УДК 004.925:004.356

О.К. Сирота

Л.В. Крупельницький

МЕТОДИ СТВОРЕННЯ Й ОПРАЦЮВАННЯ 3D-МОДЕЛЕЙ З ВИКОРИСТАННЯМ CUDA-ТЕХНОЛОГІЙ

Вінницький національний технічний університет

Анотація

У роботі проведено аналіз методів прискорення обчислювально-інтенсивних задач створення та опрацювання 3D-моделей. Досліджено характеристики пропрієтарної платформи NVIDIA CUDA та проведено її порівняльний аналіз з відкритим стандартом OpenCL з точки зору продуктивності, особливостей програмної моделі та ефективності використання апаратної архітектури GPU. Запропоновано методику порівняльного аналізу шляхом розробки програмних модулів, що реалізують ідентичні паралельні алгоритми обробки геометр та фізичних симуляцій. Розроблено рекомендації щодо оптимізації доступу до даних з використанням ієрархії пам'яті GPU для 3D-структур. Результати дослідження показують, що CUDA демонструє вищу продуктивність та має більш зрілу екосистему для роботи на GPU NVIDIA, тоді як OpenCL забезпечує апаратну незалежність та портативність програмного коду.

Ключові слова: 3D-моделювання, опрацювання 3D-моделей, CUDA, GPGPU, OpenCL, паралельні обчислення, архітектура GPU, Marching Cubes.

Abstract

The paper provides an analysis of methods for accelerating computationally intensive tasks of creating and processing 3D models. The characteristics of the proprietary NVIDIA CUDA platform are investigated, and its comparative analysis with the OpenCL open standard is conducted in terms of performance, programming model features, and efficiency of using the GPU hardware architecture. A comparative analysis methodology is proposed by developing software modules that implement identical parallel algorithms for geometry processing and physics simulations. Recommendations for optimizing data access using the GPU memory hierarchy (shared/local memory) for 3D structures are developed. The research results show that CUDA demonstrates higher performance and has a more mature ecosystem for NVIDIA GPUs, whereas OpenCL provides hardware independence and software code portability.

Keywords: 3D modeling, 3D model processing, CUDA, GPGPU, OpenCL, parallel computing, GPU architecture, Marching Cubes.

Вступ

Стрімке зростання складності 3D-моделей [1] у промисловому проєктуванні, віртуальній реальності та наукових симуляціях ставить нові виклики перед обчислювальними системами. Обробка геометричних сіток, що складаються з мільйонів полігонів, а також виконання фізичних симуляцій у реальному часі, створюють критичне навантаження на центральний процесор (CPU). Це робить послідовні алгоритми неефективними та унеможливлює інтерактивну роботу з великими масивами 3D-даних.

Актуальність теми визначається необхідністю перенесення цих обчислень на архітектуру графічних процесорів (GPU), які завдяки масовому паралелізму здатні одночасно виконувати тисячі потоків. Провідними технологіями GPGPU є пропрієтарна платформа NVIDIA CUDA та відкритий стандарт OpenCL. Проте, розробники часто стикаються з проблемою вибору між максимальною продуктивністю (CUDA) та кросплатформовою портативністю (OpenCL). Таким чином, проведення порівняльного аналізу цих технологій на реальних задачах 3D-обробки є важливою науково-прикладною задачею.

Результати дослідження

Відповідно до завдань роботи, було розроблено та реалізовано програмний стенд для порівняльного аналізу продуктивності CUDA та OpenCL. В якості обчислювально-інтенсивної задачі було обрано алгоритм реконструкції 3D-поверхні з воксельних даних "Marching Cubes" [2]. Цей алгоритм є "вузьким місцем" у багатьох 3D-конвеєрах і поєднує масовий паралелізм (незалежна обробка мільйонів вокселів) із нетривіальною логікою доступу до пам'яті (таблиці пошуку, атомарний запис результатів) . Було розроблено два програмні модулі з ідентичною алгоритмічною логікою [3], але з використанням різних GPGPU-технологій.

Обидві реалізації базувалися на спільних принципах проєктування паралельних структур даних. Вхідна воксельна сітка розміщувалася у глобальній пам'яті GPU у вигляді лінеаризованого 1D-масиву . Статичні таблиці пошуку edgeTable та triTable були розміщені у константній пам'яті constant у CUDA, const global в OpenCL для забезпечення кешованого, широкомовного доступу для всіх потоків. Ключова проблема паралельного запису вершин (оскільки кожен воксель генерує змінну кількість трикутників) була вирішена за допомогою атомарних лічильників, які дозволяли кожному потоку безпечно "резервувати" унікальний індекс у вихідному масиві вершин.

Реалізація на CUDA [4] була змодельована згідно з ієрархією Grid-Block-Thread. Кожен воксель оброблявся окремим потоком. Потоки групувалися у 3D-блоки (наприклад, 8 x 8 x 8), які, в свою чергу, формували 3D-сітку, що відповідала загальному розміру воксельного поля 8. Ключова оптимізація полягала у використанні швидкої пам'яті. Замість того, щоб кожен потік виконував 8 окремих зчитувань з повільної глобальної пам'яті, весь блок потоків спочатку кооперативно завантажував "патч" воксельних даних (наприклад, 9 x 9 x 9) у спільну пам'ять. Після бар'єрної синхронізації, всі 512 потоків блоку виконували обчислення "Marching Cubes", читаючи дані вже з надшвидкої пам'яті.

