Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

КОМПЛЕКСНЕ ДОСЛІДЖЕННЯ МЕТОДІВ ПРОГРАМНОЇ СПЕЦІАЛІЗАЦІЇ ДЛЯ ГРАФІЧНИХ ПРОЦЕСОРІВ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-19-2
Стребков Г.С.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір В.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Стребкову Георгію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Комплексне дослідження методів програмної спеціалізації для графічних процесорів

затверджена наказом по університету від «23» жовтня 2020 року №1428Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 1 грудня 2020 р.

3. Вихідні дані до роботи *технічні специалізації та протоколи, операційні системи, перелік обладнання*

4. Перелік питань, що потрібно опрацювати в роботі

1. Виконати огляд існуючих програмних інструментів для спеціалізації програм з підтримкою графічних процесорів.
2. Виділити алгоритми з перспективних областей, теоретично піддаються спеціалізації, а також розпаралелюванню для GPU.
3. Провести експериментальне дослідження ефективності спеціалізації виділених алгоритмів з обраним спеціалізатором на центральному процесорі.
4. Дослідити можливість застосування та ефективність спеціалізації обраних алгоритмів на GPU з вибраним спеціалізатором.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Графіки порівняння роботи різних приладів, допоміжні схеми та рисунки, тестові зображення*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	23.10.2020	
2	Аналіз завдання, підбір літератури	01.11.20-07.11.20	
3	Аналіз літератури з досліджуваної проблеми	07.11.20-14.11.20	
4	Аналіз технічних засобів	14.11.20-15.11.20	
5	Розробка методу	15.11.20-16.11.20	
6	Програмна реалізація	16.11.20-21.11.20	
7	Оформлення пояснювальної записки	21.11.20-25.12.20	
8	Перевірка на плагіат	03.12.20	
9	Рецензування	04.12.20	
10	Підготовка презентації та доповіді	05.12.20	
11	Занесення роботи в електронний архів	05.12.20	
12	Попередній захист атестаційної роботи	07.12.20	

Дата видачі завдання 23 жовтня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Машталір В. П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи: 82 с., 36 рис., 34 джерела.

МЕТОДИ ПРОГРАМНОЇ СПЕЦІАЛІЗАЦІЇ ДЛЯ ГРАФІЧНИХ ПРОЦЕСОРІВ, РОБОТА ЦЕНТРАЛЬНОГО ПРОЦЕСОРА, РОЗПАРАЛЕЛЮВАННЯ, ШВИДКОДІЯ ГРАФІЧНИХ ПРОЦЕСОРІВ.

Метою дослідження є програмування алгоритмів для розпаралелювання та результати тестування різного забезпечення.

Об’єктами дослідження є GPU та CPU.

Використано алгоритми з перспективних областей, що теоретично піддаються спеціалізації, а так само розпаралелюванню для відеокарт. Проведено дослідження пошуку найліпших та найактуальніших алгоритмів для використання графічними процесорами. Застосовано спеціальну програму для перевірки швидкодії цих алгоритмів. Наведені графіки швидкодії графічних процесорів при різних умовах. А також використані різні графічні карти та програмне забезпечення для дослідження обробки інформації в відеофайлах.

У результаті роботи здійснена програмна реалізація алгоритмів для перевірення швидкодії графічних процесорів та випробувано різне обладнання для перевірки його швидкодії.

METHODS OF SOFTWARE SPECIALIZATION FOR GRAPHIC PROCESSORS, CENTRAL PROCESSOR OPERATION, PARALLELING, GRAPHIC PROCESSOR SPEED.

The aim of the research is to program algorithms for parallelization and test results of various software.

The objects of research are GPU and CPU.

Algorithms from promising areas that are theoretically subject to specialization, as well as parallelization for video cards are used. A study of the search for the best and most relevant algorithms for the use of graphics processors. A special program was used to check the speed of these algorithms. Graphs of GPU performance under different conditions are given. And also used various graphics cards and software to study the processing of information in video files.

As a result, the software implementation of algorithms for checking the performance of graphics processors and tested various equipment to check the performance of various equipment.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

GPU – Graphics Processing Unit

GPGPU – General-Purpose Computation Using Graphics Hardware

BIOS – Basic Input/Output System

FSAA – Full Scene Anti-Aliasing

DAC – Digital to Analog Converter

RAMDAC – Random Access Memory Digital to Analog Converter

OpenACC – Open Accelerators

CUDA – Compute Unified Device Architecture

OpenCL – Open Computing Language

OpenMP – Open Multi-Processing

CPU – Central Processing Unit

HPC – High-performance Computing

PDE – Plugin Development Environment.

PCI – Peripheral Component Interconnect

AGP – Accelerated Graphics Port

DVI – Digital Visual Interface

VGA – Video Graphics Array

ADC – Analog to Digital Converter

FPS – Frames per Second

CAD – Computer-Aided Design

CAE – Computer-Aided Engineering

HDAO – High-Definition Ambient Occlusion

API – Application Programming Interface

SDK – Software development Kit

VIA – Very Innovative Architecture

ВСТУП

По мірі наростання складності програмних систем, а також потреби в прискоренні розробки програм в різних областях програмної інженерії все більше проявляється необхідність впровадження якісної автоматичної оптимізації коду. Одним з методів проведення подібної оптимізації є програмна спеціалізація, яку ще називають частковими або змішаними обчисленнями.

Як показали дослідження Н. Джонса [1], спеціалізація може бути застосована до широкого класу програм. Особливо, перспективними областями для застосування даного методу дослідником були названі: обробка запитів до баз даних, комп'ютерна графіка та нейронні мережі. Дійсно, в 1996 році була опублікована стаття [2] про спеціалізацію алгоритму трасування променів, а в 2017 році з'явилася стаття [3], в якій наводилися дослідження спеціалізації запитів до реляційної системі управління базами даних PostgreSQL. В роботах була показана доцільність використання спеціалізації для підвищення ефективності в перспективних на даний момент областях, що підтверджує припущення про актуальність дослідів в цій галузі.

В останні роки набули широкого поширення обчислення на графічних процесорах (GPU), що дозволяють виконувати задачі паралельно з використанням великої кількості потоків [4]. У зв'язку з цим високий академічний інтерес набуває використання вже відомої техніки спеціалізації в новій області – обчислення спільного призначення на графічних процесорах (GPGPU). Говорячи про практичну значущість спеціалізації на GPU, в багатьох областях індустрії застосовуються алгоритми, які можуть бути ефективно виконані на відеокартах. Наприклад, графові, строкові і інші алгоритми активно використовуються при обробці запитів до популярних в індустрії графовим базам даних [5]. Оптимізація цих алгоритмів може покращити ефективність взаємодії з базами даних, що є важливим практичним результатом.

1 ЯК ПРАЦЮЮТЬ ГРАФІЧНІ КАРТИ

Зображення, які ви бачите на своєму моніторі, складаються з крихітних точок, які називаються пікселями. За найпоширеніших налаштувань роздільної здатності екран відображає понад мільйон пікселів, і комп'ютер повинен вирішити, що робити з кожним із них, щоб створити зображення. Для цього йому потрібен перекладач – щось, щоб взяти двійкові дані з центрального процесора і перетворити їх на зображення, яке можна побачити. Якщо комп'ютер не має графічних можливостей, вбудованих у материнську плату, цей переклад відбувається на відеокарті.

Робота графічної карти складна, але її принципи та компоненти легко зрозуміти. Процесор, що працює разом із програмним забезпеченням, надсилає інформацію про зображення на графічну карту. Відеокарта вирішує, як використовувати пікселі на екрані для створення зображення. Потім вона передає цю інформацію на монітор за допомогою кабелю.

Створення зображення з двійкових даних – це складний процес. Для створення тривимірного зображення відеокарта спочатку створює дротяну рамку з прямих ліній. Потім вона растеризує зображення (заповнює решту пікселів). Це також додає освітлення, текстури та кольору. Для швидко модернезуючихся ігор комп'ютер повинен проходити цей процес приблизно шістдесят разів на секунду. Без графічної карти для виконання необхідних обчислень навантаження було б занадто великим для комп'ютера.

Відеокарта виконує це завдання, використовуючи чотири основні компоненти:

– підключення материнської плати для передачі даних та живлення;

– процесор, який вирішує, що робити з кожним пікселем на екрані;

– пам'ять для зберігання інформації про кожен піксель та для тимчасового зберігання завершених зображень;

– підключення монітора, щоб ви могли бачити кінцевий результат.

Якщо розглядати процесор та пам'ять більш докладніше.

1.1 Дослідження GPU

Як і материнська плата, відеокарта – це друкована плата, на якій розміщений процесор і оперативна пам’ять. Він також має мікросхему системи введення / виводу (BIOS), яка зберігає налаштування карти і виконує діагностику в пам’яті, введення та виведення під час запуску. Процесор графічної карти, який називається графічним процесором (GPU), схожий на процесор комп’ютера. Однак графічний процесор розроблений спеціально для виконання складних математичних та геометричних обчислень, необхідних для візуалізації графіки.

Деякі з найшвидших графічних процесорів мають більше транзисторів, ніж середній процесор. Графічний процесор виробляє багато тепла, тому він зазвичай знаходиться під радіатором або вентилятором.Окрім потужності обробки, графічний процесор використовує спеціальне програмування, яке допомагає аналізувати та використовувати дані. ATI та nVidia виробляють переважну більшість графічних процесорів на ринку, і обидві компанії розробили власні вдосконалення для продуктивності графічного процесора.

Для поліпшення якості зображення процесори використовують:

– повна сцена згладжування (FSAA), яка згладжує краї тривимірних об’єктів;

– анізотропна фільтрація (AF), завдяки якій зображення виглядають чіткішими.

