УДК: 004.89

ОПТИМІЗАЦІЯ АЛГОРИТМУ ПОБУДОВИ НЕВІД'ЄМНОЇ МАТРИЧНОЇ ФАКТОРИЗАЦІЇ У ЗАДАЧАХ ОБРОБКИ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ ІЗ ВИКОРИСТАННЯМ ГРАФІЧНИХ ПРОЦЕСОРІВ

О.О. Марченко, В.К. Кисенко, І.О. Березань

Київський національний університет імені Тараса Шевченка
Україна, м. Київ, проспект Академіка Глушкова, 4-Б

У роботі представлена реалізація алгоритму побудови невід'ємної матричної факторизації у задачах обробки текстів природноїю мовою із використанням графічних процесорів. Збільшення швидкодії досягається за рахунок використання паралельної архітектури сучасних графічних процесорів. Отримано значне прискорення початкового алгоритму (більш ніж на один порядок).

An implementation of the non-negative matrix factorization algorithm for the purpose of text mining on graphics processing units is presented. Performance gains of more than one order of magnitude are obtained.

Вступ

Задача автоматичного виділення важливої інформації з тексту постає у багатьох галузях. Методи розв'язку цієї задачі досить добре розвинені, проте на практиці існує необхідність знаходження шляхів часової оптимізації, зокрема, за допомогою використання альтернативних обчислювальних потужностей.

У цій роботі розглядаються шляхи пришвидшення реалізації алгоритму побудови невід'ємного розкладу матриці (Non-negative matrix factorization – NMF) [1] за допомогою обчислювальних можливостей сучасних відеоадаптерів (Graphical Processing Unit – GPU). Також розглядається застосування цього розкладу до кластеризації документів. На відміну від традиційних центральних процесорів (Central Processing Unit – CPU), обчислювальні потужності сучасних відеоадаптерів вже здатні забезпечувати до 10^{12} операцій з плаваючою точко за секунду (FLOP/*s*). Наприклад, сучасний графічний адаптер Radeon HD 6970 від AMD має 3072 потокових процесори, а тому здатен досягти обчислювальної продуктивності на рівні 5.4 TFLOP/*s* у арифметиці з одинарною точністю.

Гнучкість архітектури сучасних відеоадаптерів дозволяє застосувати їх обчислювальні потужності при розв'язанні задач, які напряму не пов'язаних з графікою. Обчислення на графічних процесорах (GPGPU) здобули останнім часом велику популярність, опубліковано ряд робіт з результатами у різних важливих практичних задачах, що свідчать про досягнення значного приросту швидкодії [2] .

На даний час існує два основних фреймворки, які виступають як платформи для розробки програм GPGPU. Обидва фреймворки надають спеціалізовану мову програмування обчислень для відеоадаптерів (такі програми називають – «кернел») та прикладний програмний інтерфейс для доступу і керування окремими обчислювальними пристроями та їх елементами.

Перший фреймворк має назву CUDA [3]. Це запатентована технологія від NVIDIA, що спеціально розроблена для їх графічних процесорів. Існує багато досліджень присвячених розв'язкам обчислювальних задач за допомогою CUDA. Попри успішну оптимізацію і значний ступінь розробок у даному напрямку, варто зазначити обмеженість їхнього практичного застосування: такі програми можуть виконуватись лише на відеоадаптерах одного виробника. Другим фреймворком, що забезпечує платформу для обчислень є OpenCL [4]. На відміну від CUDA, OpenCL є безплатним відкритим стандартом, який підтримується не лише графічними процесорами NVIDIA, а і графічними процесорами майже усіх інших виробників, а також багатоядерними процесорами і спеціалізованим устаткуванням (таким як цифрові сигнальні процесори (DSP) та FPGA). Крім того, важливим аргументом на користь якості OpenCL є те що його розробкою займається Khronos Group.

