3	23880	696.2

Тут також можна спостерігати певну особливість: часові витрати при реалізації нейронної мережі на CPU зростають монотонно, тоді як при реалізації на GPU спостерігаються стрибки зміни продуктивності. Наприклад, обробка 64-входової нейронної мережі виконується довше, ніж обробка 128-входової. Подібні ситуації можуть бути пов’язані як з особливостями звертання до комірок загальної пам’яті [2], так і з неоптимальним вибором кількості потоків. Наприклад, в цьому випадку при 512 потоках деяка частина їх простоювала, що особливо помітно зі зменшенням кількості входів.

Загальна картина зростання часових витрат при навчанні мережі, реалізованої на CPU, аналогічна до даних, наведених у таблиці.

Для тестування швидкості навчання нейронної мережі була задіяна 32-входова мережа з 1024 нейронами в прихованому шарі. Кількість потоків графічного ядра варіювалась у діапазоні від 32 до 512. Результати показано на рис. 4. Як видно, час, що витрачається на навчання, зі збільшенням кількості потоків знижується. Однак ця залежність має нелінійний характер, тому що, по-перше, якщо кількість входів менша за кількість потоків, не завжди вдається повністю завантажити їх роботою, а по-друге, навіть за невеликої кількості потоків шина доступу до глобальної пам'яті виявляється досить сильно завантаженою, що і обмежує подальше зростання продуктивності.

Під час подальшого тестування (рис. 5) фіксувалася кількість потоків (512 шт.) і кількість нейронів у прихованому шарі (1024 шт.), а кількість вхідних нейронів змінювалася в діапазоні від 32 до 512. Як видно, зміни в продуктивності спочатку були невеликі. Значне збільшення часу обчислень спостерігалося тільки при 512 входах. Подібна поведінка головно пов'язана з тим, що в разі значного збільшення кількості входів різко зростає не тільки обсяг обчислень, але і кількість вагових коефіцієнтів, які потрібно зчитувати з глобальної пам'яті, що є вузьким місцем в архітектурі CUDA. При незначній кількості входів їх вплив на загальну продуктивність нейронної мережі менший.

На рис. 6 відображено зміну часу навчання у разі зміни кількості нейронів у схованому шарі в діапазоні від 128 до 2048. Кількість входів і кількість потоків при цьому дорівнюють 128. На наведеному графіку видно, що за великої кількості нейронів у прихованому шарі залежність має майже лінійний характер, тоді як при малих значеннях спостерігається ефект, подібний до показаного в табл. 1, коли через неправильно підібрані параметри спостерігається падіння продуктивності.

Окремо протестована залежність продуктивності від кількості зразків у навчальній вибірці (рис. 7). Тут видно, що питома продуктивність знижується зі збільшенням обсягу вибірки. Цей факт пов'язаний з тим, що перед навчанням мережі необхідно передати вхідні дані, бажані вихідні значення і вагові коефіцієнти з ОЗУ в пам'ять графічного ядра.

За малих обсягів вибірки відносні витрати на передачу даних виявляються істотнішими, тому що власне передача даних, якщо їх обсяг невеликий, слабко залежить від обсягу переданих даних, а більшою мірою впливають витрати на виклики системних функцій і підготовчі операції.