Кожна компанія також розробила специфічні методи, які допомагають графічному процесору застосовувати кольори, тіні, текстури та візерунки. Оскільки графічний процесор створює зображення, йому потрібно десь зберігати інформацію та завершені зображення. Для цього він використовує оперативну пам’ять карти, зберігаючи дані про кожен піксель, його колір та розташування на екрані. Частина оперативної пам’яті може також виконувати функцію буфера кадру, що означає, що вона зберігає завершені зображення, поки не настає час їх відображення. Як правило, оперативна

пам’ять відео працює на дуже високих швидкостях і має подвійний порт, що означає, що система може одночасно читати з неї та писати на неї.Оперативна пам’ять підключається безпосередньо до цифро-аналогового перетворювача, який називається DAC [6].

Цей перетворювач, який також називають RAMDAC, перетворює зображення в аналоговий сигнал, який монітор може використовувати. Деякі картки мають кілька RAMDAC, які можуть покращити продуктивність і підтримувати більше одного монітора. Ви можете дізнатись більше про цей процес у розділі «Як працює аналоговий та цифровий запис». RAMDAC надсилає остаточне зображення на монітор за допомогою кабелю.

1.2 Обчислення на GPU за допомогою простих директив

У листопаді 2011 року був анонсований стандарт OpenACC спільне дітище суперкомп’ютерних гігантів CRAY, CAPS і PGI і лідера ринку графічних процесорів NVIDIA. Сам стандарт покликаний значно спростити роботу програміста і створити високорівневу прошарок над уже відомими CUDA і OpenCL. Варто зазначити, що до недавнього часу Стандарт не підтримувався в повній мірі жодним компілятором, але навіть те, що вже є, вражає своєю простотою і результативністю. Тепер написання програми, що виконується паралельно на тисячах ядер сучасних GPU не вимагає майже ніяких зусиль і практично повністю перекладається на компілятор. Все що потрібно зробити розставити директиви по коду на манер OpenMP. Набір директив досить великий і за один день його повністю не освоїти, але найпростішу програму можна зробити за п’ять хвилин, особливо якщо є однопоточні реалізація. Звідси і випливає основна ідея заховати від розробника майже всі деталі архітектури, звільнити його від тонкощів (а адже до появи CUDA використовувати GPU могли тільки знавці шейдеров) і залишити час на роботу над науковим або призначеним для користувача

проектом. Як і його прабатьки (PGI accelerator і CAPS HMPP) OpenACC підтримує мови C і Fortran. Отже, всі директиви в C-версії стандарту починаються як зазвичай з #pragma, далі ставиться специфікатор acc і одна з основних директив, доповнена одним, або декількома умовами. Найчастіше використовуються три директиви: parallel, kernels і data, одна з яких зазначені на рисунку 1.1.

```
 1. #include <openacc.h>
 2. #include <stdio.h>
 3. #include <stdlib.h>
 4. void main() {
 5.   int n = 100;
 6.   float a[n][n];
 7.   float b[n][n];
 8.   float c[n][n];
 9.   float elements [n];
10.   for(int i = 0; i < n; i++)
11.    for (int j=0; j<n; j++){
12.     a[i][j] = i+j;
13.     b[i][j] = 100 + 2 * i;
14.   }
15. #pragma acc kernels loop independent
16.   for(int i = 0; i < n; i++)
17.     for (int j=0; j < n; j++){
18.         for (int k=0; k<n; k++)
19.                 c[i][j]=+a[i][k]*b[k][j];
20.         }
21.   free(a); free(b); free(c);
22. } // main
```

Рисунок 1.1 – Простий приклад як можна прискорити множення матриць

Ця програма відрізняється від простої версії, виконуваної на одному ядрі CPU тільки п’ятнадцятим рядком, де можно бачити директиву kernels, яка говорить компілятору створити потоки, згруповані в кілька блоків, кількість яких він вибирає на свій розсуд. Крім того, тут же додана директива loop, після якої має починатися цикл, loop служить для того, щоб вказати, як виконувати ітерації циклу: independent незалежно, seq послідовно.

Розглянемо короткий опис деяких директив і умов до них:

– директива parallel вказує на необхідність розпаралелювання. Компілятор, проводячи аналіз коду, визначає необхідність виконання різних його частин на GPU, або на хості;

– директива kernels аналог parallel, вказує на те, що для кожного нового циклу необхідно створити окрему __device__ функцію;

– директива loop передує оператору циклу і використовується для специфікації його властивостей. Сучасні компілятори не вимагають її явної вказівки.

Всі умови передачі даних вимагають вхідні дані, що виглядають наступним чином: a [start: length], де a – масив, або покажчик на нього, start – номер стартового елемента для копіювання, a length – довжина регіону даних, що копіюється на GPU, або з нього; start і length вказуються в елементах масиву. Нижче представлені ті з них, які використовуються найчастіше:

– copy говорить компілятору скопіювати дані на пристрій перед виконанням ядра і тому після його завершення;

– copyin вказує, що дані на GPU використовуються тільки для читання, і немає необхідності копіювати їх назад на хост;

– copyout дані з’являться тільки в результаті виконання ядра на GPU і ніяк не залежать від попередніх значень за цією адресою, їх потрібно скопіювати на хост після виконання кернела;

– create виділяє в пам’яті пристрою місце для даних, які не потребують будь-якого копіювання, наприклад масив для зберігання проміжних результатів;

– present підказує компілятору, що ці дані вже були передані на пристрій раніше. Викликає помилку, якщо даних на GPU немає.

До плюсів можна віднести: високий ступінь абстракції і кроссплатформенность, відразу після виходу нової архітектури

необов’язково переписувати весь код, більшу частину компілятор зробить за нас. Наприклад, CAPS HMPP вже оголосив про підтримку прискорювачів не тільки NVIDIA, але і Intel MIC і навіть AMD FirePro.

До мінусів можно выднести: найперше, що кидається в очі все компілятори з підтримкою OpenACC коштують грошей. Другий мінус – продуктивність: жоден компілятор не зможе оптимізувати код краще, ніж це можна зробити вручну, або з використанням бібліотек від NVIDIA.

Можна відзначити, що OpenACC і правда дає можливість по-швидкому переписати свої проекти під використання GPU і практично не вимагає навичок їх програмування. З його допомогою вже прискорені десятки проектів в областях вивчення і прогнозування поведінки атмосфери, газодинаміки, гідродинаміки і фінансових потоків. П’ять років тому почалася революція масивно-паралельних обчислень і на сьогодні OpenACC – кращий спосіб залишитися на плаву, не втративши позиції і не витратити сотні годин на вивчення всіх тонкощів CUDA або OpenCL.

1.3 Різниця GPU та CPU

Графічний процесор може виконати лише частину операцій, які може виконати центральний процесор, але він робить це з неймовірною швидкістю. GPU буде використовувати сотні ядер, щоб виконати термінові обчислення для тисяч пікселів і відобразити при цьому складну 3D графіку. Але для досягнення високих швидкостей GPU повинен виконувати одноманітні операції.

Візьмемо, наприклад, Nvidia GTX 1080. Дана відеокарта має 2560 шейдерних ядер. Завдяки цим ядер Nvidia GTX 1080 може виконати 2560 інструкцій або операцій за один такт. Якщо ви захочете зробити картинку на один відсоток яскравіше, то GPU з цим впорається без особливих зусиль. А ось чотирьохядерний центральний процесор Intel Core i5 зможе виконати

тільки 4 інструкції за один такт. Проте, центральні процесори більш гнучкі, ніж графічні, це добре показано на рисунку 1.2.

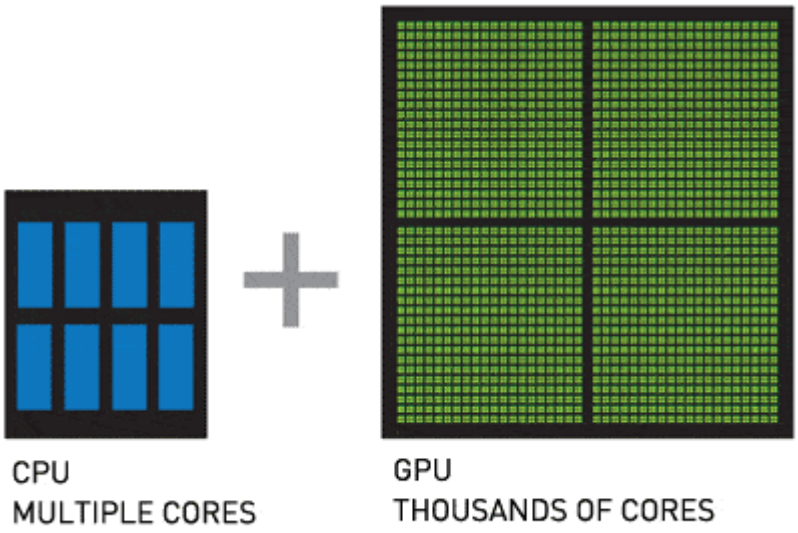

Рисунок 1.2 – Ядра GPU та CPU.

Центральні процесори мають більший набір інструкцій, тому вони можуть виконувати більш широкий діапазон функцій. Також CPU працюють на більш високих максимальних тактових частотах і мають можливість управляти введенням і виведенням компонентів комп'ютера. Наприклад, центральний процесор може інтегруватися з віртуальною пам'яттю, яка необхідна для запуску сучасної операційної системи. Це якраз те, що графічний процесор виконати не зможе.