Варто зазначити, що продуктивне використання обчислювальної потужності GPU можливе лише, у випадку, якщо реалізація базового алгоритму забезпечує значний рівень паралелізації на елементарні операції, що можуть бути виконані одночасно. Звичайні, призначенні для послідовного виконання алгоритми, часто мають бути у значній мірі модифіковані, щоб досягти прискорення від паралелізації на GPU.

Ця проблема частково (на рівні базових функцій) вирішується застосуванням вже розроблених бібліотек. У нашій задачі для реалізації паралельного алгоритма побудови невід'ємного розкладу матриці необхідні паралельні реалізації основних алгебраїчних операцій [5]. Подібних бібліотек існує декілька, в основному на технології CUDA: CUBLAS, MAGMA.

У нашій роботі ми використовуємо бібліотеку ViennaCL (Vienna Computing Library) [6], що написана на мові C++ із застосуванням технології OpenCL. Її вагомою перевагою є наявність зручних засобів реалізації

власних обчислень за допомогою відеоадаптерів та виконання їх на широкому діапазоні різних графічних процесорів.

Різні аспекти проблеми кластеризації текстів, і зокрема, їх зв'язок з декомпозицією матриць розглядаються у розділі 2 цієї роботи. Формальний опис алгоритму побудови невід'ємного розкладу матриці проведено у третьому розділі, тоді як деталі його паралельної реалізації приведено в четвертому. В п'ятому розділі описано та проаналізовано результати, а у шостому наведені висновки.

Кластеризація документів

Кластеризація це процес розбиття набору об'єктів на підмножини (кластери), в такий спосіб, що об'єкти всередині кластера більш схожі (в сенсі деяких властивості або характеристик) між собою, ніж до об'єктів з інших кластерах. Існує багато методів кластеризації [7], більшість з них потрапляють в одну з наступних категорій: ієрархічна, центроїдна (*k-means*, *k-medoids*), статистична (EM-алгоритм), графова (спектральні) кластеризація.

У цій роботі розглядається застосування кластеризації до групування документів за темами. Формально, для заданого набору з *m* текстових документів, метою кластеризації за темами є знаходження такого поділу документів на групи, що документи з одного кластера мають спільну тему. Важливо відзначити, що список тем, не є відомим до початку процесу кластеризації. Таким чином, цей процес може бути розцінений як автоматична категоризації документів та виділення тем із них.

Традиційні методи кластеризації документів часто засновані на векторно-частотній моделі (Vector Space Model – VSM). У цій моделі документ представляється як вектор частот слів. Нехай $T = \{t_1,....,t_n\}$ – набір слів з усіх документів. Тоді частотний вектор X_i документа d_i формується як $X_i = \{x_{1i},....,x_{ni}\}$, причому

$$x_{ij} = f_{ij} \log \frac{m}{c_j}, \tag{1}$$

де f_{ij} позначає частоту терміну t_j у документі d_i. Кількість документів у яких присутній терм t_j, позначається як c_j.

Таким чином, VSM використовує слова, як міру схожості документів. Після введення такої моделі, очевидним чином визначаються метрики на частотних векторах, наприклад, косинусна відстань або Евклідова метрика. Завдяки представленню документів у термінах лінійної алгебри, до них можуть бути застосовані традиційні методи кластеризації.

Одним з основних недоліків VSM, є припущення, про те що слова є статистично незалежними між собою. Однак, це часто не так для реальних текстів. Теми, концепції та семантика є ключовими ознаками документа. Для отримання таких особливостей з текстів були розроблені спеціальні методи, відомі під загальною назвою "виділення ознак" (feature extraction). Метою таких методів є виділення основних концепцій (або тем) текстів, та подання документів у вигляді їх комбінації. Такі методи успішно застосовуються у задачах обробки природно мовних текстів, таких як: кластеризація документів, знаходження семантичної відстані тощо.

Матричні розклади, такі як сингулярний розклад матриці (відомий як під назвою «латентний семантичний аналіз») [8] або невід'ємна матричний розклад (NMF) [9], часто використовуються як основа таких методів. У подальшому, у даній роботі, розглядається тільки NMF.

