УДК 004.42
DOI: 10.25140/2411-5363-2021-2(24)-131-140

Володимир Русінов, Олексій Череватенко, Леонід Пустовіт, Олександр Пустовіт

МЕТОД ПРИСКОРЕННЯ ВИКОНАННЯ ЗАДАЧ НЕЙРОННИХ МЕРЕЖ НА ГЕТЕРОГЕННИХ СИСТЕМАХ CPU-GPU

У статті розглянуто питання використання гетерогенних систем CPU-GPU для прискорення вирішення задач, пов'язаних із навчанням нейронних мереж. Досліджено основні архітектурні відомості, застосовані в кінцевій архітектурі гетерогенної системи. Розроблено метод на основі лінійної регресії для розподілу задачі між процесором та відеоприскорювачем. На основі методу, проведені експериментальні дослідження на зазначених системах. На основі отриманих результатів був проведений аналіз на основі визначених коефіцієнтів прискорення.

Ключові слова: *гетерогенні системи; нейронні мережі; машинне навчання; CPU; GPU.*
Рис.: 5. Табл.: 3. Бібл.: 11.

Актуальність теми дослідження. Сфера досліджень, що стосуються гетерогенних обчислень, нині недостатньо розвинені. Наприклад, у машинному навчанні величезні зусилля спрямовуються на створення більш ефективних алгоритмів та отримання більшої продуктивності графічних процесорів, що призводить до обмеження використання центрального процесора лише до таких задач з обслуговування ОС, як, наприклад, планування. Незважаючи на це, більшість суперкомп'ютерів і великих центрів обробки даних використовують системи, що мають як центральні процесори (CPU), так і графічні процесори (GPU) [1]. Загальні обчислення на графічному процесорі (GPGPU) відкрили користувачам ПК дорогу для експериментів та роботи над проєктами, які залучають графічні процесори для передачі трудомістких задач. Два приклади бібліотек, які дозволяють це робити – CUDA та OpenCL [2; 11].

Ще одним моментом, який необхідно врахувати, є останні події на ринку процесорів, де 4-ядерні центральні процесори стали стандартними приблизно через 15 років від виходу двоядерних процесорів на ринок, а центральні процесори на рівні підприємств або серверів мають до 64 ядер, що досягає рекомендованої теоретичної межі архітектури SMP. Це, у свою чергу, створює додаткові економічно ефективні способи отримання потужності процесора.

Також зростає попит на використання нейронних мереж у багатьох практичних сферах, оскільки багато великих компаній розробляють інтелектуальні системи, які працюють безпосередньо з рішеннями Big Data. Це пов'язано із задоволенням двох найважливіших вимог до ефективного використання нейронних мереж: якості даних та збільшення обчислювальної потужності. Успіх у використанні нейронних мереж останнім десятиліттям отримав назву «Ренесанс». Цей успіх ще більше множиться широкою доступністю дешевих та ефективних процесорів, графічних процесорів та рішень з відкритим кодом для безпосередньої взаємодії з ними.

Постановка проблеми. Нейронні мережі як спосіб знайти рішення практичних проблем з кожним днем набувають все більшого поширення. У зв'язку з цим, виникає попит на високопродуктивні комп'ютерні системи.

Аналіз останніх досліджень і публікацій. Багато досліджень описують, як працювати з графічними процесорами, які найкращі практики їх використання, як створити переглянуті алгоритми центрального процесора для роботи на графічних процесорах [5]. Процесори також доволі добре вивчені. Однак, як зазначалося раніше, гетерогенні системи не настільки ретельно досліджені, є кілька статей про те, як з ними працювати, однак досконального розуміння у цій галузі немає.

Статті, що стосуються гетерогенних систем, дають перспективні результати [1; 2; 4]. Гетерогенні системи CPU-GPU хоча і недостатньо вивчені, але показали, що прискорюють задачі, пов'язані з машинним навчанням, обробкою зображень та задачами великих даних [1; 2].

Виділення недосліджених частин загальної проблеми. Python – популярна мова програмування, і частина її слави походить від бібліотек машинного навчання, таких як Tensorflow, Scikit та ін. Більшість таких бібліотек оптимізують своє навантаження на графічному процесорі; Tensorflow робить крок далі і знаходить способи краще працювати на TPU (тензорному блоці обробки). Такий підхід є більш обґрунтованим у хмарних рішеннях PaaS, які зазвичай надають кінцевим користувачам високопродуктивний графічний процесор та низькопродуктивні процесори або віртуалізують потужний процесор у кілька віртуальних процесорів. Більшість інших рішень, включаючи власне рішення PaaS, що забезпечують доступ до високопродуктивних процесорів та графічних процесорів, зазвичай не отримують вигоди від цього об'єднання [3].