Спочатку, дані просто передавалися в одному напрямку з центрального процесора до GPU, а потім до пристрою відображення. Однак, з плином часу, для графічних процесорів стало цінним зберігати спершу прості, потім складні структури даних, які передаються назад в процесор, що аналізує зображення, або набір науково представлених даних у форматі 2D або 3D, що відеокарта може зрозуміти. Так як GPU має доступ до кожної операції малювання, він може аналізувати дані в цих формах дуже швидко; в той час як процесор повинен опитати кожен піксель або елемент даних набагато повільніше, так як швидкість доступу між центральним процесором і його

більшим простором оперативної пам’яті (або ще у гіршому випадку, жорсткого диску) є повільніша, ніж на графічних процесорах і відеокартах, які, як правило, містять менші кількості більш дорогої пам’яті, до якої доступ можна отримати набагато швидше.

Передача частини набору даних, які будуть проаналізовані на цю GPU пам’ять у вигляді текстур або інших легко зрозумілих GPU форм призводить до збільшення швидкості. Відмінною особливістю конструкції GPGPU є здатність передавати інформацію в обох напрямках назад від GPU до CPU; як правило, швидкість передачі даних в обох напрямках є ідеально висока, в результаті чого ефект множення впливає на швидкість конкретного алгоритму високого використання. GPGPU конвеєри можуть підвищити ефективність на особливо великих наборах даних та / або даних, що містять 2D або 3D зображення. Він використовується в складних графічних конвеєрах, а також наукових обчисленнях; зокрема, в областях з великими наборами даних, таких як відображення геному, або там, де двовимірний або тривимірній аналіз корисний – зокрема, аналіз біомолекул, дослідження білків, і в інших складних галузях органічної хімії. Такі конвеєри можуть також значно поліпшити ефективність обробки зображень та комп’ютерного зору, поміж інших областей.

GPGPU – це концепт програмного забезпечення, а не апаратного. Тим не менш, спеціалізовані конструкції устаткування можуть ще більше підвищити ефективність GPGPU трубопроводів, які традиційно виконують відносно мало алгоритмів на дуже великих обсягах даних. Масово розпаралелені завдання гігантських рівнів даних, таким чином, можуть бути розпаралелені ще більше за допомогою спеціалізованих установок, таких як обчислювальні стійки (багато подібних, вузькоспеціалізованих машин, побудованих в «стійці»), який додає третій шар – багато обчислювальних блоків кожен з яких використовує багато центральних процесорів щоб відповідати багатьом графічним.

1.4 Дослідження GPGPU

GPGPU – це методологія високопродуктивних обчислень використовує блоки обробки графіки для дроблення даних. Характеристики графічних алгоритмів, що дозволили розробити надзвичайно високопродуктивні графічні процесори спеціального призначення відображаються в інших алгоритмах HPC. Це ж спеціальне призначення апаратне забезпечення також можна використовувати для прискорення цих алгоритмів.

Для реалізації GPGPU добре підходять алгоритми, які мають дві властивості: вони є паралельними даними та пропускною здатністю інтенсивний. Паралельні дані означають, що процесор може виконувати операцію над різними елементами даних одночасно. Пропускна здатність Інтенсивний означає, що алгоритм буде обробляти багато елементів даних, тому буде багато можливостей для паралельної роботи. Користуючись перевагами цих двох властивостей, графічні процесори досягають надзвичайної продуктивності, включаючи партії (сотні) відносно простих одиниці обробки для одночасної роботи з багатьма елементами даних. Мабуть, не дивно, що піксельні програми, такі як комп’ютерний зір та обробка відео та зображень, дуже добре підходять для GPGPU з цієї причини багато комерційних програмних пакетів у цих областях тепер включають прискорення GPGPU. Фізичні симуляційні програми, які часто покладаються на чисельні рішення для PDE та на інтенсивне використання операцій лінійної алгебри, також дуже добре підходить для технології GPGPU. У області візуалізації трасування променів використовує технологію GPGPU для обчислення перетинів об’єктів променів, що робить трасування променів у реальному часі досяжним.

Обчислення GPGPU роблять значний вплив щодо високопродуктивних обчислень у широкому діапазоні домени додатків. Широко використовувані коди HPC в районах включаючи прогноз погоди, молекулярну динаміку та потоки рідини оновлюються з урахуванням прискорення GPGPU. Зараз

популярні комерційні наукові та інженерні програми забезпечують прискорення GPGPU, зокрема MATLAB та ANSYS; системи з відкритим кодом, включаючи AMBER, LAMMPS, NAMD; і Gromacs використовує технологію GPGPU. Ще більш значущим фактом є широке прийняття технології GPGPU у новому коді, розробка для передових наукових та інженерних досліджень.

Високопродуктивні обчислення давно покладаються на паралельні системи для досягнення продуктивності, необхідної передовій науці та інженерні дослідження. З появою технології GPGPU можливості окремих вузлів значно покращуються, але все ж не забезпечує достатньої продуктивності для поточних програм HPC, які в даний час покладаються на системи в десятки тисяч окремі центральні процесори, які працюють спільно для досягнення необхідної продуктивності. Сервери HPC TACC задовольняють ці потреби, використовуючи архітектуру системи передачі повідомлень із розподіленою пам'яттю. Цей підхід досягає цілі продуктивності, поєднуючи відносно недорогі багатоядерні вузли з високошвидкісним взаємозв'язком для явного передавання повідомлень. Це підхід добре підходить для технології GPGPU – деякі або всі вузли кластера можуть бути розширені за допомогою графічних процесорів, що забезпечують два рівні паралелізм для досягнення екстремальних показників продуктивності. У січні 2010 року TACC встановив Longhorn, візуалізацію та дані Dell з 256 вузлами кластер аналізу з двома потужними графічними процесорами NVIDIA у кожному вузлі. На додаток до його використання як суперкомп'ютера для візуалізації виробництва, він також служить платформою для досліджень та розробок для вивчення того, як можуть бути ефективними кластерні та GPGPU паралельні технології гібридизований.

TACC підтримує як OpenCL, реалізацію NVIDIA відкритого стандарту для обчислень GPGPU, так і CUDA, Запатентована мережа інструментів GPGPU від NVIDIA. NVIDIA вклала значні кошти у розробку повного, надійного набору інструментів, включаючи компілятор, налагоджувач та

профайлер. Крім того, реалізація GPGPU від NVIDIA декількох основних бібліотек HPC, що забезпечують лінійну алгебра і швидке перетворення Фур'є доступні на Longhorn.

1.5 Підключення PCI

Відеокарти підключаються до комп'ютера через материнську плату. Материнська плата подає живлення на карту і дозволяє їй взаємодіяти з центральним процесором. Нові відеокарти часто вимагають більше енергії, ніж материнська плата, тому вони також мають пряме підключення до джерела живлення комп'ютера.

Зазвичай підключення до материнської плати здійснюється через один із трьох інтерфейсів:

- взаємозв'язок периферійних компонентів (PCI);
- розширений графічний порт (AGP);
- PCI Express (PCIe).

PCI Express є найновішим із трьох і забезпечує найшвидші швидкості передачі даних між графічною картою та материнською платою. PCIe також підтримує використання двох графічних карт в одному комп'ютері.

Більшість відеокарт мають два підключення до монітора. Часто, одне з яких – це роз'єм DVI, який підтримує РК-екрани, а інше – роз'єм VGA, який підтримує ЕЛТ-екрани. Натомість деякі відеокарти мають два роз'єми DVI. Але це не виключає використання екрану ЕПТ; ЕЛТ-екрани можуть підключатися до портів DVI через адаптер. Свого часу Apple виготовляла монітори, які використовували фірмовий роз'єм ADC. Хоча ці монітори все ще використовуються, нові монітори Apple використовують підключення DVI.

Більшість людей використовують лише одне з двох підключень монітора. Люди, яким потрібно використовувати два монітори, можуть

придбати відеокарту з подвійною головкою, яка розділяє дисплей між двома екранами. Комп’ютер з двома головками з підтримкою PCIe відеокарт теоретично міг підтримувати чотири монітори.

Окрім підключень до материнської плати та монітора, деякі відеокарти мають підключення для:

- дисплей телевізора: ТВ-вихід або S-відео;
- аналогові відеокамери: ViVo або вхід / вихід відео;
- цифрові камери: FireWire або USB.

Деякі картки також мають телевізійні тюнери. Далі розглянемо, як вибрати хорошу відеокарту.

1.6 Вибір графічної карти

Найпопулярнішу відеокарту легко знайти. Вона має багато пам’яті та швидкий процесор. Часто вона візуально привабливіше, ніж будь-що інше, що має потрапити в корпус комп’ютера. Багато високопродуктивних відеокарт або мають декоративні вентилятори або радіатори.

Але карта високого класу забезпечує більше енергії, ніж насправді потрібно більшості людей. Люди, які використовують свої комп’ютери в основному для електронної пошти, обробки текстів або веб-серфінгу, можуть знайти всю необхідну графічну підтримку на материнській платі з інтегрованою графікою. Карт середнього класу достатньо для більшості випадкових геймерів. Люди, яким потрібна сила висококласної картки, – це любителі ігор та люди, які виконують багато тривимірних графічних робіт.

Хорошим загальним виміром продуктивності картки є FPS. Частота кадрів описує, скільки повних зображень може відображати карта на секунду. Людське око може обробляти близько 25 кадрів щосекунди, але для ігор швидкої дії потрібна частота кадрів не менше 60 кадрів в секунду, щоб забезпечити плавну анімацію та прокрутку.

Компонентами частоти кадрів є:

– трикутники або вершини в секунду: тривимірні зображення складаються з трикутників або багатокутників. Це вимірювання описує, наскільки швидко графічний процесор може обчислити цілий багатокутник або вершини, що його визначають;

– швидкість заповнення пікселів: це вимірювання описує, скільки пікселів графічний процесор може обробити за секунду, що означає, як швидко він може растеризувати зображення.