Загалом, NMF є розкладом (або факторизацією) матриці V в добуток двох матриць W і H. Причому, на елементи матриць V, W, H накладаються додаткове обмеження невід'ємності. Важливо відмітити, що такий розклад не є обов'язково унікальним і матриці W та H не повинні бути ортогональними.

Ця факторизація матриць стала популярною після публікації статті Лі і Суна [1]. У цій роботі запропоновано два алгоритми побудови невід'ємного розкладу матриці. Після цього було розроблено ряд інших методів, таких як: метод проективного градієнту [10], метод чергованих невід'ємних квадратів та метод розрідженого кодування.

У цій роботі використовується оригінальний метод Лі і Суна, ще відомий під назвою «мультиплікативні правила оновлення», як цільова функція використовується норма Фробеніуса.

В обробці природно мовних текстів NMF зазвичай застосовується до терм-документ матриці (TD-матриці). Кожен рядок TD-матриці відповідає одному документу, а кожен стовпець матриці відповідає одному слову. Якщо задано *m* документів і загальна кількість слів у них рівна *n*, то розмір TD-матриці V рівний $m \times n$. NMF використовується для розкладу матриці V у добуток двох матриць W і H з розмірами $m \times k$ і $k \times n$ відповідно, де зазвичай $k << \min(m,n)$. Наприклад, k може бути встановлено в значення очікуваної кількості кластерів. Часто, параметр k називають – «кількість ознак».

Інтуїтивно, параметр k може бути пояснений наступним чином: TD матриця V («документи» × «слова») розкладається у добуток двох матриць. Матриця W пов'язує «документи» і «ознаки», а матриця H пов'язує «ознаки» і «слова». Тому, як випливає з властивостей добутку матриць, кожен документ представляється у вигляді лінійної комбінації виділених ознак. Після цього, традиційні методи кластеризації, можуть бути застосовані до рядків матриці W.

У цілому повний процес кластеризації виглядає наступним чином:

- виділити документи в словах.
- побудувати TD-матрицю V.
- нормалізувати стовпці матриці V.
- застосувати NMF: $V = WH$.
- побудувати кластеризацію на основі матриць W і H.

NMF, як засіб обробки природної мови, має кілька переваг над іншими методами виділення ознак. По-перше, матриці W і H мають тільки позитивні елементи, що робить процес їх інтерпретації, у термінах обробки природної мови, простішим і логічнішим. По-друге, стовпці W не повинні бути ортогональними, а отже виділені теми (ознаки) можуть мати спільні риси, що є досить звичайним для реальних документів.

Алгоритм NMF

У цьому розділі наведено алгоритм NMF, що був запропонований Лі і Суном. Як цільова функція використовується норма Фробеніуса, що записується так:

$$\min_{W,H} \| V - WH \|_F^2 , \tag{2}$$

причому елементи матриць W та H повинні бути невід'ємними.

Для такої цільової функції, та для двох початкових матриць W_0 і H_0, NMF алгоритм складається з ітераційного виконання двох кроків:

$$(H_k)_{i,j} = (H_{k-1})_{i,j} \times \frac{(W_{k-1}^T V)_{i,j}}{(W_{k-1}^T W_{k-1} H_{k-1})_{i,j}}, \tag{3}$$

$$(W_k)_{i,j} = (W_{k-1})_{i,j} \times \frac{(V H_{k-1}^T)_{i,j}}{(W_{k-1} H_{k-1} H_{k-1}^T)_{i,j}}. \tag{4}$$

На практиці, кроки алгоритму повторюються доки не буде досягнута нерухома точка або доки не буде пройдена максимальна кількість ітерацій. Лі і Сун довели дві основні властивості цього алгоритму: по-перше, цільова функція є монотонно спадною під час застосування наведених правил; по-друге, матриці W і H стають постійними, тільки у випадку досягнення стаціонарної точки цільової функції. Крім того, важливо відмітити, що цей метод знаходить тільки локальні мінімуми, а не глобальні. Також, важливо ще раз зазначити, що такий розклад не є унікальним.