Мета дослідження. Мета цієї статті – визначити, чи доцільно використовувати гетерогенні системи для вирішення задач нейронних мереж. Для досягнення цієї мети буде проведено набір експериментів, які передбачають хронометраж кожної підзадачі та на основі цього визначають швидкість роботи кожної системи.

Виклад основного матеріалу. Перш ніж заглиблюватися в проблему, потрібно отримати розуміння про обладнання, з яким проводиться робота. Усі експерименти проводяться на графічних процесорах NVidia. Архітектурно графічні процесори досить сильно відрізняються від центральних процесорів. У той час як звичайні клієнти зосереджуються на простих показниках – пам'яті, частоті пам'яті, частоті процесора, професійних клієнтів більше цікавить, що саме входить до таких пристроїв. Для прикладу візьмемо TU102, що використовується в основних графічних процесорах RTX 2080Ti та професійних аналогах Quadro RTX 6000. Ця модель використовується як приклад, оскільки кожна сучасна модель графічного процесора походить саме від цього дизайну. TU102 підрозділяється на 6 кластерів обробки графіки, 36 кластерів обробки текстур і 72 потокових мультипроцесорів (SM). Кожен SM складається з 64 ядер CUDA, 8 Тензор-ядер, реєстрового файлу 256 КБ, 4 одиниць текстури та 96 КБ спільної пам'яті. Крім того, кожне ядро має доступ до 6144 Кб кешу L2. Щоб додати складності цієї конструкції, ядра SM поділяються залежно від їхнього використання, тому замість того, щоб описати їх як ядра CUDA, більш доречним способом їх визначення є те, який тип даних вони обчислюють. Є 64 ядра FP32 і 64 ядра INT32. Тому GPU є дуже спеціалізованим пристроєм, який використовує апаратне прискорення [6].

З огляду на архітектуру, NVidia створила спеціалізоване програмне рішення для доступу до потенціалу графічного процесора під назвою CUDA (Compute Unified Device Architecture). CUDA дозволяє розробникам використовувати графічні процесори NVidia для задач GPGPU. Платформа CUDA надає доступ до набору інструкцій, команд та інструментів паралельного програмування для успішної розробки та запуску обчислювальних ядер. Обчислювальне ядро – це рутина, складена для апаратних прискорювачів, в нашому випадку – графічних процесорів. CUDA широко використовується в різних бібліотеках, хоча офіційно єдиними підтримуваними мовами є C і C ++ [10; 11].

CUDA (Compute Unified Device Architecture) – платформа паралельного обчислення та API розроблена компанією Nvidia. Вона дозволяє розробникам використовувати відеокарти Nvidia для загальних обчислень. Платформа CUDA надає розробнику прямий доступ до системи команд відеокарти та елементів паралельного обчислення, для виконання обчислювальних ядер (compute kernel).

Обчислювальне ядро – основна робоча одиниця, за допомогою якої розробник описує алгоритм. Такий термін використовується не лише для GPU, також він використовується для FPGA, TPU, DSP. Для CUDA, парадигма програмування дуже близько інтегрована з векторними обчисленнями, за основу взято припущення, що виклик ядра виконується одночасно в низці незалежних елементах, що дозволяє паралелізм на рівні даних. Проте

також існують атомарні операції, які можуть бути використані для синхронізації між елементами. Кожен виклик отримує індекси, для 1 або більше розмірностей, за допомогою яких здійснюється адресація даних, або буферизація.

Архітектура GPU відноситься до класу SIMD, тобто на кожне вищезгадане ядро надсилаються дані, над якими за один такт виконується одна операція. Як приклад пропонується оглянути TU102, який лежить в основі mainstream GPU RTX 2080Ti та в професіональному GPU Quadro RTX 6000. Він складається з 6 кластерів графічного процесінгу, 36 кластерів процесінгу текстур та 72 Streaming Multiprocessors (SM). SM складається з 64 CUDA ядер, 8 тенсорних ядер, 256 кілобайт файлу регістру, 4 Texture Units, 96 кілобайт спільної пам'яті. Крім цього, кожне яро має доступ до 6144 кілобайт L2 кешу. На рис. 1 є зображення архітектури.