Апаратне забезпечення відеокарти безпосередньо впливає на її швидкість. Ось технічні характеристики, які найбільше впливають на швидкість картки та одиниці вимірювання:

– тактова частота графічного процесора (МГц);

– розмір шини пам’яті (біти);

– обсяг доступної пам’яті (МБ);

– тактова частота пам’яті (МГц);

– пропускна здатність пам’яті (ГБ / с);

– швидкість RAMDAC (МГц).

Процесор і материнська плата комп’ютера також відіграють важливу роль, оскільки дуже швидка відеокарта не може компенсувати нездатність материнської плати швидко доставити дані. Подібним чином підключення карти до материнської плати та швидкість, з якою вона може отримувати вказівки від центрального процесора, впливають на її продуктивність [7].

1.7 Постановка задачі дослідження

Проблема є актуальною і сьогодні, бо алгоритми, які використовують сучасні графічні процесори можно поліпшувати та прискорювати, що буде давати нові можливості для поліпшення якості графіки.

Об’єктами дослідження є GPU та CPU.

Метою дослідження є програмування алгоритмів для розпаралелювання та результати тестування різного забезпечення. А також дослідження роботи GPU та CPU, вибір алгоритмів для програмної спеціалізації, вивчення принципів роботи GPGPU, дослідження швидкодії різних методів обчислення на GPU та CPU.

Для досягнення мети були поставлені перераховані нижче задачі.

1. Виконати огляд існуючих програмних інструментів для спеціалізації програм з підтримкою графічних процесорів.
2. Виділити алгоритми з перспективних областей, теоретично піддаються спеціалізації, а також розпаралелюванню для GPU.
3. Провести експериментальне дослідження ефективності спеціалізації виділених алгоритмів з обраним спеціалізатором на центральному процесорі.
4. Дослідити можливість застосування та ефективність спеціалізації обраних алгоритмів на GPU з вибраним спеціалізатором.

2 GPGPU ТА РІЗНІ ТЕХНОЛОГІЇ

2.1 Технологія CUDA

CUDA – це паралельна обчислювальна платформа та модель програмування, розроблена Nvidia для загальних обчислень на власних графічних процесорах (графічних процесорах). CUDA дозволяє розробникам пришвидшити обчислювальні програми, використовуючи потужність графічних процесорів для паралелізації частини обчислень. Хоча були запропоновані інші API для графічних процесорів, такі як OpenCL, і є конкурентоспроможні графічні процесори інших компаній, таких як AMD, комбінація графічних процесорів CUDA та Nvidia домінує в декількох областях застосування, включаючи глибоке навчання, і є основою для деяких найшвидші комп'ютери у світі. Відеокарти, можливо, такі ж старі, як і ПК – тобто, якщо вважати монохромний дисплейний адаптер IBM 1981 року графічною картою. До 1988 року ви могли отримати 16-бітну 2D-карту VGA Wonder від ATI (компанія, яку врешті придбала компанія AMD). До 1996 року ви можете придбати 3D-графічний прискорювач у 3dfx Interactive, щоб змогли запускати шутер від першої особи Quake на повній швидкості. Також у 1996 році Nvidia почала намагатися конкурувати на ринку 3D-прискорювачів зі слабкими продуктами, але навчилася, і в 1999 році представила успішну GeForce 256, першу графічну карту, яку назвали GPU. Тоді основною причиною наявності графічного процесора була гра. Лише пізніше люди почали використовувати графічні процесори для математики, науки та техніки.

У 2003 році група дослідників під керівництвом Яна Бака представила Brook – першу широко прийняту модель програмування, яка розширила C за допомогою паралельних даних конструкцій. Пізніше Бак приєднався до Nvidia і очолив запуск CUDA в 2006 році – першого комерційного рішення для обчислень загального призначення на графічних процесорах.

Конкурент CUDA OpenCL був запущений Apple та Khronos Group в 2009 році, намагаючись забезпечити стандарт для різнорідних обчислень, який не обмежувався процесорами Intel / AMD з графічними процесорами Nvidia. Хоча OpenCL звучить привабливо через свою загальність, він не настільки успішно працював з CUDA на графічних процесорах Nvidia, і багато фреймворки глибокого навчання або не підтримують його, або підтримують лише як додаткову думку після виходу їх підтримки CUDA.

Протягом багатьох років CUDA вдосконалював і розширював сферу застосування, більш-менш поступово завдяки вдосконаленим графічним процесорам Nvidia. Починаючи з версії 9.2 CUDA, використовуючи кілька серверних графічних процесорів P100, можна досягти в 50 разів поліпшення продуктивності порівняно з центральними процесорами. V100 (показаний на рисунку 2.1) ще втричі швидший для деяких навантажень.

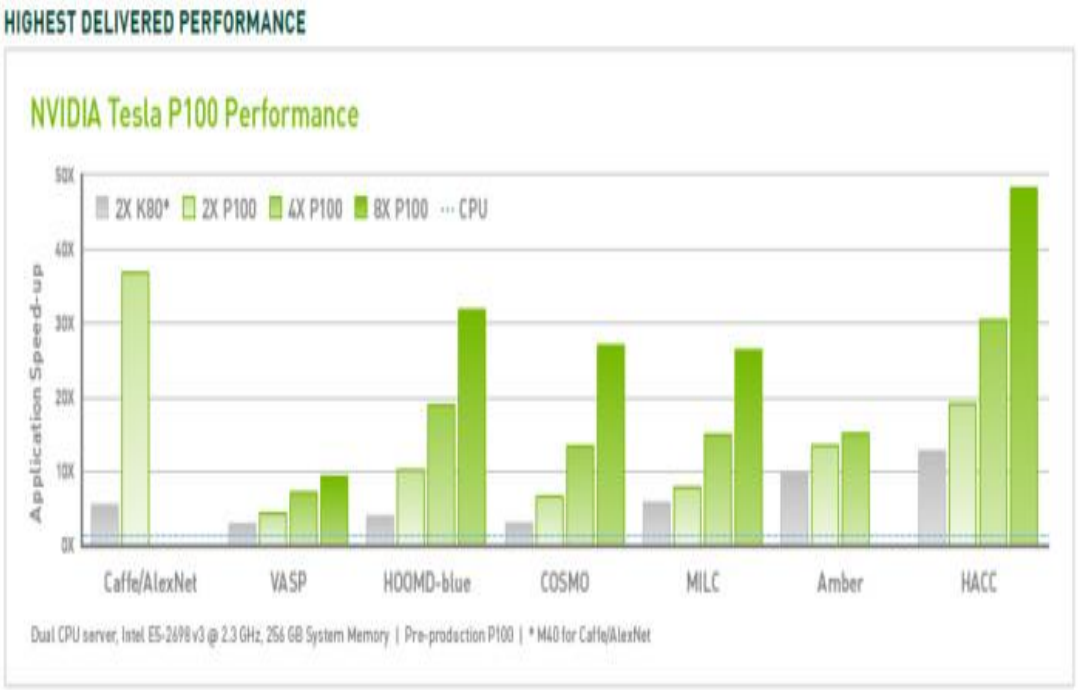

Рисунок 2.1 – Графіки продуктивності NVIDIA

Однопотокове збільшення продуктивності процесорів з часом, яке, за законом Мура, передбачалося подвоювати кожні 18 місяців, сповільнилося до 10 відсотків на рік (детально вказано на рисунку 2.2), оскільки виробники

чіпів стикалися з фізичними обмеженнями, включаючи обмеження розміру роздільної здатності маски чіпа та виходу чіпів під час виробничого процесу і обмеження тепла на тактових частотах під час роботи.

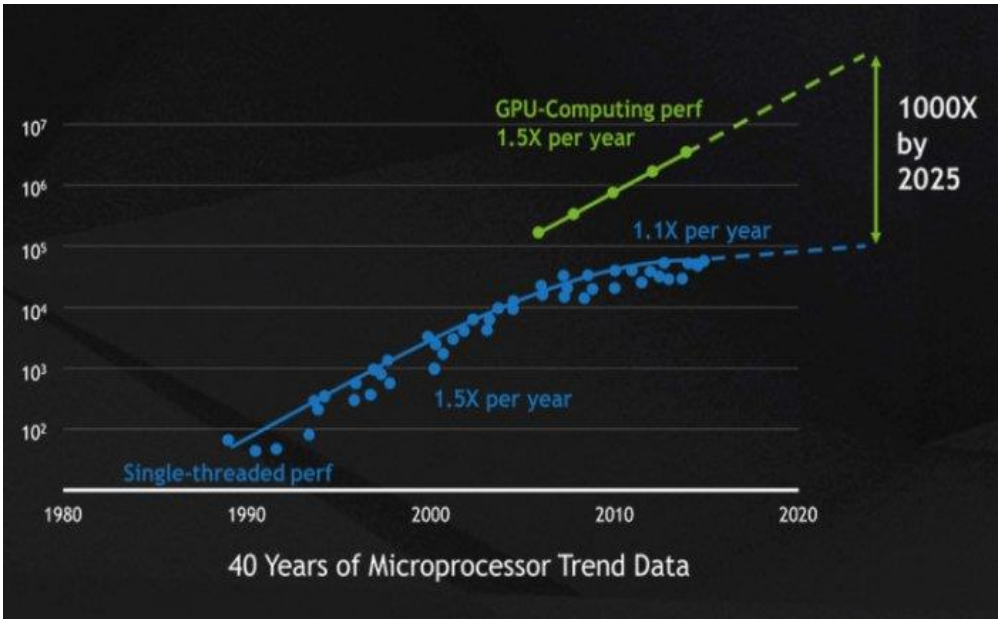

Рисунок 2.2 – Данні мікропроцессорів за 40 років

Графічні процесори CUDA та Nvidia були застосовані у багатьох областях, які потребують високих обчислювальних характеристик з плаваючою точкою, як це зображено на малюнку вище. Більш повний перелік включає:

- обчислювальні фінанси;
- моделювання клімату, погоди та океану;
- наука про дані та аналітика;
- поглиблене навчання та машинне навчання;
- оборона та розвідка;
- виробництво / AEC:
- медіа та розваги;
- медична візуалізація;
- нафта і газ;
- інструменти та управління.