Паралельна реалізація алгоритму NMF

Алгоритм NMF, що наведено у попередньому розділі може майже одразу бути реалізований за допомогою процедур, що надаються у бібліотеці ViennaCL. У ньому використовуються два види матричних операцій: множення матриці на матрицю та по-елементні дії над матрицями. Перший тип операцій вже є реалізованим у ViennaCL, а другий має бути розроблений додатково за допомогою створення ядер OpenCL. Далі наведено псевдокод обчислення NMF:

```
for(unsigned int it = 0; it < MAX_NUM_OF_ITER; it++) {
     hn   = prod(trans(W), V);         //numerator matrix for H
     htmp = prod(trans(W), W);
     hd   = prod(htmp, H);             //denominator matrix for H
     launch_mul_div_kernel(H, hn, hd); //element-wise operations


     wn   = prod(W, trans(H));         //numerator matrix for W
     wtmp = prod(W, H);
     wd   = prod(wtmp, trans(H));      //denominator matrix for W
     launch_mul_div_kernel(W, wn, wd); //element-wise operations
}
```

У цьому псевдокоді по-елементні операції виділені у функцію launch_mul_div_kernel, а матричне множення у функцію prod. Завдяки інтерфейсу високого рівня, що надається ViennaCL результуючий C++ код реалізації NMF алгоритму є дуже схожим на приведений псевдокод.

У типовому випадку, коли параметр k є малим в порівнянні з m та n, найдорожчими операціями у плані кількості обчислень, є операції в яких приймає участь матриця V. Це є вірним, тому що тільки матриця V має розмір $m \times n$, всі інші матриці мають k, як одну з розмінностей. З іншого боку, часто, у практичних застосуваннях TD матриця V є розрідженою. Тому обчислювальна ефективність алгоритму може бути суттєво

покращена з урахуванням цієї структурної інформації. Також, дуже важливим є зменшення обсягу використаної пам'яті, що є вельми значимим, тому що розмір доступної пам'яті на сучасних GPU є дуже обмеженим. Таким чином, використовуючи спеціальний формат для збереження розріджених матриць, такий як формат "стиснутих розріджених рядків" (Compressed Sparse Row – CSR), розроблена програма здатна обробляти набагато більший набір документів, у порівнянні з випадком щільних матриць.

Добуток розрідженої матриці на щільну не є реалізованим у ViennaCL та більшості інших бібліотек, тому ми змушені створити власну реалізацію. Наша реалізація засновується на раніше розробленій процедурі множення розрідженої матриці на вектор. Оскільки таке узагальнення є досить таки прямолінійним, код OpenCL ядра не приводиться тут заради стислості викладу.

Важливо відзначити, що кроки алгоритму, вимагають множення матриці V як "з ліва", так і "з права". Однак, формат зберігання елементів матриці CSR, дозволяє ефективно реалізувати множення тільки "з ліва", множення "з права" не може бути ефективно реалізоване. Існує два можливих засоби обійти цю проблему. Перший це збереження стиснутих стовпців поруч із стиснутим рядками. Крім додаткової пам'яті, такий формат вимагає розробки додаткової процедури для множення на розріджену матрицю "справа". Другий варіант полягає у зберіганні транспонованої матриці V^T у форматі CSR, тоді відповідний крок алгоритму може бути переписаний наступним чином:

$$W^T V = ((W^T V)^T)^T = (V^T W)^T . \quad (5)$$

Це дає можливість використання процедури множення "зліва" з подальшим транспонуванням результату операції множення. Оскільки розробка процедури транспонування є значно простішою, у даній роботі використовується другий варіант.

Тестування

Для отримання результатів проведено ряд експериментів. Тестові дані включають реальні і штучно-створені матриці.

У експериментах використовувалось наступне апаратне забезпечення:

- Intel Core i7 960, 3,2 ГГц (GPU);
- NVIDIA GeForce 470 GTX (GPU);
- AMD 6970 Radeon HD (GPU).