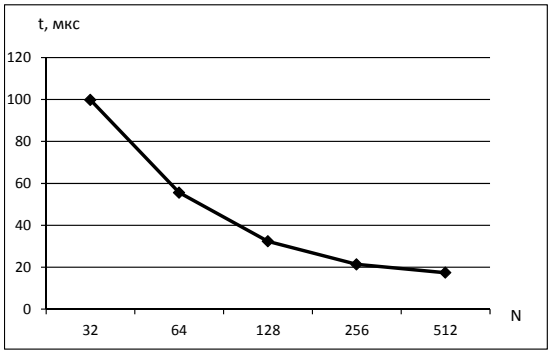

Рис. 4. Залежність часу навчання від кількості потоків

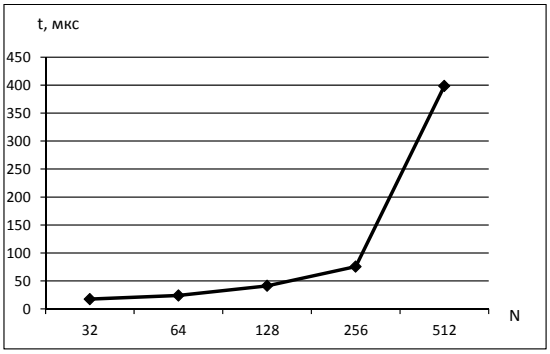

Рис. 5. Залежність часу навчання від кількості входів

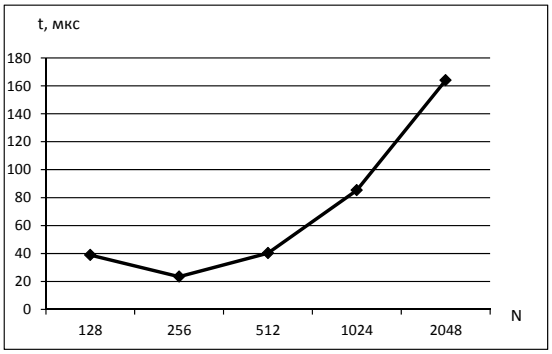

Рис. 6. Залежність часу навчання від кількості нейронів у прихованому шарі

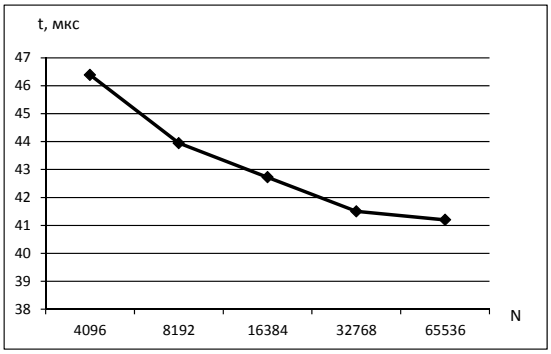

Рис. 7. Залежність часу навчання від кількості зразків у навчальній вибірці

Висновок

Повнозв'язна нейронна мережа успішно може бути змодельована на графічному ядрі загального призначення з використанням технології CUDA. Але оскільки архітектура графічного ядра і відповідна технологія, що дає змогу одержати доступ до його обчислювальних можливостей, накладає ряд обмежень, то їх необхідно врахувати ще на етапі проектування моделі нейронної мережі та на етапі реалізації відповідних алгоритмів.

Виграш продуктивності з переходом на GPU становить приблизно два порядки, що недосяжно у разі використання тільки обчислювальної потужності центрального процесора. І це відкриває широкі можливості для застосування нейронних мереж у системах реального часу.

1. Гергель В.П. Теория и практика параллельных вычислений. – М.: Бином, Лаборатория знаний, 2007. – 424 с. 2. CUDA Zone. http://www.nvidia.ru/object/cuda_home_new_ru.html (Last access: 15.08.2012). 3. Каллан Р. Основные концепции нейронных сетей: пер. с англ. – М.: Вильямс, 2001, – 288 с. 4. Изопов П.Ю., Суханов С.В., Головашкин Д.Л. Технология реализации нейросетевого алгоритма в среде CUDA на примере распознавания рукописных цифр. – М.: Институт систем обработки изображений РАН "Компьютерная оптика", т. 34. – 2010. – № 2. – С. 243–251.

УДК 004.738

Н.Я. Павич, І.П. Костирко *
Національний університет "Львівська політехніка",
кафедра програмного забезпечення,
* кафедра електронних обчислювальних машин

ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ ВЕБ-СОКЕТІВ ДЛЯ АСИНХРОННОЇ КЛІЄНТ-СЕРВЕРНОЇ ВЗАЄМОДІЇ ВЕБ-ПРОГРАМ

Проаналізовано сучасні підходи до реалізації асинхронної клієнт-серверної взаємодії веб-програм. Розглянуто ефективність та недоліки основних реалізацій. Для основних алгоритмів наведено принципи їх функціонування. Запропоновано підхід до застосування технології веб-сокетів для взаємодії в реальному часі.

Ключові слова: веб-програма, AJAX, асинхронна взаємодія, технологія опитування, черга запитів очікування, веб-сокети.

This paper analyzed existed approaches of implementation an asynchronous client-server interaction between/for application. There were considered efficiency and disadvantages of main algorithms and described the principles of their functioning. For web socket technology is described approach for real-time interaction.

Key words: web application, AJAX, asynchronous interaction, Pull technology, Push technology, Long pooling, WebSockets.

Вступ

З розвитком технологій передавання даних і разом з ними Інтернету розвивались і веб-програми – окремий вид клієнт-серверних програм, у котрих клієнтом виступає браузер, а сервером – веб-сервер. Хоч на початку розвитку можливості їх були обмежені, завдяки бурхливому розвитку Інтернету вони відіграють важливу роль у сучасному світі технологій і впевнено входять в повсякденне життя людей. Наприклад, за даними компанії Netcraft, що займається дослідженням Інтернету, кількість сайтів на червень 2000 р. досягла позначки понад сімнадцять мільйонів. Відтак показники (у вересні 2012 року) перетнули межу шістсот двадцять мільйонів веб-програм.