МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Лабораторна робота №7

“OpenCL”

Виконав:
студент групи ПМі-31
Дудинець Олександр

Львів 2024

Мета роботи

Метою цієї лабораторної роботи було розробити програму для обчислення добутку двох матриць з використанням OpenCL на C++, а також порівняння з послідовним та паралельним алгоритмами. Завдання включало реалізацію OpenCL-версії алгоритму множення матриць, оцінку часу виконання кожного з алгоритмів та аналіз ефективності OpenCL у порівнянні з іншими методами.

Опис ключових частин коду

1. **Генерація матриць (generateMatrix):**
    a. Функція створює двовимірну динамічну матрицю розміром n x m та заповнює її випадковими числами в діапазоні від 0 до 99.
    b. Використовується для створення вхідних даних для алгоритмів множення матриць.
2. **OpenCL обчислення матриць (multiplyMatricesOpenCL):**
    a. Реалізує множення матриць з використанням OpenCL.
    b. Основні кроки:
        i. Ініціалізація OpenCL середовища: Вибір платформи та пристрою (GPU або CPU), створення контексту та черги команд.
        ii. Створення та компіляція OpenCL ядра: Ядро (kernel) визначено у вигляді рядка та виконує множення матриць на пристрої.
        iii. Підготовка даних: Використання функції flattenMatrix для перетворення матриць у одномірні масиви.
        iv. Створення буферів: Створення OpenCL буферів для матриць A, B та C.
        v. Встановлення аргументів ядра: Передача буферів та параметрів розміру матриць в ядро.
        vi. Запуск ядра: Виконання ядра на пристрої з заданим глобальним розміром робочих груп.
        vii. Зчитування результатів: Копіювання результатів з буфера C та перетворення їх у двовимірну матрицю за допомогою unflattenMatrix.

c. Перевага використання OpenCL полягає в можливості виконувати обчислення на GPU, що забезпечує високу продуктивність для великих обсягів даних.

3. **Функції flatten та unflatten:**
   a. flattenMatrix: Перетворює двовимірну матрицю в одномірний масив. Це необхідно для передачі даних в OpenCL буфери, які працюють з лінійними типами даних.
   b. unflattenMatrix: Перетворює одномірний масив назад у двовимірну матрицю після отримання результатів з OpenCL буфера.
   c. Ці функції необхідні, оскільки OpenCL працює з одновимірними масивами, а наші дані зберігаються у вигляді двовимірних матриць.
4. **Порівняння матриць (areMatricesEqual):**
   a. Функція перевіряє, чи є дві матриці ідентичними за значеннями.
   b. Використовується для перевірки коректності результатів OpenCL та паралельних обчислень шляхом порівняння з результатами послідовних обчислень.
5. **Обчислення прискорення та ефективності:**
   a. calculateSpeedup: Обчислює прискорення як відношення часу виконання послідовного алгоритму до часу виконання паралельного або OpenCL алгоритму.
   b. calculateEfficiency: Обчислює ефективність як відношення прискорення до кількості потоків (для паралельного алгоритму).
6. **Аналіз (benchmark):**
   a. Функція benchmark запускає послідовний, паралельний та OpenCL алгоритми, вимірює час їх виконання та аналізує результати.
   b. Виводить на екран час виконання кожного алгоритму, прискорення та ефективність, а також перевіряє еквівалентність результатів.
7. **Параметри командного рядка:**
   a. Програма підтримує передачу аргументів для визначення розмірності матриць та кількості потоків, що дозволяє легко

змінювати параметри для тестування (Usage: <n> <m> <l> <threads> [runSequential] [runParallel] [runOpenCL]).

b. Наприклад: ./app 2000 3000 2000 8 0 1 1.

Аналіз результатів

Програма була протестована на різних розмірах матриць. Тестування проводилося на комп'ютері з процесором Apple M1 Pro, який має 10 ядер центрального процесора та 16 ядер графічного.

Матриці розміром 1000x2000 та 2000x1000:

```
- Summary:
  - Matrix size: 1000x2000 to 2000x1000
  - Threads: 8
  - Sequential time: 5253ms
  - Parallel time: 1031ms
  - OpenCL time: 306ms
  - Speedup (Parallel): 5.09505x
  - Efficiency (Parallel): 63%
  - Matrices equal (Sequential vs Parallel): Yes
  - Speedup (OpenCL): 17.1667x
  - Matrices equal (Sequential vs OpenCL): Yes
```

- OpenCL алгоритм показує значно більше прискорення порівняно з паралельним алгоритмом на CPU.
- Висока продуктивність OpenCL обумовлена використанням GPU для обчислень, що має велику кількість обчислювальних ядер та широку пам'ять.