Для порівняння запропонованої прискорення за допомогою GPU реалізації, була розроблена програма, що використовує тільки CPU для розрахунків. Ця програма використовує бібліотеку Eigen, яка розповсюджується у відкритих вихідних кодах та містить високо-оптимізовані алгоритми лінійної алгебри. Для наших тестів, ця версія була скомпільована з максимальним рівнем оптимізації та з підтримкою векторних інструкцій SIMD. Також була включена підтримка OpenMP, що дало змогу використовувати усю потужність багатоядерних процесорів.

Для першого експерименту створено набір випадково створених матриць, що мають розміри від 1000 на 1000 до 10000 на 10000. Всі матриці з цього набору мають приблизно 1% ненульових елементів, що дозволяє використовувати їх у тестах для обох видів матриць (щільних і розріджених). Кількості ознак k приймає одне з трьох значень: 3, 20 і 200.

Результати виконання програм на випадкових матрицях показано на рис. 1. Можна побачити, що використання типу розріджених матриць, замість типу щільних матриць, призводить до приросту продуктивності, як на GPU, так і на GPU. Перевага від використання розріджених матриць при k = 3 становить – 10 разів, при k = 200 прискорення – 2 рази. На графічних процесорах приріст становить від 3 до 10 раз.

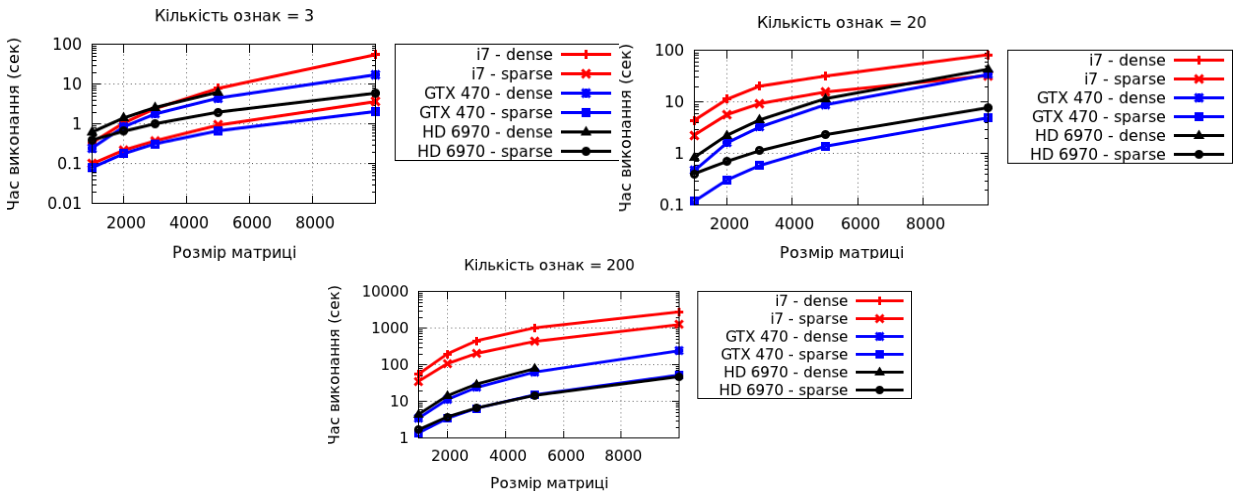

Рис. 1. Час виконання програм на штучно створених матрицях для значень параметра $k = 3, 20, 200$

Порівняння швидкості виконання між центральним і графічними процесорами показує різницю у часі до 25 разів для великих матриць та значення параметра $k = 200$. У свою чергу, для значення параметра $k = 3$, отриманий приріст швидкодії становить два рази, для $k = 20$ – приблизно 10 разів.

Як експеримент, що заснований на реальних даних використовується відомий тестовий набір документів Newsgroups. Тексти з цього набору даних є повідомленнями, що були зібрані з 20 різних груп новин. Загальне число документів у наборі – 18827. Завдяки характеру текстів, які зазвичай є досить короткими, TD-матриця V для цього набору даних є дуже розрідженою. Розмір результуючої матриці становить 18827 на 87014 елементів, причому кількість ненульових серед них – 1553867. Слід звернути увагу, що при використанні щільної матриці такого ж розміру, потрібно було б зберегти більш ніж один мільярд записів, що значно перевищує обсяг пам'яті, який є доступним у сучасних настільних комп'ютерах.