Реалізація на OpenCL [5] була спроєктована дзеркально. Ієрархія обчислень була представлена як NDRange (глобальний розмір) з 3D-робочими групами (також 8 x 8 x 8). Кожен робочий елемент так само відповідав за один воксель. Аналогічна оптимізація доступу до пам'яті була реалізована з використанням локальної пам'яті (аналог shared в OpenCL). Робоча група кооперативно завантажувала дані в масив локальної пам’яті, синхронізувалася за допомогою бар'єру і лише потім виконувала обчислення.

Експериментальні дослідження проводились на GPU NVIDIA GeForce RTX 3060, де вимірювався На першому етапі тестувалася задача обчислення усереднених нормалей для полігональної сітки, що складалася з 2.5 мільйонів вершин. Реалізація на CUDA продемонструвала середній час виконання 14.2 мс. Аналогічна реалізація на OpenCL, запущена на тому ж апаратному забезпеченні, потребувала 18.1 мс. Різниця у продуктивності становить близько 21%, що можна пояснити більш ефективною роботою драйвера CUDA з кешуванням та оптимізованими патернами доступу до глобальної пам'яті.

На другому, більш складному етапі, тестувався алгоритм "Marching Cubes" для реконструкції воксельних сіток різного розміру. Для сітки 256^3 вокселів, модуль CUDA виконав завдання за 20.3 мс, тоді як модуль OpenCL — за 26.4 мс (різниця ~23%). При збільшенні навантаження до 512^3 вокселів, час виконання склав 158.2 мс для CUDA та 219.5 мс для OpenCL. Тут різниця у продуктивності зросла до ~28%. Це свідчить про те, що перевага CUDA стає більш вираженою при зростанні обчислювальної складності, особливо в задачах, що активно використовують атомарні операції (atomicAdd) для паралельного запису у вихідний буфер вершин.

Додатково було досліджено апаратну масштабованість CUDA-реалізації при переході з NVIDIA RTX 3060 (28 SM) на RTX 4070 (46 SM). Приріст продуктивності для задачі "Marching Cubes" склав у середньому 1.55x (або 55%), що є близьким до теоретичного приросту кількості обчислювальних блоків (1.64x). Це підтверджує, що розроблена паралельна модель ефективно масштабується і не впирається у "вузькі місця", окрім апаратної потужності GPU.

Висновки

Проведене дослідження підтвердило високу ефективність застосування GPGPU-технологій для прискорення задач 3D-опрацювання. Аналіз реалізації алгоритму "Marching Cubes" дозволив встановити, що платформа NVIDIA CUDA демонструє об'єктивно вищу продуктивність (із приростом 20-25%) на графічних процесорах NVIDIA порівняно з OpenCL. Це пов'язано з тісною апаратно-програмною інтеграцією, зрілою екосистемою та високооптимізованими компіляторами. Водночас, OpenCL, хоча й поступається у чистій швидкодії на даному обладнанні, залишається єдиним життєздатним рішенням для розробки кросплатформових програмних продуктів, здатних працювати на GPU від AMD, Intel та інших виробників. Практичне значення роботи полягає в отриманні конкретних коефіцієнтів прискорення, що дає розробникам технічну основу для обґрунтованого вибору технології: CUDA — для проєктів, що вимагають максимальної продуктивності в екосистемі NVIDIA, та OpenCL — для проєктів, де пріоритетом є універсальність та апаратна незалежність.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What is 3D Modeling and Why it is used? [Електронний ресурс] — Режим доступу до ресурсу: https://www.geeksforgeeks.org/blogs/what-is-3d-modeling
2. Computer Graphics - 3D Computer Graphics [Електронний ресурс] — Режим доступу до ресурсу: https://www.tutorialspoint.com/computer_graphics/3d_computer_graphics.htm
3. Mathematics for 3D Graphics [Електронний ресурс] — Режим доступу до ресурсу: https://medium.com/imagecraft/mathematics-for-3d-graphics-with-opengl-800e9c10e2df
4. CUDA Architecture Overview [Електронний ресурс] — Режим доступу до ресурсу: https://docs.nvidia.com/cuda/cuda-c-programming-guide/
5. OpenCL System Architecture [Електронний ресурс] — Режим доступу до ресурсу: https://www.intel.com/content/www/us/en/docs/programmable/683045/18-1/opencl-system-architecture.html

Сирота Олексій Костянтинович - студент групи 1КІ-24м, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: a.surota2003@gmail.com

Крупельницький Леонід Віталійович - канд. техн. наук, доцент кафедри ОТ, Вінницький національний технічний університет, Вінниця, e-mail: krup@vntu.edu.ua

Oleksii Kostyantynovich Syrota - student of group 1KI-24м, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: a.surota2003@gmail.com

Krupelnytskyi Leonid Vitaliyovich - Candidate of Technical Sciences, Associate Professor of the Department of OT, Vinnytsia National Technical University, Vinnytsia, e-mail: krup@vntu.edu.ua