Матриці розміром 100x200 та 200x100:

```
- Summary:
  - Matrix size: 100x200 to 200x100
  - Threads: 8
  - Sequential time: 10ms
  - Parallel time: 1ms
  - OpenCL time: 51ms
  - Speedup (Parallel): 10x
  - Efficiency (Parallel): 125%
  - Matrices equal (Sequential vs Parallel): Yes
  - Speedup (OpenCL): 0.196078x
  - Matrices equal (Sequential vs OpenCL): Yes
```

- На малих розмірах матриць OpenCL алгоритм працює повільніше, ніж послідовний та паралельний алгоритми.
- Причиною є накладні витрати на ініціалізацію OpenCL контексту, передачу даних та запуск ядра, які стають значними при малих обсягах обчислень.
- Паралельний алгоритм на CPU показує найкращі результати завдяки низьким накладним витратам на створення потоків для невеликого обсягу даних.

Матриці розміром 3000x3000:

```
- Summary:
  - Matrix size: 3000x3000 to 3000x3000
  - Threads: 8
  - Sequential time: 166706ms
  - Parallel time: 32080ms
  - OpenCL time: 4783ms
  - Speedup (Parallel): 5.19657x
  - Efficiency (Parallel): 64%
  - Matrices equal (Sequential vs Parallel): Yes
  - Speedup (OpenCL): 34.8539x
  - Matrices equal (Sequential vs OpenCL): Yes
```

- При збільшенні розміру матриць OpenCL алгоритм демонструє ще більше прискорення.
- GPU може ефективно розпаралелювати великі обсяги даних, використовуючи свою архітектуру для масивних паралельних обчислень.

Матриці розміром 5000x5000 та 10000x10000:

```
- Summary:
  - Matrix size: 5000x5000 to 5000x5000
  - Threads: 8
  - OpenCL time: 37578ms
```

```
- Summary:
  - Matrix size: 10000x10000 to 10000x10000
  - Threads: 8
  - OpenCL time: 301214ms
```

- Послідовний та паралельний алгоритми на CPU були пропущені через надто довгий час виконання.
- OpenCL алгоритм залишається практичним навіть для дуже великих матриць, тоді як CPU-алгоритми стають непридатними через величезний час виконання.
- Це підтверджує перевагу використання OpenCL та GPU для задач, що вимагають великих обсягів обчислень.

Загальні висновки з результатів

- **OpenCL ефективний на великих матрицях:** Накладні витрати на ініціалізацію OpenCL та передачу даних стають незначними у порівнянні з часом виконання обчислень на великих матрицях. Це призводить до значного прискорення.
- **Неефективність OpenCL на малих матрицях:** Для невеликих матриць накладні витрати OpenCL перевищують вигоду від паралельних обчислень на GPU, що робить послідовний або паралельний алгоритм на CPU більш ефективним.
- **Порівняння з паралельним алгоритмом на CPU:** Хоча паралельний алгоритм на CPU показує покращення у порівнянні з послідовним, OpenCL алгоритм в рази перевершує його на великих розмірах матриць.

Додаткове дослідження Metal та CUDA

Окрім реалізації алгоритму множення матриць з використанням OpenCL, було також проведено дослідження з використанням технології Metal для ARM процесорів Apple. OpenCL є кросплатформним стандартом для паралельних обчислень, однак на платформах Apple (особливо з переходом на ARM архітектуру процесорів) він поступово втрачає актуальність. Компанія Apple рекомендує розробникам використовувати Metal, який забезпечує глибшу інтеграцію з графічними можливостями та кращу оптимізацію під архітектуру їхніх пристроїв.

Мною було проведено експеримент з реалізацією цього ж алгоритму множення матриць за допомогою Metal. У результатах було зафіксовано навіть ще більш швидке виконання, ніж у OpenCL, особливо на великих матрицях. На прикладі тестів, які були виконані для матриць розміром 5000x5000 та 10000x10000, час виконання з використанням Metal був на 10-15% швидше в порівнянні з OpenCL.

Для інших архітектур, таких як NVIDIA, альтернативою є технологія CUDA. CUDA, оптимізована саме під GPU NVIDIA, тому забезпечує ще вищий рівень продуктивності на цих графічних процесорах. На жаль, у мене немає доступу до GPU NVIDIA, тому порівняти CUDA з OpenCL та Metal я не зміг.

Висновок

У цій лабораторній роботі було реалізовано алгоритм множення матриць з використанням OpenCL, а також його послідовний та паралельний аналоги на C++. OpenCL алгоритм продемонстрував значне прискорення на великих розмірах матриць завдяки ефективному використанню обчислювальних ресурсів GPU. Було виявлено, що OpenCL стає ефективнішим зі збільшенням розмірів матриць, тоді як для малих матриць краще використовувати традиційні методи.

Під час роботи було поглиблено розуміння принципів паралельних обчислень на GPU та особливостей роботи з OpenCL. Також було проведено невелике порівняння OpenCL з іншим фреймворком для паралельних обчислень – Metal.