Рис. 1. Архітектура GPU TU102

У межах гетерогенної системи, відеокарта підключена до материнської плати за допомогою інтерфейсу PCIe, через який ведеться обмін даними між спільною пам'яттю системи та пам'яттю відеокарти. CPU в таких системах мають тип архітектури SMP. Зважаючи на те, що SMP системи хоча і можна масштабувати нескінченно, але у ході масштабування можна отримати ефект перенасичення, що буде заважати приросту ефективності на пізніх етапах масштабування.

Для проведення експериментів із зазначеними системами потрібне просте рішення планування. Простіше кажучи, нам потрібно розподілити роботу між центральним процесором і графічним процесором. Маючи справу із системою CPU-GPU, необхідне розуміння, що час виконання на графічному процесорі не можна передбачити. Перш ніж ми проведемо експерименти, нам потрібно виміряти, скільки часу потрібно центральному процесору та графічному процесору, щоб виконати задачу, і побачити, скільки часу потрібно для відправки даних назад із графічного процесора.

Сама задача включає деякі кроки, які робить модель нейронної мережі під час навчання. Ці кроки – нормалізація, зважування, активація.

Нормалізація – це функція, яка підходить для початкових значень до загальної шкали, в нашому випадку, від 0 до 1. На відміну від процедури нормалізації бази даних, різні нейронні мережі, розроблені для інших сценаріїв, будуть використовувати різні методи нормалізації, залежно від вхідних даних.

Зважування – це процес застосування параметра ваги до вхідних даних шляхом множення. Вага – це коефіцієнт, який постійно змінюється в нейронних мережах під час навчання. Він визначає, як особливість вузла, яку представляє ця вага, впливає на бажаний результат.

Активація використовує спеціальну функцію активації, яка визначає вихідні дані. Існує багато різних функцій активації, для цього випадку ми будемо використовувати функцію гіперболічної дотичної, визначену наступною формулою:

$$f(x) = \tanh(\mathrm{x}) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}. \quad (1)$$

Використовуючи вимірювання часу, ми створюємо просту регресійну модель із використанням методу найменших квадратів.

Особливий інтерес тут представляє формула ефективності паралельної обробки на гетерогенній системі, яка бере за основу метод лінійної регресії. Імперичний підхід до встановлення формули ізоефективності визначає можливість досягнення необхідної ефективності паралельних обчислювальних систем. Така можливість надається за рахунок варіювання параметрами n і N. На основі такого варіювання можна досягти лінійного нарощування продуктивності при збільшенні розмірності задачі. Це означає, що при виконанні обчислювальних процесів можна заздалегідь визначати необхідну ефективність їх реалізації.

Такі системи, в яких ефективність вирішення завдання може задаватися заздалегідь і постійно підтримуватися, називають ізоефективними системами.

Для обчислень ми використовуємо принцип лінійної регресії для того, щоб розподілити розмірність задач між графічним процесором і центральним процесором.

Лінійна регресія є моделлю залежності змінної від однієї або декількох інших змінних або факторів із лінійною функцією залежності.

Уявімо, що нам заданий набор із точок. Метою лінійної регресії є пошук лінії, яка найкращим чином відповідає цим точкам. Загальне рівняння для прямої:

$$f(x) = mx + b, \quad (2)$$

де m – нахил лінії, a b – його *y*-зрушення. Таким чином, рішення лінійної регресії визначає значення для m і b, так що *f(x)* наближається якомога ближче до *y*.

З-поміж декількох ліній необхідно визначити, яка саме найбільше підходить. Це можна зробити, визначивши функцію втрат. Функція втрат є мірою кількості помилок, які лінійна регресія робить на наборі даних. Хоча є різні функції втрат, всі вони обчислюють відстань між передбаченим значенням *y* і його фактичним значенням. З допомогою функціонального позначення можна виділити дистанцію помилки між фактичними і прогнозованими значеннями.

При збільшенні кількості вимірів та змінних постановка завдання залишається тією самою, у двох, трьох або більшій кількості вимірів. Застосовується функція втрат, графік якої виглядає як гіперчаша, де, як і раніше, метою є знайти найнижчу частину цієї чаші, найменше значення об’єктивно, яке функція втрат може приймати, залежно від вибору параметрів і набору даних.

Визначення значення цієї точки ми і робимо за допомогою поширеного підходу – методу найменших квадратів, який вирішує цю задачу аналітично. У нашому випадку наявні лише два виміри та, відповідно, дві змінні у функції, тому обчислення по методу можуть бути виконані вручну.

