УДК 004.89, 004.048 **doi: 10.26906/SUNZ.2025.3.136-141**

Д. В. Сальніков, О. Г. Васильченков, Д. Г. Караман

Національний технічний університет «Харківський політехнічний інститут», Харків, Україна

ОПТИМІЗАЦІЯ МНОЖЕННЯ КВАНТОВАНИХ ОДНОБІТНИХ МАТРИЦЬ ДЛЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Анотація. У зв'язку з активним розвитком та вдосконаленням систем штучного інтелекту останнім часом обробка природної мови стала однією з найбільш актуальних та затребуваних задач. Засоби та алгоритми на базі великих мовних моделей, що забезпечують обробку природної мови та перетворення мови в текстові данні, активно застосовуються для реалізації методів автоматизації різноманітних повсякденних задач, а також систем обслуговування та оперативної взаємодії з людиною. Для швидкого та адекватного опрацювання природної мови, з врахуванням синтаксичних та національних особливостей необхідно використовувати досить складні мовні моделі. Але великі мовні моделі потребують значних обсягів пам'яті та обчислювальної потужності, що ускладнює їх повсякденне використання на пристроях з обмеженими ресурсами, таких як мобільні пристрої з автономним живленням, вбудовані системи та пристрої Інтернету речей. Таким чином, оптимізація алгоритмів роботи мовних моделей та зниження апаратних витрат на їх реалізацію є як ніколи актуальною задачею. Для пришвидшення виконання та зменшення необхідних об'ємів пам'яті використовують алгоритми квантування коефіцієнтів мовних моделей. В даній роботі сформульовано проблеми, що виникають під час виконання квантованих операцій множення матриць, розглянуто популярні підходи до реалізації алгоритму множення матриць на графічних прискорювачах, та реалізовано оптимізоване за швидкістю виконання ядро множення квантованих 1-бітних матриць.

Ключові слова: квантовані операції, множення матриць, трансформери, великі мовні моделі, CUDA, GPU, LLM, Pytorch, нейронні мережі.

Вступ

Великі мовні моделі (LLM, Large Language Models) є однією з найзначніших інновацій в галузі штучного інтелекту і обробки природної мови. Вони здатні аналізувати, розуміти і генерувати змістовний текст, що робить їх надзвичайно корисними для виконання різних завдань, пов'язаних з обробкою природної мови. Однак, щоб забезпечити ефективну роботу таких моделей, необхідно вирішити низку технічних проблем, пов'язаних з виконанням матричних операцій. Матричні операції є ключовим компонентом в архітектурі нейронних мереж, особливо в трансформерах, які є основою багатьох сучасних LLM, таких як GPT-3 і BERT.

Одна з основних проблем матричних операцій для великих мовних моделей полягає у складності процесу їх обчислення. Матричні операції, як-от матричні множення та обчислення власних векторів, потребують значних обчислювальних ресурсів, особливо коли йдеться про обробку великих обсягів даних. До прикладу, у [1] описано основи архітектури трансформерів, які використовують матричні операції для обробки послідовностей даних. Іншою важливою проблемою є не достатньо ефективне використання пам'яті. Великі мовні моделі потребують значної кількості оперативної пам'яті для зберігання проміжних результатів матричних обчислень. Це стає особливо критичним у випадках, коли модель має обробляти довгі послідовності тексту або великі обсяги даних. Докладно з переліком основних методів оптимізації використання пам'яті у великих мовних моделях можна ознайомитися у роботі [2].

Також варто звернути увагу на питання обчислювальної потужності та енергоспоживання. Виконання складних матричних операцій вимагає високої обчислювальної потужності, що призводить до значного енергоспоживання. Це піднімає питання про екологічну стійкість та вартість експлуатації великих мовних моделей. Автори роботи [3] приводять дослідження аспектів енергоефективності та обчислювальної потужності сучасних нейронних мереж.

Окрім технічних аспектів, важливим є питання масштабування матричних операцій. Зі зростанням розмірів мовних моделей і збільшенням обсягів даних, які вони обробляють, зростає потреба у більш ефективних методах масштабування обчислень. У [4] проведено аналіз різних підходів до масштабування мовних моделей і його впливу на ефективність обчислень. Одним із напрямків вирішення цих проблем є розробка спеціалізованого апаратного забезпечення, такого як графічні процесори (GPU) та тензорні процесори (TPU, Tensor Processing Units), які оптимізовані для виконання матричних операцій. У [5] надаються особливості та переваги використання TPU для виконання матричних обчислень у нейронних мережах. Ще одним важливим напрямком є оптимізація алгоритмів для матричних операцій. Використання сучасних алгоритмів та методів оптимізації може значно знизити обчислювальну складність та підвищити ефективність матричних обчислень. У [6] розглянуто способи оптимізації матричних операцій у контексті глибоких нейронних мереж.

