Д.О. Романенко, магістр, гр. ПІ-46м, ФІКТ
Науковий керівник – к.т.н., доц. А.М. Ковальчук
Житомирський державний технологічний університет

РОЗВ'ЯЗОК ЗАДАЧІ КОМІВОЯЖЕРА МЕТОДОМ ЛІТТЛА З ВИКОРИСТАННЯМ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ NVIDIA CUDA

Nvidia CUDA – програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність завдяки використанню графічних процесорів фірми Nvidia. CUDA SDK дозволяє програмістам реалізовувати на спеціальному спрощеному діалекті мови програмування С алгоритми, які виконуються на графічних процесорах Nvidia. Архітектура CUDA дає розробнику можливість використовувати набір інструкцій графічного обчислюваного пристрою і управляти його пам'яттю.

Задача комівояжера – задача пошуку мінімальної вартості маршрута без повторень на повному зваженому графі з n вершинами. Вершини графу є місцями, які повинен відвідати комівояжер, а вага ребер відповідає довжині між цими місцями. Ця задача є NP-повною - точний переборний алгоритм має факторіальну складність.

Прості методи розв'язання задачі комівояжера: повний лексичний перебір, жадібні алгоритми (метод найближчого сусіда), метод включення найближчого міста, метод найдешевшого включення, метод мінімального кістяка дерева. На практиці застосовують різні модифікації ефективніших методів: метод гілок і меж (алгоритм Літтла), метод генетичних алгоритмів, а також алгоритм мурашиної колонії.

Алгоритм Літтла є окремим випадком використання методу "гілок і меж" для конкретного завдання. Загальна ідея тривіальна: потрібно розділити величезне число перебраних варіантів на класи і отримати оцінки (знизу - в завданні мінімізації, зверху - в завданні максимізації) для цих класів, щоб мати можливість відкидати варіанти не поодинці, а цілими класами. Складність полягає в тому, щоб знайти таке розділення на класи (гілки) та такі оцінки (границі), щоб процедура була ефективною.

Алгоритм Літтла для вирішення задачі комівояжера можна сформулювати так:

1. У кожному рядку матриці вартості знайдемо мінімальний елемент і віднімемо його з усіх елементів рядка. Зробимо це і для стовпців, що не містять нуля. Отримаємо матрицю вартості, кожен рядок і кожен стовпець якої містять хоча б один нульовий елемент.
2. Для кожного нульового елемента матриці c_{ij} розрахуємо коефіцієнт $\Theta_{i,j}$, що дорівнює сумі найменшого елемента *i* строки (виключаючи елемент $c_{i,j} = 0$) і найменшого елемента *j* стовпця. З усіх коефіцієнтів $\Theta_{i,j}$ виберемо такий, який є максимальним $\Theta_{k,l} = \max\{\Theta_{i,j}\}$. У гамільтонів контур вноситься відповідна дуга (*k*,*l*).
3. Вилучаємо *k*-ту строку та стовпець *l*, поміняємо на нескінченність значення елемента $c_{l,k}$ (оскільки дуга (*k*,*l*) включена в контур, то зворотний шлях з *l* в *k* неприпустимий).
4. Повторюємо алгоритм з кроку 1, поки порядок матриці не дорівнюватиме двом.
5. Потім у поточний орієнтований граф вносимо дві відсутні дуги, що визначаються однозначно матрицею порядку 2. Отримуємо Гамільтонів контур.

Для ефективної реалізації паралельних обчислень, було прийнято рішення залишити розгалудження множини, та роботу з рекурсією на CPU (адже рекурсія на GPU не підтримується, або підтримується частково); на обробку GPU перенести, лише елементи пов'язані з обчисленням матриць, а саме:

- знаходження мінімальних елементів та віднімання в рядках/стовпчиках;
- розрахунок коефіцієнта Θ;
- знаходження пари, що ініціює розгалудження;
- копіювання елементів масива до нового, з виключенням *k*-того рядка та стовпця *l*.

Порівняємо швидкість роботи алгоритму, використовуючи однопоточну реалізацію на мові програмування JavaScript, та реалізацію на мові програмування C++ з використанням паралельних обчислень Nvidia CUDA.

Для цього ми на вхід програми подамо матриці наступного розміру: 5x5, 10x10, 25x25, 50x50, 85x85, 115x115, 155x155, 200x200.