Глибоке навчання має велику потребу в швидкості обчислень. Наприклад, для підготовки моделей для Google Translate у 2016 році команди Google Brain та Google Translate провели сотні тижневих запусків TensorFlow з використанням графічних процесорів; вони придбали у Nvidia 2000 серверних графічних процесорів. Без графічних процесорів ці навчальні прогони зайняли б місяці, а не тиждень, щоб сходитися. Для розгортання цих моделей перекладу TensorFlow Google використовував новий спеціальний процесорний чіп, TPU. На додаток до TensorFlow, багато інших фреймворків DL покладаються на CUDA для підтримки графічного процесора, зокрема Caffe2, CNTK, Databricks, H2O.ai, Keras, MXNet, PyTorch, Theano та Torch. У більшості випадків вони використовують бібліотеку cuDNN для обчислень глибокої нейронної мережі. Ця бібліотека настільки важлива для навчання фреймворків глибокого навчання, що всі фреймворки, що використовують задану версію cuDNN, мають по суті однакові показники продуктивності для еквівалентних випадків використання. Коли CUDA та cuDNN покращуються від версії до версії, усі основи глибокого навчання, які оновлюються до нової версії, бачать приріст продуктивності. Де продуктивність, як правило, відрізняється від фреймворку до того, наскільки якісно вони масштабуються до декількох графічних процесорів та декількох вузлів [9].

Набір інструментів CUDA включає бібліотеки, засоби налагодження та оптимізації, компілятор, документацію та бібліотеку середовища виконання для розгортання ваших програм. Він має компоненти, які підтримують глибоке навчання, лінійну алгебру, обробку сигналів та паралельні алгоритми. Загалом, бібліотеки CUDA підтримують усі сімейства графічних процесорів Nvidia, але найкраще працюють на останньому поколінні, наприклад V100, який може бути в три рази швидшим за P100 для навчальних робочих навантажень (показано на рисунку 2.3). Використання однієї або декількох бібліотек – це найпростіший спосіб скористатися графічними процесорами, якщо алгоритми, які вам потрібні, були реалізовані у відповідній бібліотеці.

Рисунок 2.3 – Глибоке навчання за декілька годин

У сфері глибокого навчання є три основні бібліотеки, прискорені GPU: cuDNN, яку вже згадували раніше як компонент GPU для більшості платформ глибокого навчання з відкритим кодом; TensorRT, який є високоефективним оптимізатором виводу глибокого навчання та середовищем виконання; та DeepStream, бібліотека відеовисновків. TensorRT допомагає оптимізувати нейромережеві моделі, відкалібрувати для меншої точності з високою точністю та розгорнути навчені моделі в хмарах, центрах обробки даних, вбудованих системах або платформах автомобільних продуктів [10].

Усі три бібліотеки для паралельних алгоритмів мають різне призначення. NCCL призначена для масштабування програм на декількох графічних процесорах і вузлах; nvGRAPH – для аналізу паралельних графіків; і Thrust – це бібліотека шаблонів C ++ для CUDA на основі стандартної бібліотеки шаблонів C ++. Thrust забезпечує багату колекцію даних паралельних примітивів, таких як сканування, сортування та зменшення.

У деяких випадках замість еквівалентних функцій центрального процесора можна використовувати функцію CUDA, що випадає. Наприклад,

підпрограми множення матриць GEMM від BLAS можна замінити версіями графічного процесора, просто зробивши посилання на бібліотеку NVBLAS.

2.2 Технологія OpenCL

Основним місцем, де можна зустріти гетерогенні системи, є високопродуктивні обчислення: від моделювання фізичних процесів в прикордонному шарі до кодування відео і рендеринга тривимірних сцен. Раніше подібні завдання вирішували застосовуючи суперкомп'ютери або дуже потужні настільні системи. З появою технологій NVidia CUDA / AMD Stream стало можливим відносно просто писати програми, які використовують обчислювальні можливості GPU.

CUDA стала набирати обертів, а тим часом (а точніше дещо раніше) в кузні, розташованої глибоко під землею, біля підніжжя гори Фуджі (Fuji), японськими інженерами був викуваний процесор всевладдя Cell. В даний час Cell використовується у всіх суперкомп'ютерах, що поставляються IBM, на його основі постоїть найпродуктивніші у світі суперкомп'ютери. Трохи менше року тому компанія Toshiba оголосила про випуск плати розширення SpursEngine для PC для прискорення декодування відео та інших ресурсоємних операцій, використовуючи обчислювальні блоки (SPE), розроблені для Cell.

Отже, маємо: машину, на якій проводяться обчислення. що може містити процесори x86, x86-64, Itanium, SpursEngine (Cell), NVidia GPU, AMD GPU, VIA (S3 Graphics) GPU. Для кожного з цих типів процесів існує свій SDK (ну крім хіба що VIA), своя мова програмування і програмна модель. Тобто можно зробити так, щоб движок рендеринга або програма розрахунку навантажень на крило боїнгу 787 працювала на простий робочої станції, суперкомп'ютері BlueGene, або комп'ютері обладнаному двома прискорювачами NVidia Tesla – буде необхідно переписувати досить велику

частину програми, так як кожна з платформ в силу своєї архітектури має набір жорстких обмежень.

Так як програмісти – не хочуть писати одне і те ж для п’яти різних платформ з урахуванням всіх особливостей і вчитися використовувати різні програмні засоби і моделі, а замовники - народ жадібний і не хочуть платити за програму для кожної платформи як за окремий продукт і оплачувати курси навчання для програмістів, було вирішено створити якийсь єдиний стандарт для програм, що виконуються в гетерогенному середовищі. Це означає, що програма, взагалі кажучи, повинна бути здатна виконуватися на комп’ютері, в якому встановлені одночасно GPU NVidia і AMD, Toshiba SpursEngine і тд. Для розробки відкритого стандарту вирішили залучити людей, у яких вже є досвід в розробці подібного стандарту: Khronos Group, на чиїй совісті вже OpenGL і OpenML і ще багато всього. OpenCL є торговою маркою Apple Inc. У розробці (і фінансуванні), крім Apple, брали участь такі ділки IT як AMD, IBM, Activision Blizzard, Intel, NVidia ітд.

Компанія NVidia особливо не афішувала свою участь в проекті, і швидкими темпами нарощувала функціональність і продуктивність CUDA. Тим часом кілька провідних інженерів NVidia брали участь у створенні OpenCL. Ймовірно, саме участь NVidia великою мірою визначило синтаксичну та ідеологічну схожість OpenCL і CUDA. Втім програмісти від цього тільки виграли – простіше буде перейти від CUDA до OpenCL при необхідності.

Перша версія стандарту була опублікована в кінці 2008 року і з тих пір вже встигла зазнати кілька ревізій.

Майже відразу після того як стандарт був опублікований, компанія NVidia заявила що підтримка OpenCL не складе ніякої складності для неї і незабаром буде реалізована в рамках GPU Computing SDK поверх CUDA Driver API. Нічого подібного від головного конкурента NVidia – AMD чутно не було.

Драйвер для OpenCL був випущений NVidia і пройшов перевірку на відповідність стандарту, але все ще доступний тільки для обмеженого кола людей – зареєстрованих розробників (заявку на реєстрацію подати може будь-хто, в моєму випадку розгляд зайняло 2 тижні, після чого поштою прийшло запрошення). Обмеження доступу до SDK і драйверів змушують задуматися про те, що на даний момент існують якісь проблеми або помилки, які поки не вдається виправити, тобто продукт все ще перебуває в стадії бета-тестування [11].

Реалізація OpenCL для NVidia була досить легким завданням, так як основні ідеї подібні: і CUDA і OpenCL – деякі розширення мови C, з подібним синтаксисом, що використовують однакову програмну модель в якості основної: Data Parallel (SIMD), так само OpenCL підтримує Task Parallel programming model – модель, коли одночасно можуть виконуватися різні kernel (work-group містить один елемент) (показано на рисунку 2.4). Про схожості двох технологій говорить навіть те що NVidia випустила спеціальний документ про те як писати для CUDA так, щоб потім легко перейти на OpenCL.