Результати виконання програм на наборі Newsgroups показано на рис. 2 . Приріст продуктивності, що був отриманий завдяки використанню графічних процесорів, становить від 2 до 20 разів і залежить від величини параметра k .

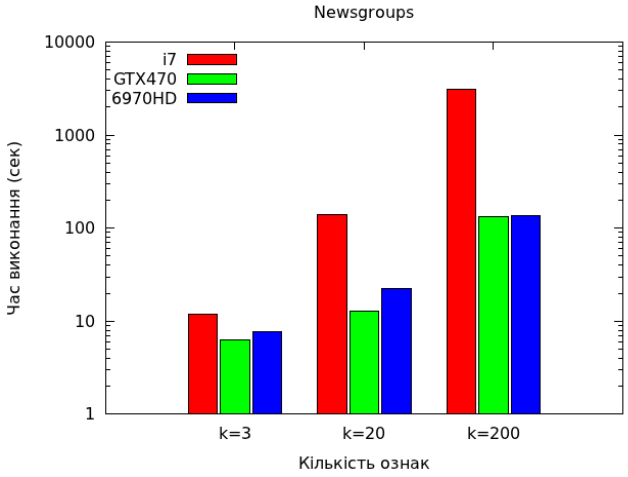

Рис. 2. Час виконання програм для тестового набору Newsgroups

Висновок

У даному дослідженні вивчається можливість прискорення алгоритму NMF за допомогою графічних процесорів. Основний акцент зроблено на оптимізації роботи алгоритму на даних, що є характерним для задач обробки природно мовних текстів, таких як, кластеризація документів. Запропонована імплементація розроблена за допомогою бібліотеки ViennaCL, що надає ряд матричних операцій високого рівня. Також, у роботі описана можливість застосування розріджених матриць у алгоритмі NMF, що дозволяє отримати значне прискорення алгоритму, навіть без застосування графічних процесорів.

У цілому, із застосуванням GPU, алгоритм було прискорено у 30 разів. Таким чином, такі результати доводять перспективність використання сучасних графічних адаптерів у обрахунках, що виникають у задачах обробки природно мовних текстів.

1. *Lee D.D. and Seung H.S.* Algorithms for Non-Negative Matrix Factorization // NIPS, 2000.
2. *Harish P. and Narayanan P.J.* Accelerating Large Graph Algorithms on the GPU using CUDA. Proc.14th Intl.Conf.on High Performance Computing (HiPC'07). – 2007.
3. *NVIDIA*, CUDA, URL: http://www.nvidia.com/
4. *Khronos Group*. OpenCL. URL: http://www.khronos.org/opencl/
5. *Agullo E. et al.* Numerical Linear Algebra on Emerging Architectures: The PLASMA and MAGMA projects // J.Phys.: Conf.Ser. – 2009. – Vol.180.
6. *Rupp K. et al.* ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. Proc. Intl. Workshop on GPUs and Scientific Applications (GPUScA 2010). – 2010. – P. 51– 56.
7. *Xu R. and Wunsch D.*, Survey of Clustering Algorithms. IEEE Trans.on Neural Networks. – 2005. – Vol.16. – P. 645–678.
8. *Deerwester S. et al.* Indexing by Latent Semantic Analysis // Journal of the American Society for Information Science. – 1990, Vol. 41. – P. 391–407.
9. *Xu W. et al.* Document Clustering Based on Non-Negative Matrix Factorization. Proc. 26th Intl. Conf. Research and Development in Information Retrieval. – 2003. – P. 267 – 273.
10. *Pauca V. et al.* Text Mining Using Non-Negative Matrix Factorizations. Proc.4th SIAM Intl.Conf.Data Mining. – 2004.