Таким чином, вирішення наведених проблем матричних операцій є ключовим аспектом для розвитку та розширення сфер використання великих мовних моделей. Поєднання апаратних і програмних методів дозволяє забезпечити ефективну роботу мовних моделей, знижуючи обчислювальну складність, енергоспоживання та обсяги необхідної пам'яті. Це відкриває нові можливості для застосування великих мовних моделей у різних галузях: медицині, освіті, бізнесі та виробництві, а також використання у кіберфізичних системах та пристроях з обмеженими ресурсами, таких як автономні роботи, вбудовані системи та пристрої Інтернету речей.

Вирішення цих викликів вимагає тісної співпраці між дослідниками, розробниками апаратного забезпечення та інженерами-програмістами. Лише за умови комплексного підходу можливо досягти значних успіхів у оптимізації матричних операцій та забезпечити подальший розвиток великих мовних моделей. Подальші дослідження у цій галузі допоможуть виявити нові методи та підходи, що сприятимуть підвищенню ефективності і надійності мовних моделей, а також пошуку шляхів для їх реалізації для використання у сучасних кіберфізичних системах та пристроях Інтернету речей.

Основною метою статті є розробка та дослідження ефективності оптимізованого за швидкістю виконання реалізації ядра множення квантованих 1-бітних матриць з урахуванням оптимальних алгоритмів множення матриць на графічних прискорювачах.

Множення матриць з квантованими коефіцієнтами

Більшість конкурентних нейронних моделей перетворення послідовностей мають структуру енкодер-декодер. У такій архітектурі енкодер відображає вхідну послідовність символів $(x_1,...,x_n)$ у послідовність безперервних представлень $z = (z_1,...,z_n)$. На основі z декодер генерує вихідну послідовність символів $(y_1,...,y_m)$ по одному елементу за раз. На кожному кроці модель працює в авторегресивному режимі, використовуючи раніше згенеровані символи як додаткове введення для прогнозування наступного символу. Трансформер реалізує цю архітектуру, використовуючи багатошаровий механізм формування уваги та покомпонентні повнозв'язні шари як у енкодері, так і в декодері.

Функція уваги визначається як відображення запиту та набору пар «ключ-значення» у вихідний вектор, де запит, ключі, значення та вихідні дані представлені у вигляді матриць. Вихідна матриця обчислюється як зважена сума значень, де ваги визначаються за допомогою функції сумісності між запитом і відповідним ключем. На практиці функція уваги обчислюється одночасно для цілого набору запитів, які об'єднуються в матрицю Q. Аналогічно, ключі та значення організовуються у матриці K і V відповідно. Схема обчислення функції уваги зображена на рис. 1. Роботи схеми можна описати виразом (1). Вираз включає в себе декілька операцій множення матриць. Зазвичай такі операції є одними з найбільш оптимізованих, оскільки задача множення матриць дуже часто є першочерговим об'єктом розв'язку в області високопродуктивних операцій.

$$Att(Q,K,V) = softmax\left(Q \times K^T \Big/ \sqrt{d_k}\right) \times V, \qquad (1)$$

де Q, K та V — це матриці запитів, ключів та значень, відповідно, а d_k — це розмірність кожного вектора ключа/запиту.

Завдяки можливості розпаралелювати множення матриць, сучасні графічні процесори (GPU) або спеціалізовані пристрої, такі як TPU, можуть значно пришвидшити обчислення, виконуючи безліч операцій одночасно.