Характеристики обладнання, на якому проводилося тестування:

- Процесор: Intel Core i5 – 2450M (тактова частота – 2.5GHz);
- Відеокарта: Nvidia GeForce GT 630M (ядер CUDA – 96, тактова частота – 800MHz);
- ОЗУ: 8 Гб.

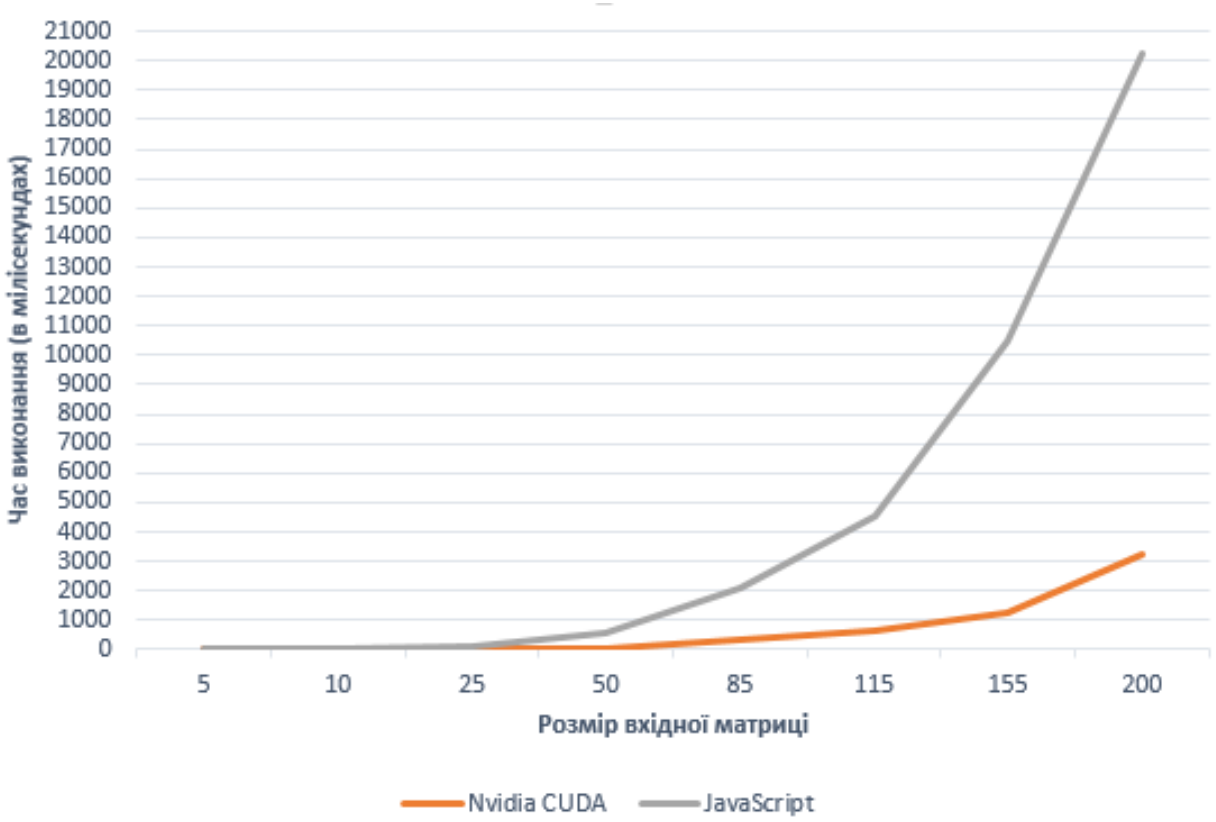

Рис. 1. Час виконання методу Літтла на Nvidia CUDA та JavaScript, залежно від розміру вхідної матриці

Як показують обрахунки (рис. 1.), можна зробити висновок, що алгоритм Літтла ефективніше працює з використанням паралельних обчислень Nvidia CUDA. Ми отримуємо 10-кратний приріст в швидкості обчислень, при збільшенні розмірів вхідної матриці. Даний ефект досягається можливістю одночасно використовувати велику кількість ядер, що розміщенні на GPU(в нашому випадку, до 96 ядер).