Час виконання завдання прогнозується за допомогою інформації про раніше виконані завдання того ж типу. На основі запропонованої моделі, розроблений планувальник задач, який працює, використовує вищеописаний принцип ізоефективності, щоб спочатку отримати результат, змодельований із уже виконаних задач для прогнозування часу виконання нових задач. Планувальник аналізує розміри даних та виконання задач, час усіх попередніх задач певного типу та використовує лінійну регресію для визначення кореляції між ними.

За допомогою детермінованої моделі кореляції можна передбачити час виконання задачі того ж типу з використанням розміру її даних як вхідного параметра.

Першим кроком є початковий розподіл за такою формулою:

$$N_{gpu} = W_{gpu} * N, \quad (3)$$

де N – розмір задачі, а W_{gpu} – обчислювальна вага. Для визначення обчислювальної ваги використовується така формула:

$$W_{gpu} = \frac{t_{gpu}}{t_{gpu} + t_{cpu}}, \quad (4)$$

де t_{gpu} – час виконання на графічному процесорі, t_{cpu} – час виконання на центральному процесорі, а W_{gpu} – обчислювальна вага графічного процесора.

Для кількісної оцінки прискорень ми вводимо два коефіцієнти: коефіцієнт прискорення відносно центрального процесора та коефіцієнт прискорення відносно графічного процесора – оскільки ми досліджуємо гетерогенну систему, центральний та графічний процесори працюють по-різному. Для розрахунку коефіцієнта прискорення відносно центрального процесора ми використовуємо таку формулу:

$$K_{pg} = \frac{T_g + T_s}{T_h}, \quad (5)$$

де T_g – час завершення задачі на графічному процесорі, T_s – час відправлення даних із графічного процесора на центральний процесор, а T_h – час виконання задачі на гетерогенній системі.

Для обчислення коефіцієнта прискорення на графічному процесорі ми використовуємо таку формулу:

$$K_{pg} = \frac{T_c}{T_h}, \quad (6)$$

де T_c – час виконання задачі на центральному процесорі, а T_h – час виконання задачі на гетерогенній системі.

Щоб отримати ці результати, програма запускалась 100-1000 разів, залежно від того, наскільки різними були результати. Кінцеві результати – це в середньому 100-1000 вимірювань часу, зроблених під час виконання.

Як уже зазначалося, метою цієї статті є з'ясувати, чи це можливо і наскільки добре вона працює для задач нейронної мережі, а найпопулярнішою мовою для інженерії даних та машинного навчання є Python. Numba для Python – це бібліотека, яка надає доступ до JIT-компілятора, який може компілювати код Python в обчислювальних ядрах [11].

Результати. Тестування проводиться на таких комп'ютерних системах:

1. AMD Ryzen 9 3900X 12 ядер, 24 потоки, 3.8 GHz базова частота, 4.1 GHz розігнана частота, RTX 2060, 16GB RAM, чипсет X570.
2. AMD Ryzen 5 2400G 4 ядра, 8 потоків, 3.6 GHz базова частота, 3.8 GHz розігнана частота, GTX 1060-3 GB, 16 GB RAM, чипсет A320.
3. AMD Athlon 760K 4 ядра, 4 потоки, 3.8 GHz базова частота, 4.1 GHz розігнана частота, GTS 450, 4 GB RAM, чипсет X89.
4. Intel Core i5-7200U 2 ядра, 4 потоки, 2.5 GHz базова частота, 3.1 GHz розігнана частота, Geforce 940MX, 8 GB RAM, чипсет KBU.
5. Intel Xeon E5-2630 2 ядра, 2 потоки, 2.3 GHz базова частота, 2.8 GHz розігнана частота, Tesla K80, 8 GB RAM, чипсет C604.

На наступних графіках (рис. 2-4) показано час виконання на графічному процесорі та центральному процесорі, а також час відправки даних із графічного процесора до центрального процесора.