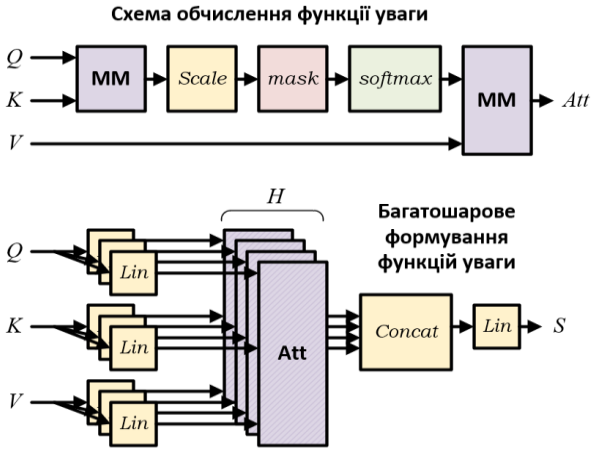


Рис. 1. Схема обчислення функції уваги

На основі виразу (1) можна зробити висновок, що сама модель трансформатора не є складною в обчислювальному плані. Проте для підвищення точності отримуваних результатів вона має багаторівневу структуру. Таким чином, складність моделі зростає із зростанням кількості рівнів уваги моделі.

Із збільшенням якості роботи нейромережевих моделей зростає також і кількість обчислень, необхідних для їх функціонування, а також об'єм пам'яті для зберігання результатів обчислення та вагових коефіцієнтів моделі. Сучасні графічні прискорювачі мають від 80 Гб (Nvidia H100) до 141 Гб (Nvidia H200) оперативної пам'яті, що придатна для зберігання даних, коефіцієнтів та результатів обчислень, тоді як персональні комп'ютери та мобільні телефони обмежені 16-64 Гб. В той самий час, лише тільки об'єм вагових коефіцієнтів топології великої мовної моделі GPT-4 перевищує 300 Гб. Оскільки кількість оперативної пам'яті є суттєвим обмежуючим фактором сучасних обчислювальних засобів, закономірно виникає необхідність стискання даних якими оперують нейронні мережі – квантування. Квантування — це метод стиснення моделей, який перетворює ваги та коефіцієнти активації великих мовних моделей із формату представлення з високою точністю у формат з меншою точністю. Це означає зміну типу даних на такий, що містить менше інформації. Типовим прикладом є кодування 32-бітних чисел з плаваючою комою на певному діапазоні у 8-бітні цілі числа, або компресія зображень, що передаються мережею для відображення на веб-сторінках.

Зменшення кількості бітів, необхідних для представлення ваг або коефіцієнтів активації моделі, суттєво скорочує її загальний розмір. Як наслідок, квантування дозволяє зменшити обсяг пам'яті, необхідний для зберігання моделі, знизити вимоги до сховища та підвищити енергоефективність великих мовних моделей. Квантування великої мовної моделі (LLM) також знижує її вимоги до засобів обчислення, що дає змогу запускати її на менш потужному апаратному забезпеченні, зберігаючи при цьому прийнятний рівень продуктивності. Більшість сучасних нейронних мереж можуть бути квантовані для зменшення об'ємів пам'яті, що використовується, без суттєвого зниження точності роботи алгори-

тму. Алгоритми для квантування представляють коефіцієнти у вигляді співвідношення (2):

$$Coef = Val \times Scale + Bias . \quad (2)$$

Параметри Scale та Bias вибирають один на певний об'єм даних залежно від задачі в якій він використовується. Основні алгоритми квантування розглянуті в [7–10]. Оскільки зберігання параметрів та коефіцієнтів в стислому вигляді є необхідним для більшості сучасних алгоритмів нейронних мереж, постає задача ефективного виконання операції множення матриць з різними типами даних.

