DOI 10.36074/logos-15.11.2024.034

ПРИСКОРЕННЯ МНОЖЕННЯ МАТРИЦЬ ЗА ДОПОМОГОЮ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Горішня Катерина Олексіївна[1], Кравець Наталя Сергіївна[2]

1. здобувач вищої освіти факультету комп'ютерних наук
Харківський національний університет радіоелектроніки, УКРАЇНА
ORCID ID: 0009-0001-9032-4249

2. канд. техн. наук, доцент, доцент кафедри програмної інженерії
Харківський національний університет радіоелектроніки, УКРАЇНА
ORCID ID: 0000-0002-6753-3333

Множення матриць є однією з найважливіших і найбільш обчислювально інтенсивних операцій, що застосовуються в багатьох галузях науки та техніки, включаючи машинне навчання, фізичне моделювання та обробку великих обсягів даних [5]. Виконання множення великих матриць потребує значних обчислювальних ресурсів і часу, тому оптимізація цієї операції є важливою задачею в сучасних обчисленнях. Зокрема, паралельне програмування дозволяє суттєво прискорити обчислення шляхом використання багатоядерних процесорів і графічних процесорів [4], що обумовлює актуальність розробки ефективних паралельних алгоритмів множення матриць.

Сучасні методи прискорення множення матриць зосереджені на розпаралелюванні обчислень для зменшення часу виконання. Серед поширених підходів — розбиття матриць на блоки та розподіл обчислень між потоками [3]. Використання паралельних потоків дозволяє одночасно обробляти незалежні частини матриць, тим самим знижуючи навантаження на процесор і забезпечуючи підвищення продуктивності. Основними технологіями, що застосовуються для паралельного множення матриць, є OpenMP, MPI, CUDA, Numpy [1] та інші.

У даній роботі проведено порівняння класичного та оптимізованого алгоритмів множення матриць. Класичний алгоритм перебирає всі елементи матриць по рядках та по стовпчиках, тоді як оптимізований застосовує перестановку індексів вкладених циклів, що дозволяє перебирати дані тільки по рядках, зберігаючи результат обчислень. Паралелізація алгоритмів

виконувалася з використанням бібліотеки OpenMP, застосовуючи директиву #pragma omp parallel for, яка дозволяє розпаралелювати цикли [2]. Для оцінки прискорення проводився експеримент з різною кількістю потоків та вимірювався час виконання.

Для експериментів використовувався комп'ютер із процесором Intel Core i5-2430M з тактовою частотою 2.40 ГГц (2 фізичних ядра, 4 потоки) та 12 ГБ оперативної пам'яті, мова програмування – C#.

Експерименти проводилися для обох алгоритмів з використанням 2, 3 і 4 потоків для матриць типу double розміром 1024x1024. Після кожного запуску фіксувалися такі показники, як час виконання, прискорення та ефективність. Прискорення обчислюється як співвідношення часу виконання послідовної версії програми до часу паралельної версії при використанні певної кількості потоків. Ефективність обчислюється як співвідношення прискорення до кількості потоків.

Першим етапом стало створення послідовної версії обох алгоритмів, яка дозволила отримати базовий час виконання без паралельного обчислення, що слугувало еталоном для порівняння з паралельними версіями. Далі, для кожного алгоритму окремо виконувалося паралельне множення з використанням OpenMP.

На основі проведених експериментів склали таблицю часу виконання та прискорення для різної кількості потоків для обох алгоритмів (табл. 1, табл. 2). Ефективність зменшувалася зі збільшенням кількості потоків, що пояснюється накладними витратами на синхронізацію між потоками.

Таблиця 1

Результати експериментів для класичного алоритму

Етапи експерименту	Час виконання	Прискорення	Ефективність
Послідовне виконання	72.457с.	1.00	100%
2 потоки	34.395с.	2.106	105.33%
3 потоки	27.383с.	2.645	88.199%
4 потоки	28.273с.	2.566	64.152%

авторська розробка

Таблиця 2

Результати експериментів для оптимізованого алоритму

Етапи експерименту	Час виконання	Прискорення	Ефективність
Послідовне виконання	65.753с.	1.00	100%
2 потоки	24.838с.	2.647	132.36%
3 потоки	19.807с.	3.319	110.655%
4 потоки	20.066с.	3.276	81.918%

авторська розробка

Згідно з результатами проведених експериментів, оптимізований алгоритм множення матриць демонструє вищу продуктивність порівняно з класичним алгоритмом, завдяки перестановці індексів у вкладених циклах, що зменшує комунікації із пам'яттю. Це видно з часу послідовного виконання, який для оптимізованого алгоритму становить менше.

Під час експериментів для обох алгоритмів використовувалася різна кількість потоків. Найбільш ефективне прискорення та продуктивність були досягнуті при використанні 2 та 3 потоків. Використання 3 потоків для оптимізованого алгоритму забезпечило найвищий показник прискорення (3.319) і ефективності (110.655%), що свідчить про ефективне використання апаратних ресурсів. Проте, при збільшенні кількості потоків до 4 спостерігалося зниження ефективності для обох алгоритмів: до 64.152% для класичного алгоритму та до 81.918% для оптимізованого. Це пов'язано з тим, що процесор має лише 2 фізичних ядра і 4 потоки, при використанні максимального числа потоків призводить до збільшення накладних витрат на синхронізацію та менш ефективного використання ресурсів.

Отже, оптимальною кількістю потоків для цього процесора є 2 або 3, оскільки це дозволяє досягти високого прискорення без значних втрат в ефективності. Використання 4 потоків не забезпечує пропорційного збільшення продуктивності через архітектурні обмеження процесора, що підкреслює важливість вибору кількості потоків для досягнення максимальної продуктивності.

Для подальшого покращення продуктивності слід розглянути можливість використання графічних процесорів (GPU), які здатні обробляти великі обсяги паралельних обчислень більш ефективно. Подальші дослідження можуть бути спрямовані на оптимізацію алгоритмів для покращення використання кешу та ресурсів пам'яті, що дозволить досягти ще більшої продуктивності при паралельному множенні матриць.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

[1] Chapman, B. (2008). Using OpenMP: Portable shared memory parallel programming. The MIT Press.
[2] Jungletronics. (2024, August27). Parallel computing using OpenMP. Medium. Retrieved from https://medium.com/jungletronics/parallel-computing-using-openmp-0b61c5c9d2e6
[3] Kozyrakis, C., Hennessy, J. L., & Patterson, D. A. (2024). Computer Architecture: A Quantitative Approach. Elsevier Science & Technology Books.
[4] Pacheco, P., & Malensek, M. (2021). Introduction to Parallel Programming. Elsevier Science & Technology Books.
[5] Watson, T. J., Pandey, A. B., & Kendig, K. L. (2013). Affordable Metal-Matrix Composites for High Performance Applications II. Wiley & Sons, Incorporated, John.