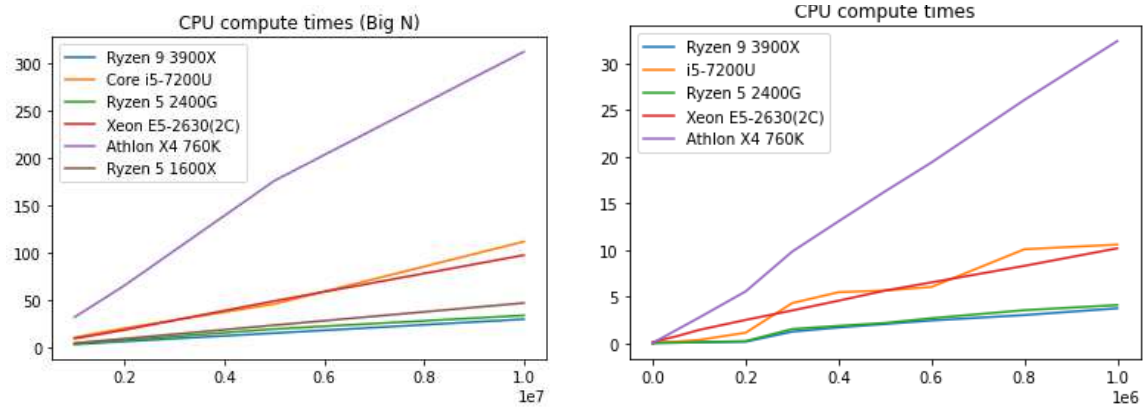

Рис. 2. Час виконання на центральному процесорі (CPU)

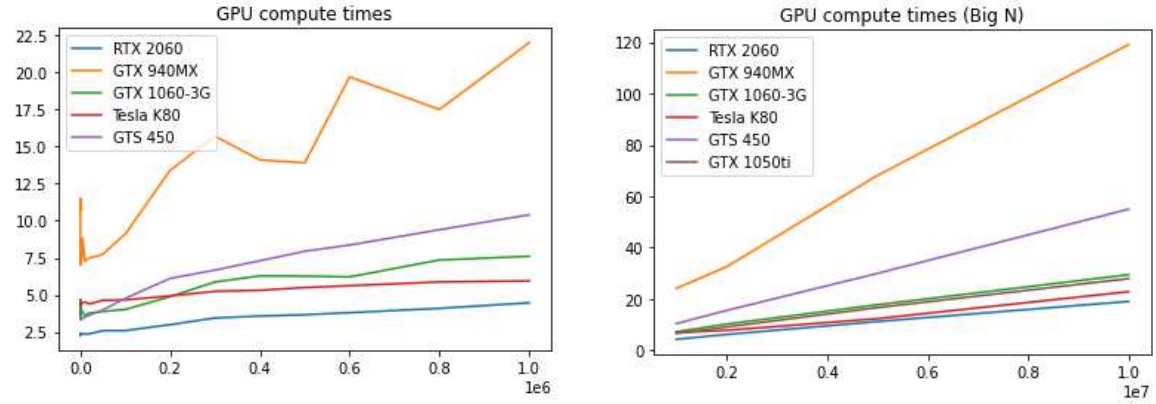

Рис. 3. Час виконання на графічному процесорі (GPU)

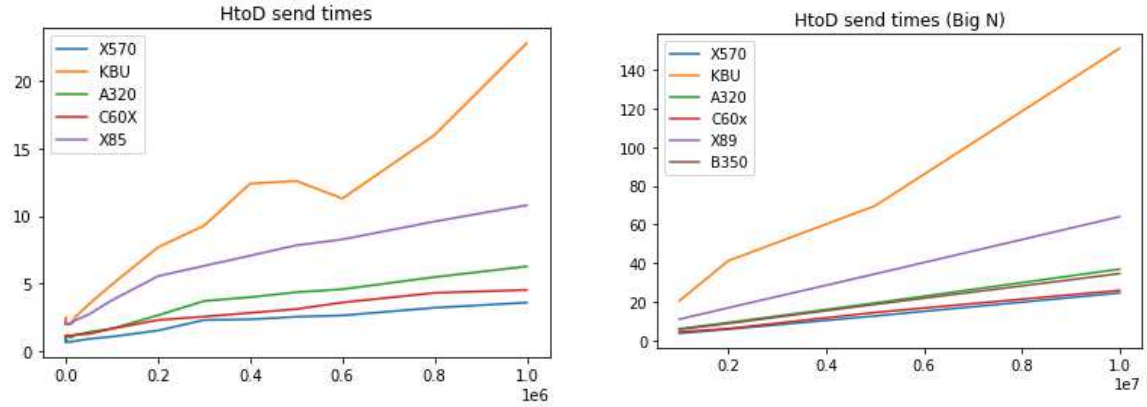

Рис. 4. Час відправки даних з GPU на CPU

Наступний крок – взяти модель регресії та знайти значення для N_{gpu} та N_{cpu}, обчислене на минулому кроці. Таким чином ми отримуємо нову пару прогнозів часу, за допомогою яких робимо новий розподіл. Цей крок повторюється 10 разів, щоб отримати більш «справедливий» розподіл. Після розподілу отримуємо наступні результати:

Таблиця 1 – Розподіл роботи між GPU та CPU

Розмір	GPU	CPU
1	2	3
R9 3900X + RTX 2060		
5000000	2098724	2901276
10000000	4302225	5697775
25000000	10783998	14216002
50000000	21500424	28499576

Закінчення табл. 1

1	2	3
R5 2400G + GTX 1060-3GB		
5000000	1897180	3102820
10000000	3319339	6680661
25000000	9503751	15496249
50000000	18664993	31335007
I5 7200U + Geforce 940MX		
5000000	1708896	3291104
10000000	3308685	6691315
25000000	7836838	17163162
50000000	15244193	34755807
Xeon E5-2630(2C) + Tesla K80		
5000000	2112889	2887111
10000000	6198501	3801499
25000000	16353979	8646021
50000000	33230526	16769474
Athlon X4 760k + GTS 450		
5000000	3301774	1698226
10000000	6930421	3069579
25000000	17943324	7056676
50000000	36351454	13648546

Таблиця 2 – Коефіцієнт прискорення на графічному процесорі

Система	5000000	1000000	25000000	5000000
R9 3900X + RTX 2060	1.73	2.01	2.18	2.36
R5 2400G + GTX 1060-3GB	1.8	2.14	2.59	2.77
I5-7200U + Geforce 940MX	2.94	3.66	4.06	4.39
Athlon X4 760k + GTS 450	1.16	1.25	1.25	1.26
Xeon E5-2630 (2C) + Tesla K80	1.11	2.82	2.71	3.01

Таблиця 3 – Коефіцієнт прискорення на центральному процесорі

Система	5000000	1000000	25000000	5000000
R9 3900X + RTX 2060	1.12	1.39	1.58	1.73
R5 2400G + GTX 1060-3GB	0.96	1.1	1.43	1.55
I5-7200U + Geforce 940MX	0.99	1.51	1.66	1.81
Athlon X4 760k + GTS 450	3.17	3.28	3.43	3.48
Xeon E5-2630 (2C) + Tesla K80	1.59	1.72	2.61	2.81

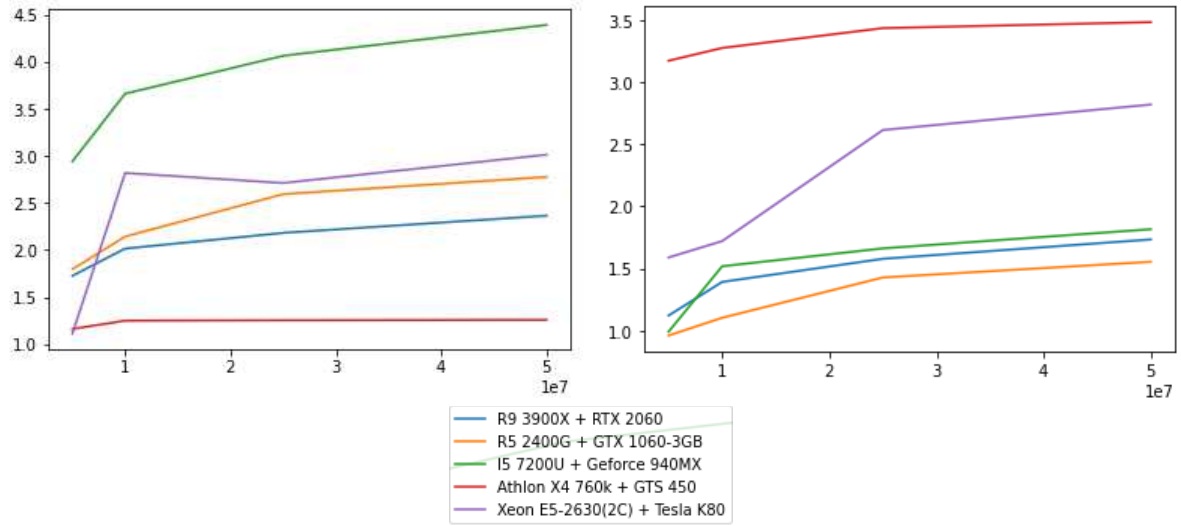

Рис. 5. Коефіцієнти прискорення на гетерогенній системі

Висновок. У статті вивчались можливості для прискорення нейромережевих підзадач, таких як нормалізація, зважування та активація, використовуючи комбінований обчислювальний потенціал центрального та графічного процесорів, що працюють паралельно. Результати показують, що гетерогенний підхід ефективний у задачах нейронної мережі. Застосування цього підходу значно скоротило загальний час виконання.