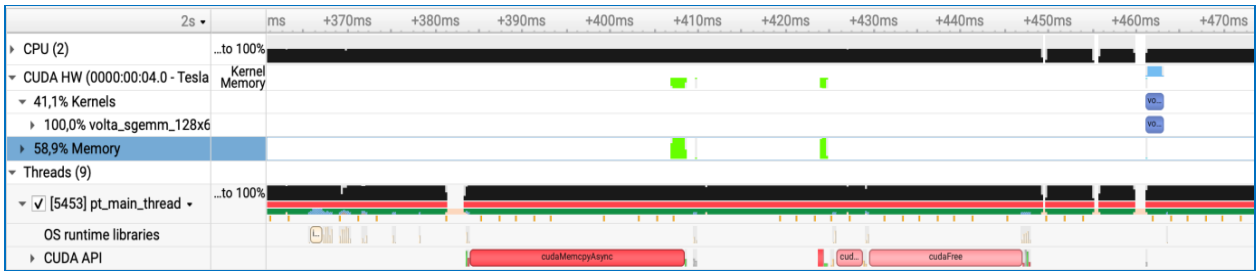
Одним з перспективних, на поточний момент, видів квантування є метод квантування, описаний у роботах [11–14]. Автори пропонують використання так званого посттренувального квантування для створення моделей з екстремально малою кількістю бітів для подання значень ваг та коефіцієнтів активації. Ця методика знижує точність представлення ваг та активацій, що суттєво зменшує вимоги до пам'яті та обчислювальних ресурсів LLM. Поточна тенденція полягає в поступовому переході від 16-бітних форматів до ще нижчих розрядностей, наприклад 4-бітних. Хоча посттренувальне квантування не можна назвати оптимальним рішенням, воно широко використовується в LLM для практичного застосування у різних галузях. Останні дослідження у сфері 1-бітних архітектур [11] відкривають перспективний напрям для зниження вартості використання LLM без значних втрат продуктивності. Типові мовні моделі оперують 16-бітними числами з плаваючою комою (FP16 або BF16), а їхня основна обчислювальна складність пов'язана з операціями множення матриць. Основна частина витрат припадає на додавання та множення чисел з плаваючою комою. Натомість, у запропонованому рішенні множення матриць виконується лише за допомогою операцій цілочисельного додавання, що дозволяє зменшити енергоспоживання на порядки. Зменшення енергоспоживання безпосередньо сприяє прискоренню обчислень. Окрім обчислювальних витрат, значним фактором є передача параметрів моделі з віддаленої пам'яті (DRAM) до пам'яті внутрішнього прискорювача (SRAM) під час виконання запитів. Спроби збільшити обсяг SRAM для покращення пропускної здатності призводять лише до значного зростання вартості та технологічної складності виготовлення обчислювача.

Порівняно з моделями повної точності, 1-бітні LLM мають суттєво знижене споживання пам'яті як з точки зору обсягу, так і пропускної здатності. Це значно скорочує витрати часу та ресурсів на завантаження ваг із DRAM, що, у свою чергу, сприяє швидшому та ефективнішому виконанню запитів. Такий екстремальний вид квантування дає не тільки високі показники стискання, а й не знижує суттєво якість роботи мовної моделі.

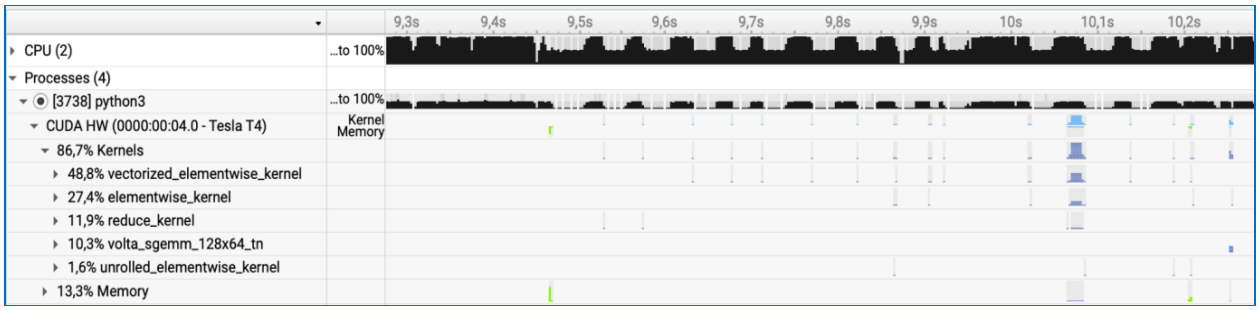
Тим не менше з точки зору швидкості виконання подібний підхід призводить до ускладнення алгоритму розрахунків. Наприклад, рівняння деквантування з урахуванням бітових операцій виглядатиме таким чином:

$$Coef = \left(\left(Val \,\&\, mask\right) \gg \log(mask)\right) \times Scale + Bias . \quad (3)$$

Еталонна реалізація нейромережевого шару використовує примітиви бібліотеки PyTorch, що дає змогу швидко реалізувати та випробувати алгоритм але не є оптимальним. Зокрема з результатів профайлінгу (рис. 2 та табл. 1) видно, що для розрахунків використовується стандартне ядро volta_sgemm_128 та операції зсуву, які виконуються процесором системи. Ядра множення матриць CUDA (зокрема volta sgemm 128) сильно оптимізовані для досягнення найбільшої можливої швидкодії. Проте вони спроектовані для розрахунків в межах одного типу даних — float32, bfloat, hfloat і т. п.