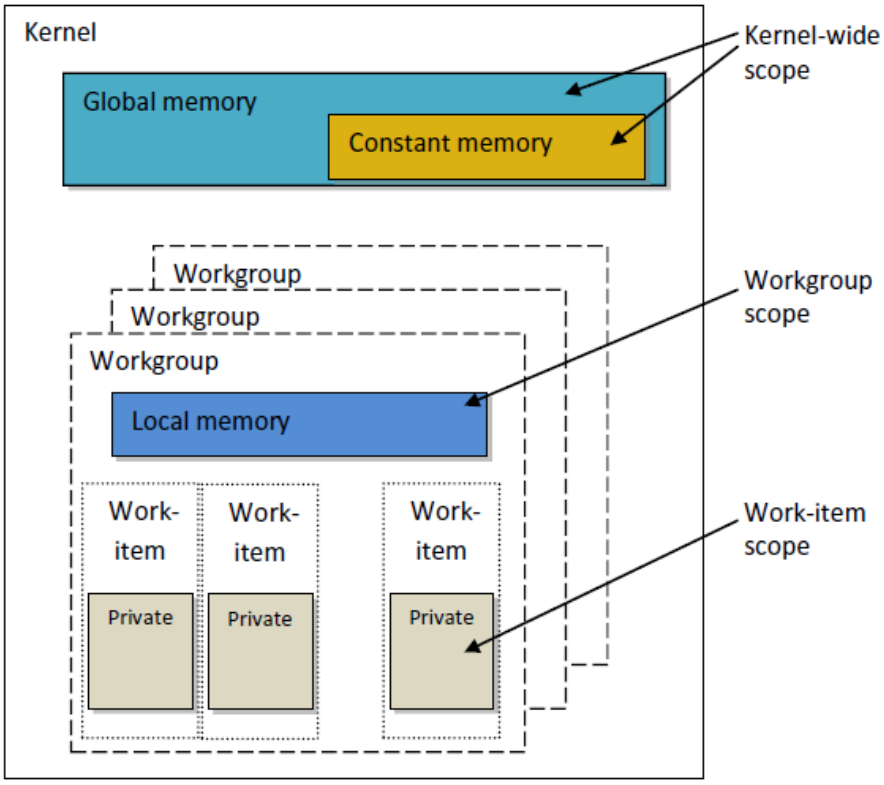

Рисунок 2.4 – Модель пам’яті OpenCL

Основною проблемою реалізації OpenCL від NVidia є низька продуктивність у порівнянні з CUDA, але з кожним новим релізом драйверів продуктивність OpenCL під керуванням CUDA все ближче підбирається до продуктивності CUDA додатків. За заявами розробників такий же шлях пройшла і продуктивність самих CUDA додатків – від порівняно невисокою на ранній версіях драйверів до вражаючою в даний час.

Саме AMD робила великі ставки на нову технологію, враховуючи що AMD Stream не вдавалося хоч скільки-небудь змагатися в популярності з NVidia CUDA – виною тому відставання Stream від CUDA в технічному плані.

Влітку 2009 року компанія AMD зробила заяву про підтримку і відповідно стандарту OpenCL в новій версії Stream SDK. На ділі ж виявилося, що підтримка була реалізована тільки для CPU. Так, саме так, це нічому не суперечить – OpenCL стандарт для гетерогенних систем і нічого не заважає запустити kernel на CPU, більш того – це дуже зручно в разі якщо в системі

немає іншого OpenCL пристрою. В такому випадку програма буде продовжувати працювати, тільки повільніше. Або ж ви можете задіяти всі обчислювальні потужності, які є в комп'ютері – як GPU так і CPU, хоча на практиці це не має особливого сенсу, так як час виконання kernel'ов які виконуються на CPU буде набагато більше тих що виконуються на GPU – швидкість процесора стане вузьким місцем. Зате для налагодження додатків це більш ніж зручно [12].

Підтримка OpenCL для графічних адаптерів AMD так само не змусила себе довго чекати.

Так як OpenCL повинен працювати поверх деякої специфічної для заліза оболонки, а значить для того щоб можна цей стандарт дійсно став єдиним для різних гетерогенних систем – треба щоб відповідні оболонки (драйвери) були випущені і для IBM Cell і для Intel Larrabie. Поки від цих гігантів IT нічого не чути, таким чином OpenCL залишається ще одним засобом розробки для GPU на ряду з CUDA, Stream і DirectX Compute.

Apple також заявляє про підтримку OpenCL, яка, втім, забезпечується за рахунок NVidia CUDA [13].

Технологія OpenCL представляє інтерес для різних компанія IT сфери – від розробників ігор до виробників чіпів, а це означає що у неї великі шанси стати фактичним стандартом для розробки високопродуктивних обчислень, відібравши цей титул у чільної в цьому секторі CUDA.

2.3 Технологія DirectCompute

DirectCompute – інтерфейс програмування додатків (API), який входить до складу DirectX (набору API від Microsoft), який призначений для роботи на IBM PC-сумісних комп'ютерах під управлінням операційних систем сімейства Microsoft Windows. DirectCompute призначений для виконання обчислень загального призначення на графічних процесорах, будучи

реалізацією концепції GPGPU, поряд з програмними інтерфейсами CUDA, ATI Stream і OpenCL. Спочатку DirectCompute був опублікований в складі DirectX 11, проте пізніше став доступний і для DirectX 10.

У теорії, такий підхід до обробки може істотно збільшити пропускну здатність, оскільки у вас буде пам’ять GDDR5 зі швидкістю 4 GT / s замість системної пам’яті зі швидкістю 1600 MT / s або близько того, а також великий приріст обчислювальної продуктивності і продуктивності на ват, коли використовуються вірні алгоритми. Також враховуйте, що хоча CPU все ж перевершує GPU в багатьох задачах, ресурси і затримки, необхідні для комунікації між чіпами, можуть сильніше нашкодити швидкості, ніж виконання завдання тільки на GPU [14].

Використання додаткового потенціалу GPU вимагає додаткової роботи над софтом. Є кілька способів використовувати здібності апаратного забезпечення, і DirectCompute – частина DirectX – це API від Microsoft, що зв’язують GPU і додатки. Раніше було описано, як ранні спроби використовувати універсальні обчислювальні можливості графічного процесора давали користь для різних ринків, таких як ринки дослідження, наукового моделювання і вивчення. Звідти GPGPU почали просочуватися на споживчий ринок, але дуже повільно, особливо відеообробка і перекодування. Тепер можна бачимо, як DirectCompute і OpenCL виконують додаткові функції. Розробники ігор починають застосовувати DirectCompute для поліпшення рендеринга, AI, оклюзії, освітлення і фізики.

Найбільш часто DirectCompute використовується в іграх для технології Ambient Occlusion (спочатку розробленої компанією Industrial Light and Magic більше десяти років тому), яка імітує, як світло взаємодіє з поверхнею і текстурами. Промені виходять від всієї поверхні і ті, які досягають фону, додають яскравість поверхні, в той час як промінь, який вдаряється об інший об’єкт не дає світла, а поглинається. Таким чином, об’єкти, оточені іншими об’єктами темні, а ті, які не оточені перешкодами, більш яскраві (зображено на рисунку 2.5).

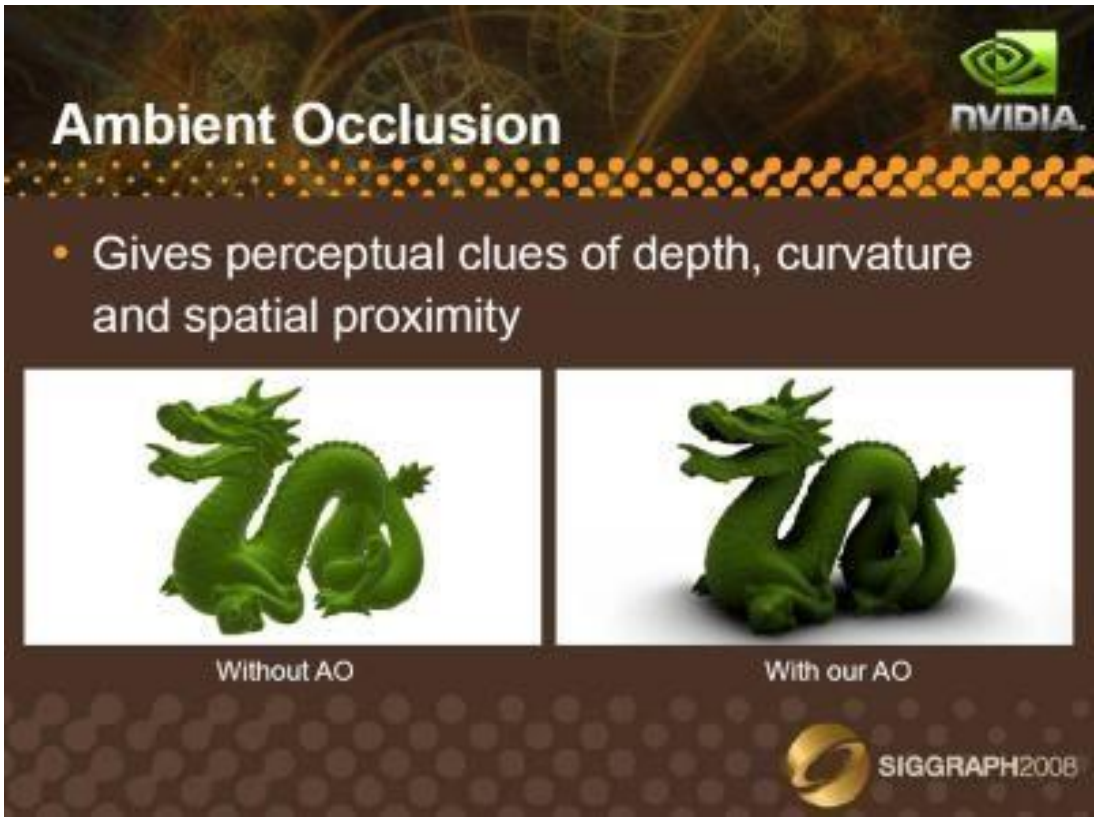

Рисунок 2.5 – Різниця використання затінення фонового освітлення

Затінення фонового освітлення високої чіткості (HDAO) – це екранний ефект, який відповідає за глибину картинки і обчислює коефіцієнт затемнення залежить від того, закрита або відкрита область з точки зору буфера глибини. Наприклад, ущелини між покришками на скріншотах повинні бути темнішими, оскільки менше відбитого світла може досягти цієї області. HDAO можна охарактеризувати як кількість відбитого світла, яке може побачити піксель. Потім він включається в модель освітлення при рендеринге сцени. Це не самий ідеальний метод, оскільки глибина сцени з точки зору камери містить недостатньо інформації для коректного розрахунку AO. Однак так виразно краще, ніж без затінення фонового освітлення взагалі [15].