З результатів ми можемо зробити висновок, що програма використовує переваги нових процесорів і прискорення продуктивності графічного процесора в діапазоні від 1,11 до 4,39, а також прискорення продуктивності центрального процесора в діапазоні від 0,96 до 3,48 показують, що в більшості випадків продуктивність центрального процесора може бути покращена, якщо в цей же час використовується графічний процесор.

У деяких прогонах підхід з використанням лише центрального процесора показує кращі результати, оскільки коефіцієнт іноді опускається нижче 1 і досягає 0,96, однак у більшості експериментів цього не відбувається. Це може бути пов'язане з тим, графічному процесору потрібна певна кількість часу, щоб відправити назад до процесора всі обчислені дані. Передача даних сама по собі займає 40-60 % часу. Графіки показують, що навіть одне число з плаваючою комою може зайняти до 6 мілісекунд для обробки графічним процесором через синхронний характер архітектури графічного процесора.

Список використаних джерел

1. Chen Tianqi, Li Mu, Li Yutian, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems, 2015.

2. Stone J. E., Gohara D., Shi G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science & Engineering*. May-June 2010. Vol. 12, No. 3. Pp. 66-73. DOI: 10.1109/MCSE.2010.69.

3. Nurvitadhi E., Sim Jaewoong, Sheffield D., Mishra A., Krishnan S., Marr D. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. *2016 26th International Conference on Field Programmable Logic and Applications (FPL).* Lausanne, 2016. Pp. 1-4. DOI: 10.1109/FPL.2016.7577314.

4. Van Werkhoven Ben, Jason Maassen, Frank Seinstra, Henri Bal. Performance models for CPU-GPU data transfers. *Proceedings - 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. CCGrid 2014. DOI: 10.1109/CCGrid.2014.16.

5. Kim Y., Mercati P., More A., Shriver E., Rosing T. P4: Phase-based power/performance prediction of heterogeneous systems via neural networks. *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Irvine, CA, 2017. Pp. 683-690. DOI: 10.1109/ICCAD.2017.8203843.

6. GPU-Accelerated Applications. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/gpu-applications-catalog.pdf.

7. NVidia Turing GPU Architecture. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

8. Canqun Yang, Feng Wang, Yunfei Du, Juan Chen, Jie Liu, Huizhan Yi, Kai Lu. Adaptive Optimization for Petascale Heterogeneous CPU/GPU Computing. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*. 2010. Pp. 19-28. DOI: 10.1109/CLUSTER.2010.12.

9. Hestness J., Keckler S. W., Wood D. A. GPU Computing Pipeline Inefficiencies and Optimization Opportunities in Heterogeneous CPU-GPU Processors. *2015 IEEE International Symposium on Workload Characterization*. Atlanta, GA, 2015. Pp. 87-97.

10. Soyata T. GPU Parallel Program Development Using CUDA. New York: CRC Press, 2018.

11. CUDA Refresher: Reviewing the Origins of GPU Computing. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

References

1. Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyuan, Zhang, Zheng. (2015). MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems.

2. Stone, J. E., Gohara, D., & Shi, G. (May-June 2010). OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. In *Computing in Science & Engineering*, *12*(3), 66-73. DOI: 10.1109/MCSE.2010.69.

3. Nurvitadhi, E., Sim, Jaewoong, Sheffield, D., Mishra, A., Krishnan, S., & Marr, D. (2016). Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. *26th International Conference on Field Programmable Logic and Applications (FPL)* (pp. 1-4). Lausanne. DOI: 10.1109/FPL.2016.7577314.

4. Van Werkhoven, Ben, Maassen, J., Seinstra, F., & Bal, H. (2014). Performance models for CPU-GPU data transfers. Proceedings *14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2014*. 10.1109/CCGrid.2014.16.

5. Kim, Y., Mercati, P., More, A., Shriver, E., & Rosing, T. (2017). P4: Phase-based power/performance prediction of heterogeneous systems via neural networks. *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (pp. 683-690). Irvine, CA. DOI: 10.1109/ICCAD.2017.8203843.

6. *GPU-Accelerated Applications*. (2020). NVIDIA Corporation.

7. *NVidia Turing GPU Architecture*. (2018). NVIDIA Corporation.