а



б

Рис. 2. Результати профайлінгу роботи алгоритму за допомогою Nsight Compute:
а – операції трансферу даних між процесором та GPU; б – ядра GPU використані для реалізації шару

Таблиця 1 – **Кількісні результати профайлінгу роботи алгоритму**

Назва	CPU, мс	CUDA, мс	CUDA середнє	Викликів
aten::mm	615.19	10539.00	1.054 мс	10000
volta_sgemm_128x64_tn	0.00	10525.00	1.054 мс	10000
native::elementwise_kernel 1	0.00	1424.00	71.196 мкс	20000
aten::mul	221.29	731.01	73.101 мкс	10000
native::elementwise_kernel 2	0.00	731.01	73.101 мкс	10000
aten::sub	246.81	724.06	72.406 мкс	10000
aten::add_	228.42	699.87	69.986 мкс	10000
aten::copy_	1368.00	540.07	6.751 мкс	80000
native::unrolled_elementwise_kernel	0.00	540.07	6.751 мкс	80000
aten::bitwise_and	4829.00	263.67	3.296 мкс	80000
native::vectorized_elementwise_kernel	0.00	263.67	3.296 мкс	80000
aten::rshift__	2203.00	222.69	3.181 мкс	70000

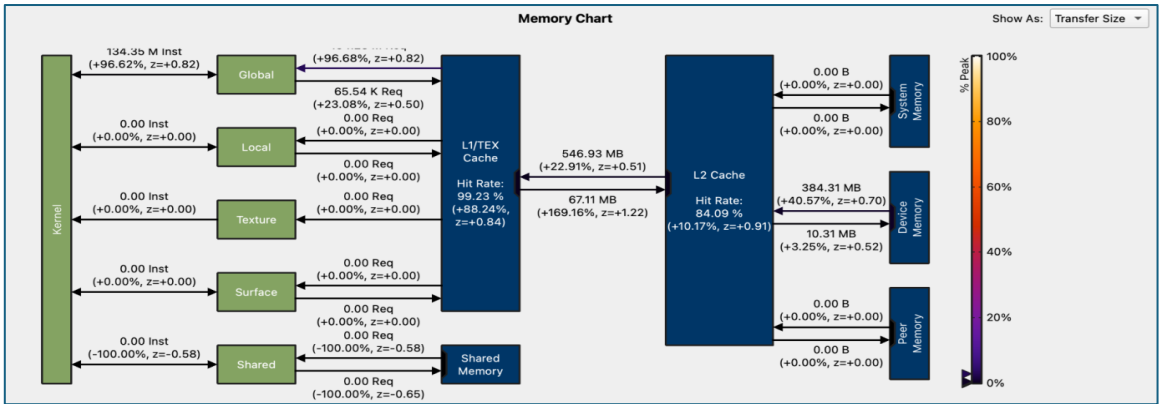
Більше того сучасні графічні прискорювачі не мають апаратної підтримки цілих типів даних менше 8 біт. Через це використовується операція розпаковки даних з однобітного формату в один з форматів з рухомою комою. Така операція використовує операції з накладення бітових масок та зсуву, після чого відбувається множення на Scale та додавання Bias. Якщо ці операції виконує процесор системи, дані потім необхідно скопіювати до пам'яті графічного прискорювача. Таким чином, не зважаючи на те що алгоритм множення матриць є добре оптимізованим, більшу частину часу квантований шар моделі виконує операції перетворення однобітних коефіцієнтів у 32-бітні числа з рухомою комою.

Методи оптимізації алгоритму

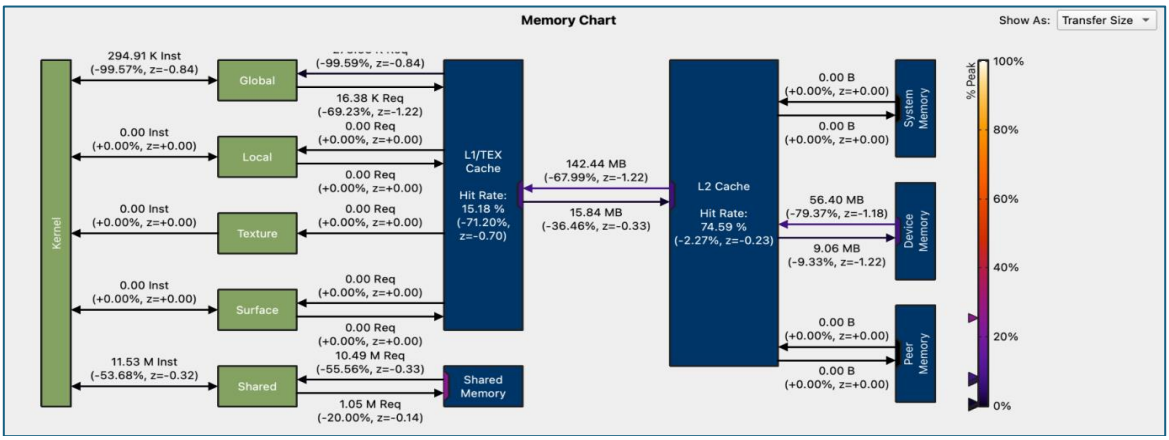
Питання оптимізації обчислень специфічних для нейромережевих алгоритмів розглянуто в роботах [15, 16].

