# Тестування роботи точного алгоритму для задачі про суму підмножини на різних персональних комп'ютерах

*Михайло Ленський*
ORCID: https://orcid.org/0009-0001-1445-2142
Дніпровський національний університет імені Олеся Гончара, Дніпро

*Ганна Михальчук*
ORCID: https://orcid.org/0000-0002-5476-6349
Дніпровський національний університет імені Олеся Гончара, Дніпро

Задача про суму підмножини є відомою NP-повною задачею, яка полягає у визначенні, чи існує підмножина з даного набору чисел, сума якої дорівнює заданому цільовому значенню. Ця задача має багато застосувань у криптографії, комбінаториці та оптимізації. Знаходження точного розв'язку для великих екземплярів задачі є обчислювально складним, оскільки кількість можливих підмножин зростає експоненціально зі збільшенням розміру вхідного набору.

У цій роботі розглядається точний алгоритм для розв'язання задачі про суму підмножини з прискоренням на GPU. Алгоритм базується на методі пошуку з поверненням (backtracking) [1], загальній техніці, яка досліджує простір можливих розв'язків шляхом рекурсивного розгалуження на кожному елементі вхідного набору.

Алгоритм складається з двох фаз: фази пошуку в ширину, що виконується на центральному процесорі (CPU), та фази пошуку в глибину, що виконується на графічному процесорі (GPU).

У фазі пошуку в ширину алгоритм починає роботу з порожньої підмножини та розширює її, додаючи по одному елементу за один крок у порядку вхідного набору. Проміжні підмножини та їх суми зберігаються в черзі. Також застосовуються правила обрізання гілок для відкидання непродуктивних підмножин. Фаза пошуку в ширину зупиняється, коли досягнуто заздалегідь визначене обмеження глибини, або коли черга порожня, або коли знайдено розв'язок.

На початку фази пошуку в глибину дані з черги передаються в пам'ять графічного процесору, після чого запускається певна кількість паралельних потоків. Кожен потік виконує пошук в глибину на різних піддеревах простору пошуку, починаючи з різних підмножин у черзі. Фаза пошуку в глибину слідує тій самій логіці, що й фаза пошуку в ширину, але використовує стек замість черги для зберігання проміжних підмножин. Фаза завершується, коли всі потоки завершують свою роботу, або коли знайдено розв'язок.

В розглянутому алгоритмі використовується обрізання гілок для

обмеження простору пошуку. Коли поточна часткова сума перевищує цільове значення, гілка вважається непродуктивною, і алгоритм повертається назад, щоб дослідити альтернативні гілки. Крім того, якщо сума поточної підмножини та максимально можливої суми, що залишилася, менша за цільове значення, гілка відкидається, щоб уникнути вичерпного дослідження непродуктивних шляхів. Ці правила обрізання гілок дозволяють виключити великі частини простору пошуку, які не можуть містити допустимого розв'язку.

Алгоритм було реалізовано мовою програмування C# з використанням бібліотеки ILGPU, яка забезпечує високорівневу абстракцію для програмування на GPU. Ця бібліотека дозволяє розробникам писати код для графічних процесорів, використовуючи звичну для них мову C#, що значно спрощує процес розробки паралельних обчислювальних програм.

Основною перевагою використання ILGPU є її здатність автоматично генерувати високоефективний код для GPU, оптимізований під конкретну архітектуру графічного процесора. Це дозволяє досягти максимальної продуктивності, не вдаючись до низькорівневого програмування на мовах, таких як CUDA або OpenCL, які вимагають глибоких знань архітектури GPU та специфічних оптимізацій.

Було проведено тестування швидкості роботи алгоритму на трьох різних персональних комп'ютерах, що обладнані наступними зв'язками CPU та GPU:

1. Intel Core i5-8250U, Nvidia GeForce MX150.
2. AMD Ryzen 7 5800H, Nvidia GeForce GTX 1650.
3. AMD Ryzen 5 7600, Nvidia GeForce RTX 4070 Ti SUPER.

Для проведення обчислювального експерименту було створено 5 наборів тестових даних. Кожен такий набір містить від 50 до 1000 цілих додатних чисел. Для кожного набору даних обрано цільове значення, яке дорівнює сумі певної кількості чисел з набору. Таким чином, гарантовано існує принаймні одна підмножина чисел з кожного набору, сума якої дорівнює цільовому значенню.

Щоб зменшити кількість можливих розв'язків, числа в наборах були згенеровані в діапазоні від $10^7$ до $10^8$. Оскільки час роботи алгоритму значно залежить від порядку чисел у наборі даних, алгоритм запускався по 25 разів на кожному наборі. Перед кожним запуском алгоритму числа в тестовому наборі даних перемішувалися. Порядок перемішаних чисел є однаковим на кожному з персональних комп'ютерів. Запуски алгоритму на GPU проводилися на різних кількостях потоків (від 512 до 65536). Результати тестування наведені в табл. 1.