8. Yang, Canqun, Wang, Feng, Du, Yunfei, Chen, Juan, Liu, Jie, Yi, Huizhan, Lu, Kai. (2010). Adaptive Optimization for Petascale Heterogeneous CPU/GPU Computing. *Proceedings IEEE International Conference on Cluster Computing, ICCC* (pp. 19-28). 10.1109/CLUSTER.2010.12.

9. Hestness, J., Keckler, S. W., Wood, D. A. (2015). GPU Computing Pipeline Inefficiencies and Optimization Opportunities in Heterogeneous CPU-GPU Processors. *2015 IEEE International Symposium on Workload Characterization* (pp. 87-97). Atlanta, GA.

10. Soyata, T. (2018). GPU Parallel Program Development Using CUDA. New York: CRC Press.

11. *CUDA Refresher: Reviewing the Origins of GPU Computing*. (2018). NVIDIA Corporation.

UDC 004.42

Volodymyr Rusinov, Oleksii Cherevatenko, Leonid Pustovit, Oleksandr Pustovit

METHOD OF ACCELERATION OF NEURAL NETWORK TASKS ON HETEROGENEOUS CPU-GPU SYSTEMS

Research areas related to heterogeneous computing are currently underdeveloped. In machine learning, huge efforts are directed towards creating more efficient algorithms and getting more performance from GPUs. There is also a growing demand for the use of neural networks in many practical areas, since many large companies are developing intelligent systems that work directly with Big Data solutions. General purpose of GPU computing hands GPUs time-consuming tasks, however there is limited research on the capabilities of heterogeneous systems.

Neural networks as a way to find solutions to practical problems are becoming more widespread every day. In this regard, there is a demand for high-performance computer systems.

Articles dealing with heterogeneous systems provide promising results. Heterogeneous CPU-GPU systems, although not well understood, have been shown to speed up tasks associated with image processing and big data tasks.

Most modern machine learning articles and libraries perform tasks on CPUs or GPUs, however, heterogeneous systems have not been sufficiently investigated for possibilities in machine learning tasks.

The aim of the study is to develop an algorithm for scheduling a problem for a heterogeneous system and to carry out a comparative analysis of the results on the use of homogeneous systems.

An algorithm for scheduling a problem for a heterogeneous system based on a linear regression model is described, the problem which is modeled is shown and a comparative analysis of the results of various heterogeneous systems is carried out.

The results show that the heterogeneous approach is effective in neural network tasks. From the results we can conclude that the program takes advantage of new processors and GPU performance acceleration in the range of 1.11 to 4.39, as well as CPU performance acceleration in the range of 0.96 to 3.48 show that in most cases, CPU performance can be improved if the GPU is used at the same time.

Keywords: *heterogeneous systems; neural networks; machine learning; CPU; GPU.*

Fig.: 5. Table: 3. References: 11.

Русінов Володимир Володимирович – студент, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
Rusinov Volodymyr – Student, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (37 Pobedy Av., 03056 Kyiv, Ukraine).
E-mail: volodymyr.r.v@ukr.net
ORCID: https://orcid.org/0000-0002-4362-0248

Череватенко Олексій Володимирович – студент, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
Cherevatenko Oleksii – Student, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (37 Pobedy Av., 03056 Kyiv, Ukraine).
E-mail: chereva@ukr.net
ORCID: https://orcid.org/0000-0001-9686-0555

Пустовіт Леонід Михайлович – студент, Київський національний університет імені Тараса Шевченка (вул. Володимирська, 64/13, м. Київ, 01601, Україна).
Pustovit Leonid – Student, Department of General Physics, Taras Shevchenko National University of Kyiv (64/13, Volodymyrska Street, City of Kyiv, Ukraine, 01601).
E-mail: lrerednawl@gmail.com
ORCID: https://orcid.org/0000-0002-4606-1971

Пустовіт Олександр Михайлович – студент, Київський національний університет імені Тараса Шевченка (64/13 Volodymyrska Str., 01601 Kyiv, Ukraine).
Pustovit Oleksandr – Student, Taras Shevchenko National University of Kyiv (64/13 Volodymyrska Str., 01601 Kyiv, Ukraine).
E-mail: PuSaMy1998@gmail.com
ORCID: https://orcid.org/0000-0001-6659-4786

Русінов В., Череватенко О., Пустовіт Л., Пустовіт О. Метод прискорення виконання задач нейронних мереж на гетерогенних системах CPU-GPU. *Технічні науки та технології.* 2021. № 2(24). С. 131-140.