Затінення фонового освітлення можна також представити через піксельні шейдери. Розробники можуть вибирати між двома методами, і на даному етапі нам незрозуміло, чому DirectCompute може бути краще. Зрештою, в більш ранніх тестах можно побачити як DC-ефекти надавали

значної вплив на продуктивність (і не обов'язково позитивний). Ну, для початку, DirectCompute більше не впливає на продуктивність на відміну від піксельних шейдеров.

Кожен піксель, для якого розраховується AO, вимагає безлічі зчитувань глибини текстури. У піксельні шейдери кожне зчитування текстури займає цикл. В обчислювальному шейдера LDS заповнений інформацією глибини текстури, і послідовні зчитування проходять набагато швидше, ніж текстурная вибірка.

2.4 Технологія C++ AMP

Технологія C++ Accelerated Massive Parallelism (C++ AMP) побудована на платформі Microsoft DirectX, відповідно до нового блогу-посту. Microsoft планує зробити цю технологію частиною наступного компілятора Visual C++ і повністю інтегрувати в наступну версію Visual Studio під кодовим ім'ям Visual Studio vNext. Microsoft анонсировала C++ AMP сьогодні на саміті розробників AMD Fusion. На конференції Герб Саттер розказав присутнім у роботі Microsoft за створенням технологій для використання можливостей C++ на графічному процесорі [16]. Ціль – дозволити розробникам перекомпілювати програму для розподілу виплат між процесорами та процесорами графічного процесора. Конечна мета Microsoft допомагає C++ розробникам використовувати цю модель для багатоядерних та хмарних систем. Microsoft планує надіслати специфікацію C++ AMP неназваною організацією за стандартизацією, щоб технологія стала "відкритою специфікацією", яка могла реалізувати будь-який компілятор. У планах зробити цю технологію доступною на компіляторах для Windows і не на платформі Windows.

Ключові моменти щодо C++ AMP, які можуть послужити вступом та поширеними запитаннями:

– знижує бар’єр для неоднорідної програмної апаратної програмованості та приносить продуктивність до основного потоку, не жертвуючи продуктивністю розробника та портативністю рішення;

– розроблений не лише для того, щоб допомогти вам вирішити суттєво паралельне обладнання (тобто графічні процесори та APU), але він також підтверджує ваші інвестиції в код завдяки перспективному дизайну;

– є частиною Visual C++. Вам не потрібно використовувати інший компілятор або вивчати інший синтаксис;

– є сучасним C++. Не C або якась інша похідна;

– інтегрований та повністю підтримується у Visual Studio 11.

Редагування, побудова, налагодження, профілювання та всі інші переваги Visual Studio добре працюють із C++ AMP;

– надає STL-подібну бібліотеку як частину існуючого простору імен паралельності та доставляється у новому заголовковому файлі amp.h;

– дозволяє надзвичайно легко працювати з великими багатовимірними даними на неоднорідному обладнанні; таким чином, що викриває паралелізацію;

– представляє лише одне основне розширення мови C++;

– базується на DirectX (або DirectCompute), який пропонує чудовий апаратний рівень абстракції, який є повсюдним і надійним. Архітектура така, що цей момент можна сприймати як деталь реалізації, яка не виходить на поверхню API.

2.5 Технологія ATI Stream

ATI Stream, є однією з переважаючих обчислювальних платформ на GPU. Ідея обчислень на GPU криється в тому, щоб перенести виконання високо паралельних завдань з CPU на GPU, які при цьому будуть працювати швидше і ефективніше. Програмовані блоки шейдеров дуже добре підходять для розрахунків з плаваючою комою [17]. Кожен блок шейдерів має свого роду власним обчислювальним ядром, тому замість чотирьох або восьми потоків, які працюють паралельно на CPU, ви можете отримати 64 або 320 або іншу кількість потокових процесорів, які виконують роботу паралельно на GPU.

Коли архітектура Stream була оголошена, AMD використовувала її тільки для прискорення кодування у формати MPEG-2 і H.264. І прискорення дійсно було. Однак AMD не врахувала, що її будуть критикувати за якість кодування. Втім, з виходом травневого драйвера Catalyst 9.5 проблеми якості були вирішені, і тепер отримано більш повний конвеєр прискорення, що містить декодування MPEG-2 і H.264, а також зміна дозволу [18].

З випуском драйвера ATI Catalyst 8.12 AMD офіційно надала доступ до Stream для масових користувачів, а щоб показати можливості технологій, AMD перетворила ATI Avivo Video Converter в Stream-сумісний (зображено на рисунку 2.6). Подія дійсно була значущою, а AMD пропонувала технологію всім, хто нею цікавився. На жаль, програма була з багами та недоліками. Початкові позитивні враження від приросту продуктивності швидко зникали через критичні проблем з виведенням.

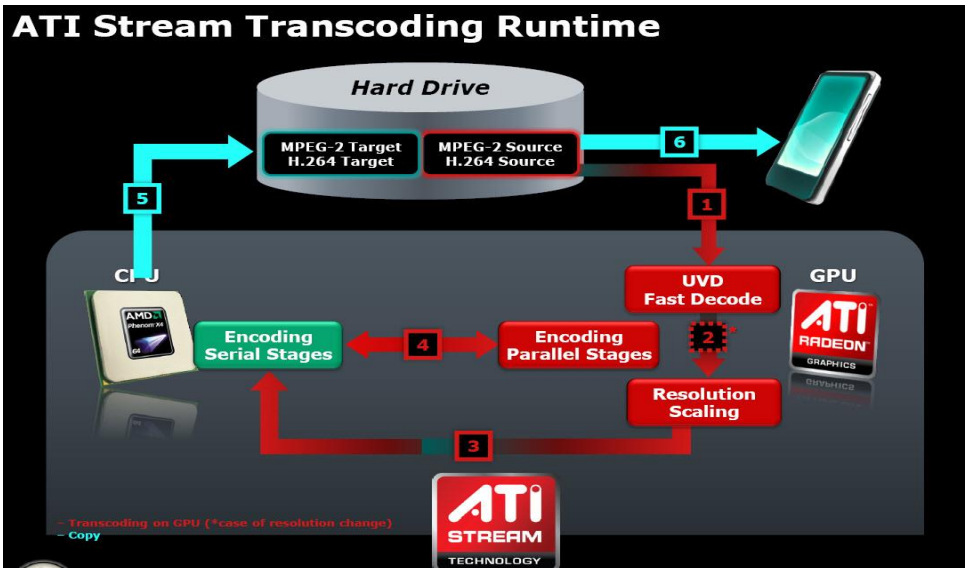

Рисунок 2.6 – ATI Stream

Очікувалося, що AMD піде за лідером nVidia, представивши сумісні пропозиції. Все це здавалося розумним, враховуючи, що в презентації Catalyst 8.12 в листопаді 2008 AMD представила слайд (який зображено на рисунку 2.7), в якому були вказані розробники, готові підтримати оголошення Stream – включаючи досить великих гравців. У Adobe навіть був окремий слайд з Acrobat Reader, Photoshop CS4 Extended, After Effects CS4 і Flash 10. Microsoft теж отримала свій слайд з Vista, Expression Encoder, PowerPoint 2007 і Silverlight.

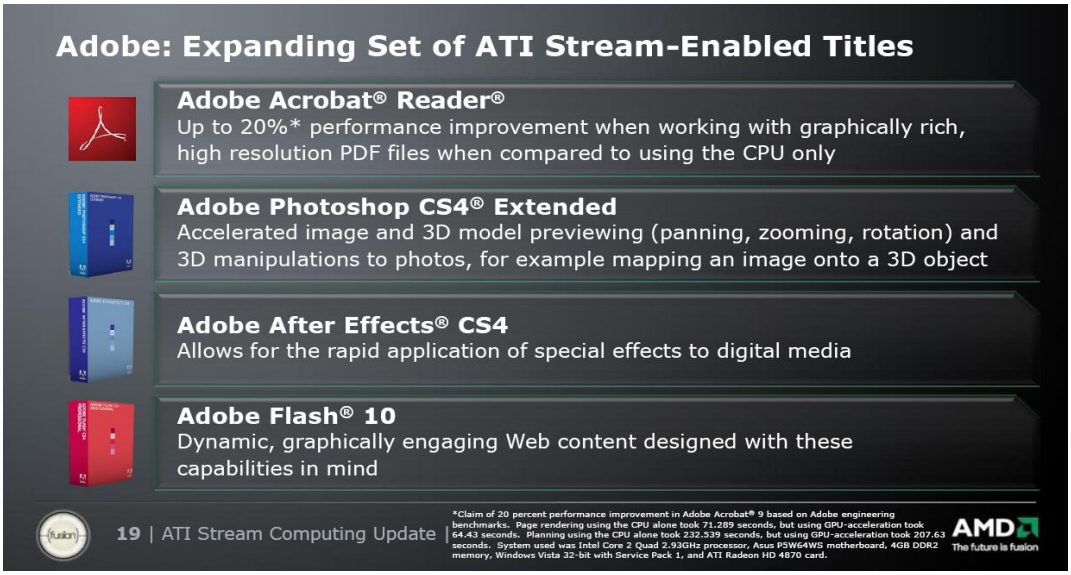

Рисунок 2.7 – Окремий слайд Adobe

Насправді підтримали технологію Stream – CyberLink, з програмами PowerDirector 7 і MediaShow Espresso, а також ArcSoft з плагіном SimHD для плеєра TotalMedia Theatre 3. Розробник LoiLo був в презентації AMD Stream, але компанія поки ще дописувала код з оптимізаціями Stream для свого редактора відео. Попередній відео в LoiLoScope отримав UVD-прискорення ще в січні, але це все ж таки дещо інше, ніж прискорення Stream. Минуло не так багато часу, і презентація повторного оголошення Stream була змінена з активною присутністю Espresso і майже повним ігноруванням PowerDirector 7Щоб отримати повний виграш від Stream, потрібно було запускати Espresso [19].