**Таблиця 1 – Середній час роботи алгоритму на різних наборах даних**

| Тип | ПК | Середній час роботи за різної кількості елементів, мс | | | | |
|---|---|---|---|---|---|---|
| | | 50 | 100 | 200 | 500 | 1000 |
| CPU | 1 | 4603 | 8271 | 4115 | 4854 | 24358 |
| | 2 | 1485 | 3087 | 1484 | 1829 | 9995 |
| | 3 | 883 | 1912 | 860 | 1100 | 6458 |
| GPU (512 потоків) | 1 | 1023 | 2292 | 1524 | 1834 | 11879 |
| | 2 | 966 | 2326 | 1450 | 1615 | 10333 |
| | 3 | 588 | 1427 | 849 | 938 | 5988 |
| GPU (1024 потоків) | 1 | 1017 | 2144 | 1251 | 1399 | 10832 |
| | 2 | 1035 | 2278 | 1196 | 1264 | 9544 |
| | 3 | 300 | 588 | 345 | 487 | 3103 |
| GPU (2048 потоків) | 1 | 1440 | 4114 | 1796 | 2403 | 23025 |
| | 2 | 575 | 1494 | 648 | 653 | 7218 |
| | 3 | 198 | 508 | 213 | 236 | 2507 |
| GPU (4096 потоків) | 1 | 2078 | 4944 | 3526 | 3454 | 35688 |
| | 2 | 697 | 1690 | 1281 | 1289 | 13301 |
| | 3 | 97 | 303 | 132 | 156 | 1524 |
| GPU (8192 потоків) | 1 | 2550 | 4386 | 3656 | 4684 | 48355 |
| | 2 | 1305 | 1898 | 1450 | 1923 | 18507 |
| | 3 | 76 | 129 | 82 | 107 | 865 |
| GPU (16384 потоків) | 1 | 2516 | 5518 | 2726 | 5290 | 41121 |
| | 2 | 1499 | 2206 | 1508 | 2395 | 21424 |
| | 3 | 48 | 87 | 55 | 96 | 567 |
| GPU (32768 потоків) | 1 | 2878 | 5078 | 2905 | 5492 | 41584 |
| | 2 | 1202 | 1956 | 1450 | 2220 | 21318 |
| | 3 | 74 | 117 | 66 | 181 | 759 |
| GPU (65536 потоків) | 1 | 2764 | 5815 | 3274 | 5798 | 45441 |
| | 2 | 1372 | 2599 | 1188 | 2349 | 24731 |
| | 3 | 69 | 126 | 130 | 245 | 933 |

Аналіз результатів, наведених у таблиці 1, демонструє значні переваги паралельних обчислень на GPU порівняно з послідовним виконанням на CPU. Для великих вхідних наборів даних найкращі показники часу досягаються на ПК з графічним процесором RTX 4070 Ti SUPER у разі використання від 4096 до 65536 потоків.

Зокрема, для набору з 1000 елементів найшвидший час 567 мс було отримано на RTX 4070 Ti SUPER з 16384 потоками, що майже в 12 разів швидше, ніж виконання на CPU того самого комп'ютера (6458 мс), і в 43 рази швидше, ніж на найслабшому з протестованих ПК із CPU Intel Core i5-8250U (24358 мс).

Крім того, результати демонструють, що існує оптимальна кількість потоків для різних розмірів вхідних даних і різних GPU. Наприклад, MX150 більш ефективно працює на невеликій кількості потоків,

а RTX 4070 Ti SUPER – навпаки.

Результати тестування демонструють значні переваги використання GPU для прискорення розв'язання задачі про суму підмножини. Ключовими чинниками, що зумовлюють високу ефективність GPU є:

– Масивний паралелізм. Сучасні GPU містять тисячі обчислювальних ядер (наприклад, графічний процесор RTX 4070 Ti SUPER має 8448 ядер CUDA), що дозволяє одночасно виконувати велику кількість потоків і підвищує продуктивність для добре паралелізованих завдань. У розглянутому алгоритмі кожен потік виконує пошук у одному з піддерев простору розв’язків.

– Спеціалізована архітектура. GPU оптимізовані для виконання операцій над масивами даних та матричних обчислень з використанням апаратно реалізованих примітивів і вбудованих функцій. Хоча в запропонованому алгоритмі не використовуються специфічні для GPU операції, його паралельна природа добре підходить для архітектури GPU.

– Висока пропускна здатність пам'яті. GPU обладнані окремою високошвидкісною пам'яттю (наприклад, RTX 4070 Ti SUPER має 16 ГБ пам'яті GDDR6X з пропускною здатністю 672,3 ГБ/с), що дозволяє ефективно забезпечувати обчислювальні потоки необхідними даними.

Ці властивості роблять GPU ідеальними для прискорення задач, де можна паралелізувати обчислення і розподілити їх між численними потоками. Задача про суму підмножини є яскравим прикладом такої задачі через можливість незалежного рекурсивного пошуку в різних гілках дерева розв’язків.

Підбиваючи підсумки, розглянутий алгоритм є ефективним методом розв'язання NP-повної задачі про суму підмножини. Завдяки паралельним обчисленням на GPU та застосуванню правил обрізання гілок, алгоритм може знаходити точні розв'язки для великих екземплярів задачі за прийнятний час. Крім того, алгоритм можна розширити для розв'язання інших споріднених NP-повних задач. Використання потужного GPU може значно прискорити виконання алгоритму.

## Посилання

1. Skiena S. S. The Algorithm Design Manual. 2nd ed. London : Springer, 2008. 730 p.