Слід зазначити, що на транзакції пам'яті з головної системи до пристрою та назад, а також на виконання операцій розпаковки даних у формат з рухомою комою витрачається суттєвий відсоток часу роботи алгоритму.

Враховуючі це, навіть не оптимізований алгоритм множення матриць, що не виконує такі трансфери даних є більш оптимальним з точки зору швидкості виконання (рис. 3).



а



б

Рис. 3. Статистика транзакцій між пам'яттю та ядром при множенні матриць:
а – статистика транзакцій пам'яті не-оптимізованого ядра; б – статистика транзакцій пам'яті оптимізованого ядра

Алгоритм множення матриць є досить простим з точки зору розрахунків. Як видно з візуалізації на рис. 4, кожен рядок матриці A множиться на кожен стовпець матриці B, результати накопичуються та зберігаються в матрицю результату C.

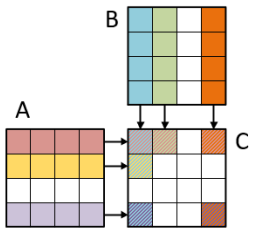


Рис. 4. Візуалізація алгоритму множення матриць

У випадку, що розглядається, матриця A має тип float32, в той час як матриця B – тип 1-bit представлений у тензорі uint8. З точки зору оптимізації програмного забезпечення, найбільш проблемною частиною є організація доступу до пам'яті параметрів алгоритму. Сучасні контролери пам'яті забезпечують певну оптимізацію послідовних транзакцій зчитування та запису. Таким чином, необхідно організувати процес завантаження для мінімізації повторних операцій з пам'яттю та максимізації послідовного доступу. Для додаткової оптимізації було використано кеш пам'ять прискорювача. Як показано на рис. 5 кеш пам'ять GPU організована як N банків пам'яті розміром M.

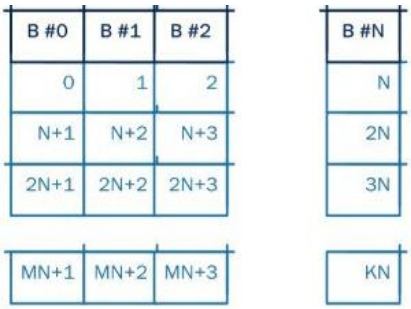


Рис. 5. Структура кеш пам'яті GPU

Виконання декількох операцій зчитування різними SM-процесорами одночасно призводить до затримок обробки, у той час як зчитування/запис з/до різних банків виконується за 1 такт. Отже, в запропонованому рішенні реалізовано алгоритм множення тайлами, де кожен рядок матриці A множиться на декілька стовпців матриці B; 32 елементи рядку зчитуються та зберігаються до кеш пам'яті; кожен SM-процесор GPU використовує окремий банк пам'яті; проміжні результати зберігаються до кеш-пам'яті; наприкінці алгоритму виконується синхронізація SM-процесорів та збереження результату до HBM-пам'яті.

Як видно з рис. 5, даний підхід дозволяє суттєво знизити кількість транзакцій до зовнішньої пам'яті (б) порівняно з наївним алгоритмом (а). Слід зазначити що, кількість звернень до кеш-пам'яті при цьому збільшується, однак саме висока швидкість доступу до кешу як раз і забезпечує суттєве прискорення алгоритму. Реалізований алгоритм досягає 1.7 TFLOPS/s з теоретично можливих 2.9 TFLOPS/s, а отже залишається суттєвий простір для подальшої оптимізації обчислювального ядра.

Результати оцінки тривалості виконання операцій для запропонованого ядра у порівнянні з бібліотекою HQQ представлені на рис. 6.

