

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра програмування

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №8

Тема: «Використання архітектури OpenCL для паралельного множення матриць»

Виконав:

Ст. Лук'янчук Денис

Група ПМІ-33

Тема: «Використання архітектури OpenCL для паралельного множення матриць»

Мета роботи: Ознайомитися з принципами програмно-апаратної архітектури паралельних обчислень OpenCL, навчитися реалізовувати паралельні обчислення на CPU або GPU, оцінювати прискорення порівняно з традиційною реалізацією на CPU.

Теоретичний матеріал

OpenCL — це відкритий стандарт для гетерогенних обчислень, який дозволяє виконувати один і той самий код на різних пристроях: CPU, GPU, iGPU, FPGA тощо.

Головна ідея — розподіл обчислень між тисячами незалежних потоків, які одночасно виконують однакові інструкції над різними даними.

У лабораторній реалізовано множення матриць $C = A \times B$, де:

$$C_{ij} = \sum_{k=1}^N A_{ik} \cdot B_{kj}$$

Програма містить два варіанти реалізації:

1. Послідовне множення на CPU — три вкладені цикли без паралелізму.
2. Паралельне множення з використанням OpenCL, де кожен work-item обчислює один елемент матриці результату.

Для OpenCL створюється контекст, командна черга, і передаються буфери даних A, B, C. Далі запускається ядро, яке виконується тисячами потоків на GPU. Кожен потік обчислює один елемент результату незалежно від інших.

Хід роботи

Спочатку я створив дві квадратні матриці — A і B, які заповнюються випадковими числами від 1 до 10. Розмір матриці (N×N) я задавав прямо в коді.

Далі я реалізував два варіанти множення:

1. Звичайний (послідовний) — через три вкладені цикли на процесорі. Це базовий варіант, щоб знати, скільки часу займає обчислення без паралелізації.
2. Паралельний (через OpenCL) — де кожен елемент результату обчислюється окремо, незалежно від інших.

Для OpenCL я спочатку створюю контекст і чергу команд — це середовище, де запускаються обчислення. Потім я передаю туди матриці A і B через спеціальні буфери.

OpenCL запускає ядро — невелику функцію, яка виконується тисячами разів одночасно. Кожен "робочий елемент" (work-item) рахує лише один елемент матриці результату $C[row][col]$. Завдяки цьому вся матриця множитья паралельно.

Після завершення роботи ядра я зчитую готову матрицю C назад у звичайну пам'ять і порівнюю її з результатом CPU-варіанта. Якщо значення відрізняються не більше ніж на 0.01 — вважаю, що все працює правильно.

Потім я вимірюю час виконання обох варіантів — спочатку послідовного, потім OpenCL. Порівнюю їх між собою, щоб дізнатись, наскільки швидше (або повільніше) працює OpenCL.

Експерименти проводив для кількох розмірів матриць — 100, 500, 1000, 2000 — щоб побачити, як змінюється швидкість при збільшенні задачі. На малих розмірах CPU виявився швидшим, а на великих OpenCL уже дав помітне прискорення.

Для початку я виконав множення матриць розміром 100×100 :

```
OpenCL Matrix Multiplication (N=100)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 0.732133 ms
GPU time: 2.06661 ms
Compute Units: 64
Speedup: 0.354267x
Validation: OK
```

При малих розмірах матриць швидкість виконання через CPU швидша за виконання через OpenCL. Відповідно, прискорення дорівнює менше одиниці.

Далі я продовжив збільшувати розмірність матриці до 500 на 500:

```
OpenCL Matrix Multiplication (N=500)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 53.5966 ms
GPU time: 6.9882 ms
Compute Units: 64
Speedup: 7.66959x
Validation: OK
```

При матрицях розмірності 500 на 500 бачимо те, що виконання через GPU стало швидшим за час CPU і прискорення сягнуло більше одиниці.

Матриці 1000 на 1000:

```
OpenCL Matrix Multiplication (N=1000)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 450.373 ms
GPU time: 38.9808 ms
Compute Units: 64
Speedup: 11.5537x
Validation: OK
```

При матрицях розмірності 1000 на 1000 бачимо аналогічну тенденцію, що виконання через OpenCL є швидшим за час CPU разом зі прискоренням.