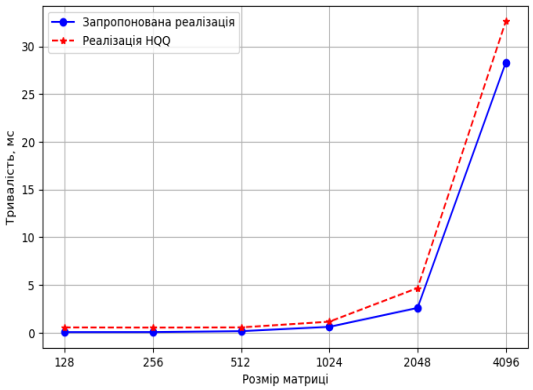


Рис. 6. Тривалість виконання квантованого ядра множення матриць

Висновки

Ядра множення матриць із квантованими параметрами широко використовуються в галузі машинного навчання, зокрема при реалізації великих мовних моделей, завдяки високій обчислювальній ефективності та оптимальному використанню пам'яті.

Розроблене в цій роботі ядро множення матриць із квантованими входами є оптимальнішим як за часом обчислень, так і за апаратними витратами, досягаючи продуктивності класичних ядер множення матриць CUDA, що працюють із єдиним форматом даних для всіх операндів. Крім того, запропоноване ядро зменшує потребу в пам'яті, оскільки відсутня необхідність у зберіганні розпакованих коефіцієнтів у пам'яті пристрою.

Подальші дослідження будуть зосереджені на реалізації підтримки інших типів даних, спеціалізації алгоритму, а також підвищенні швидкодії за рахунок використання векторних операцій.

СПИСОК ЛІТЕРАТУРИ:

1. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. “Attention Is All You Need,” arXiv:1706.03762, Computation and Language (cs.CL); Machine Learning, 2023. DOI: 10.48550/arXiv.1706.03762.
2. Shoeybi M., Patwary M., Puri R., LeGresley P., Casper J., Catanzaro B. “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism,” arXiv:1909.08053, Computation and Language, 2020. DOI: 10.48550/arXiv.1909.08053.
3. Patterson D., Gonzalez J., Le Q., Liang C., Munguia L.-M., Rothchild D., So D., Texier M., Dean J. “Carbon Emissions and Large Neural Network Training,” arXiv:2104.10350, Computation and Language, 2021. DOI: 10.48550/arXiv.2104.10350.
4. Kaplan J., McCandlish S., Henighan T., Brown T. B., Chess B., Child R., Gray S., Radford A., Wu J., Amodei D. “Scaling Laws for Neural Language Models,” arXiv:2001.08361, Machine Learning (cs.LG); Machine Learning (stat.ML), 2020. URL: https://arxiv.org/abs/2001.08361; DOI: 10.48550/arXiv.2001.08361.

5. Jouppi N. P., Young C., Patil N., Patterson D. “In-Datacenter Performance Analysis of a Tensor Processing Unit,” arXiv:1704.04760, Hardware Architecture (cs.AR); Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE), 2017. DOI: 10.48550/arXiv.1704.04760.
6. Cao Y., Romero J., Olson J. P., Degroote M., Johnson P. D., Kieferová M., Kivlichan I. D., Menke T., Peropadre B., Sawaya N. P. D., Sim S., Veis L., Aspuru-Guzik A. “Quantum Chemistry in the Age of Quantum Computing,” Chemical Reviews, vol. 119 (19), ISSN 1520-6890, 2019, pp. 10856–10915. DOI: 10.1021/acs.chemrev.8b00803.
7. Zhou Y., Moosavi-Dezfooli S.M., Cheung N. M. and Frossard P. “Adaptive quantization for deep neural network.” In Proceedings of the 32th AAAI Conference on Artificial Intelligence (Vol. 32, No. 1), 2018. DOI: 10.1609/aaai.v32i1.11623.
8. Kodali R. K., Upreti Y. P., Boppana L. “A Quantization Approach for the Reduced Size of Large Language Models.” 16th International Conference on Knowledge and Smart Technology (KST), Krabi, Thailand, 2024, pp. 144-148, DOI: 10.1109/KST61284.2024.10499664.
9. Krishnamoorthi R. “Quantizing deep convolutional networks for efficient inference: A whitepaper.” arXiv:1806.08342, Machine Learning (cs.LG); Computer Vision and Pattern Recognition (cs.CV); 2018. DOI: 10.48550/arXiv.1806.08342.
10. Elias F., Ashkboos S., Hoefler T., Alistarh D. “Gptq: Accurate post-training quantization for generative pretrained transformers,” arXiv:2210.17323, Machine Learning (cs.LG), 2022. DOI: 10.48550/arXiv.2210.17323.
11. Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, Furu Wei. “The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits,” arXiv:2402.17764 (preprint), Computation and Language (cs.CL); Machine Learning (cs.LG), 2024. DOI: 10.48550/arXiv.2402.17764.
12. Chee J., Cai Y., Kuleshov V., De Sa C. “QuIP: 2-bit quantization of large language models with guarantees,” arXiv:abs/2307.13304, Machine Learning (cs.LG); Computation and Language, 2023. DOI: 10.48550/arXiv.2307.13304.
13. Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, Song Han. “AWQ: activation-aware weight quantization for LLM compression and acceleration,” arXiv:abs/2306.00978, 2023. DOI: 10.48550/arXiv.2306.00978.
14. Chetlur S., Woolley C., Vandermersch P., Cohen J., Tran J., Catanzaro B., Shelhamer, E. “cuDNN: Efficient primitives for deep learning,” arXiv:1410.0759, Neural and Evolutionary Computing (cs.NE); 2014. DOI: 10.48550/arXiv.1410.0759.
15. Kumar R., Negi K. C., Sharma N. K., Gupta P. “Deep Learning-Driven Compiler Enhancements for Efficient Matrix Multiplication.” Journal of Computers, Mechanical and Management, 3(2), 2024. pp.08-18. DOI: 10.57159/gadl.jcmm.3.2.240122.
16. Xiao G., Yin C., Zhou T., Li X., Chen Y., Li, K. “A Survey of Accelerating Parallel Sparse Linear Algebra.” ACM Computing Surveys, 56 (1), 2023. Article No.: 21, pp. 1-38. DOI: 10.1145/3604606.

Received (Надійшла) 17.03.2025
Accepted for publication (Прийнята до друку) 02.07.2025

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Сальніков Дмитро Валентинович – кандидат технічних наук, старший викладач автоматики та управління в технічних системах, Національний технічний університет «Харківський політехнічний інститут», Харків, Україна;
Dmytro Salnikov – Candidate of Technical Sciences, Senior Lecturer at the Department of Automation and Control in Technical Systems, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine;
e-mail: dmytro.salnikov@khpi.edu.ua; ORCID Author ID: https://orcid.org/0009-0007-6201-5370;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57225126629.

Васильченков Олег Георгійович – кандидат технічних наук, доцент, доцент автоматики та управління в технічних системах, Національний технічний університет «Харківський політехнічний інститут», Харків, Україна;
Oleg Vasylchenkov – Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Automation and Control in Technical Systems, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine;
e-mail: oleh.vasylchenkov@khpi.edu.ua; ORCID Author ID: https://orcid.org/0000-0002-0969-2248;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57225129501.

Караман Дмитро Григорович – старший викладач автоматики та управління в технічних системах, Національний технічний університет «Харківський політехнічний інститут», Харків, Україна;
Dmytro Karaman – Senior Lecturer at the Department of Automation and Control in Technical Systems, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine;
e-mail: dmytro.karaman@khpi.edu.ua; ORCID Author ID: https://orcid.org/0000-0002-7252-3172;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57202228956.

Optimization of 1-bit quantized matrix multiplication for large language models

Dmytro Salnikov, Oleg Vasylchenkov, Dmytro Karaman

Abstract. With the rapid development and improvement of artificial intelligence systems, natural language processing has recently become one of the most relevant and in-demand tasks. Tools and algorithms based on large language models (LLMs) that enable natural language processing and speech-to-text conversion are actively used for automating various everyday tasks, as well as for service systems and real-time human interaction. Efficient and accurate natural language processing, taking into account syntactic and linguistic peculiarities, requires highly complex language models. However, large language models demand significant memory and computational resources, making their widespread use challenging on resource-constrained devices such as battery-powered mobile devices, embedded systems, and Internet of Things (IoT) devices. Thus, optimizing language model algorithms and reducing hardware costs for their deployment is an increasingly pressing issue. To accelerate execution and minimize memory requirements, quantization algorithms for language model parameters are employed. This work formulates key challenges associated with performing quantized matrix multiplication operations, explores popular approaches to implementing matrix multiplication algorithms on GPUs, and presents an optimized high-performance kernel for 1-bit quantized matrix multiplication.

Keywords: quantized operations, matrix multiplication, transformers, large language models, CUDA, GPU, LLM, PyTorch, neural networks.