Порівняння з попередньою лабораторною роботою

Ліворуч — результати попередньої лабораторної. Праворуч — нинішньої роботи:

Матриця A: 100x100
Матриця B: 100x100
Кількість потоків: 16

Послідовне множення завершено за 5 мс (0,01 с)
Паралельне множення завершено за 6 мс (0,01 с)

Прискорення: 0,83
Ефективність: 5,21%

```
OpenCL Matrix Multiplication (N=100)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 0.732133 ms
GPU time: 2.06661 ms
Compute Units: 64
Speedup: 0.354267x
Validation: OK
```

Матриця A: 500x500
Матриця B: 500x500
Кількість потоків: 16

Послідовне множення завершено за 477 мс (0,48 с)
Паралельне множення завершено за 127 мс (0,13 с)

Прискорення: 3,76
Ефективність: 23,47%

```
OpenCL Matrix Multiplication (N=500)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 53.5966 ms
GPU time: 6.9882 ms
Compute Units: 64
Speedup: 7.66959x
Validation: OK
```

Матриця A: 1000x1000
Матриця B: 1000x1000
Кількість потоків: 16

Послідовне множення завершено за 3772 мс (3,77 с)
Паралельне множення завершено за 704 мс (0,70 с)

Прискорення: 5,36
Ефективність: 33,49%

```
OpenCL Matrix Multiplication (N=1000)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 450.373 ms
GPU time: 38.9808 ms
Compute Units: 64
Speedup: 11.5537x
Validation: OK
```

```
Матриця A: 2000x2000
Матриця B: 2000x2000
Кількість потоків: 16
```

```
Послідовне множення завершено за 34013 мс (34,01 с)
Паралельне множення завершено за 6293 мс (6,29 с)
```

```
Прискорення: 5,40
Ефективність: 33,78%
```

```
OpenCL Matrix Multiplication (N=2000)
Device: Intel(R) Graphics [0xa7a8]
CPU time: 13039.8 ms
GPU time: 232.112 ms
Compute Units: 64
Speedup: 56.1788x
Validation: OK
```

Підсумок:

- Для малих матриць (до 100×100) OpenCL не показує приросту швидкодії — послідовне виконання на CPU навіть трохи швидше через накладні витрати на створення контексту та копіювання даних.
- Починаючи з розмірів матриці 500×500 , паралельне множення через OpenCL починає перевищувати CPU-варіант, показуючи прискорення понад $7\times$.
- При середніх/великих матрицях (1000×1000 і більше) OpenCL стає істотно швидшим — прискорення сягає від $11\times$.
- У порівнянні з попередньою лабораторною, де використовувались звичайні потоки, OpenCL демонструє меншу ефективність на дуже малих задачах, але виграє на великих.
- Таким чином, OpenCL доцільно використовувати для обчислень з великими масивами даних, тоді як для невеликих матриць ефективніше залишатись на потоках CPU.

Варто зазначити, що попередня лабораторна робота була реалізована мовою C#, тоді як поточна — на C++ із використанням OpenCL, що пояснює відмінності у підходах до паралельних обчислень і рівні продуктивності. Саме через це поточна реалізація демонструє вищу швидкодію порівняно з попередньою, оскільки OpenCL забезпечує глибше апаратне розпаралелення та ефективніше використовує ресурси процесора.

Висновок: У лабораторній роботі я навчився реалізовувати множення матриць із використанням технології OpenCL та проведено порівняння результатів із попередньою лабораторною, де застосовувалось паралельне множення через потоки.

Отримані результати показали, що при малих розмірах матриць OpenCL поступається CPU через накладні витрати на створення контексту, компіляцію ядра та передачу даних. Проте зі збільшенням розмірності матриць ефективність OpenCL зростає, і при великих об'ємах обчислень ($N \geq 500$) він починає перевершувати CPU-варіант за швидкодією.

Таким чином, технологія OpenCL є доцільною для розв'язання задач із великими обсягами даних, де можливо задіяти апаратний паралелізм обчислювальних блоків. Для невеликих задач ефективніше застосовувати традиційне розпаралелення потоків на процесорі.