Одна з найбільших переваг Badaboom криється в досить низькому навантаженні CPU в порівнянні з конкуруючими кодирувальниками CUDA: близько 60% замість звичного рівня 95%. Перекодування вимагає дуже серйозних обчислювальних ресурсів, а розробники програмного забезпечення хочуть зробити свої продукти максимально швидкими. Якщо це призведе до повного завантаження CPU і GPU [20], витісняючи всі інші завдання, то розробники все одно підуть на такий крок. Концепція збалансованої платформи (рисунок 2.8) у AMD намагається зрівняти кілька основних компонентів системи і розподілити навантаження настільки рівномірно, наскільки це можливо, при цьому виконуючи попередню кількість роботи в колишню кількість часу, але забезпечуючи достатній запас для інших додатків, щоб вони працювали нормально [21].

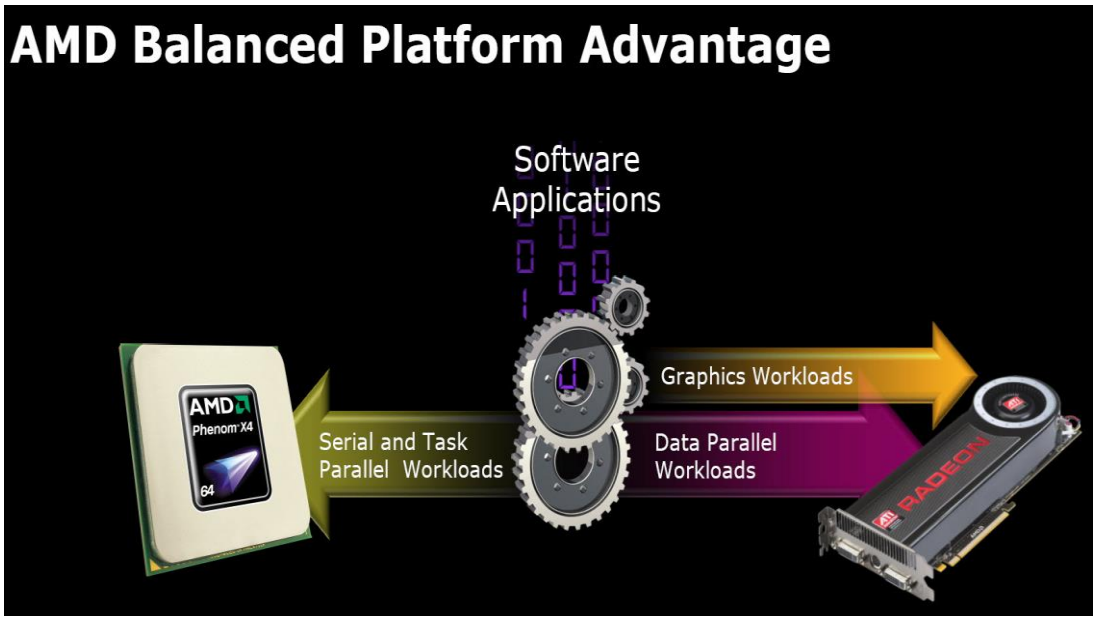

Рисунок 2.8 – Збалансована платформи

Існує приклад збалансованої платформи, зроблений у момент праці над найпершим тестом Espresso, в якому бралося HD-відео YouTube (MPEG-4, 1280x720) і перекодувати в профіль iPhone 640x360, теж H.264 MPEG-4. На рисунку 2.9 та рисунку 2.10 приведена продуктивність в системі з HD 4890. Зверху можно побачити тест без прискорення на GPU, а знизу – з активною підтримкою Stream. Можна побачити, що при кодуванні тільки на CPU всі чотири ядра Phenom II завантажені фактично на максимум, а GPU-Z вказує досить стабільне навантаження на GPU на мінімумі шість відсотків, тобто під час перекодування використовуються деякі елементи конвеєра UVD. У разі прискорення на GPU ситуація змінюється. Ядро два залишається навантаженим на максимум (ми не знаємо, чому так багато утиліти перекодування навантажують саме це ядро), проте навантаження ядер один, три і чотири падає до рівня нижче 50%, а навантаження на GPU зростає.

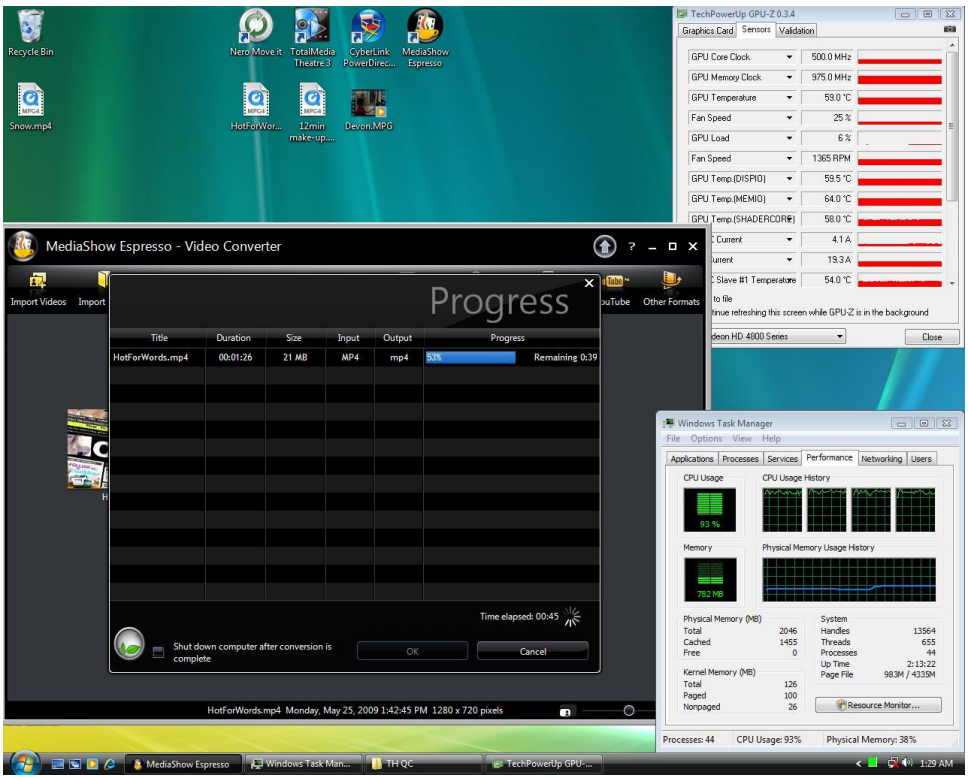

Рисунок 2.9 – Тест без прискорення на GPU

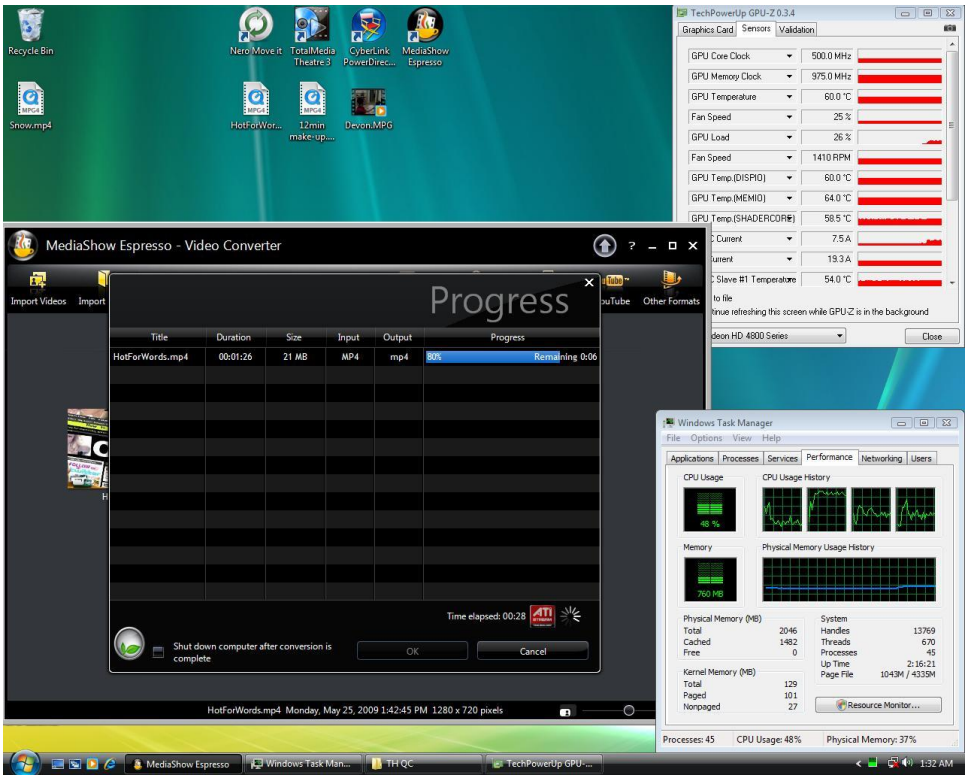

Рисунок 2.10 – Тест з активною підтримкою Stream

Порівняємо приведені вище приклади з технологією CUDA. Можна отримати деяку інформацію за результатами навантаження на CPU і фінальної продуктивності. Помітно, що на рисунку 2.11, коли використовується тільки CPU, процесор досить сильно навантажений – аналогічно до випадку з відеокартою AMD. Коли додається підтримку CUDA (рис. 2.12), то навантаження на CPU майже не змінюється.

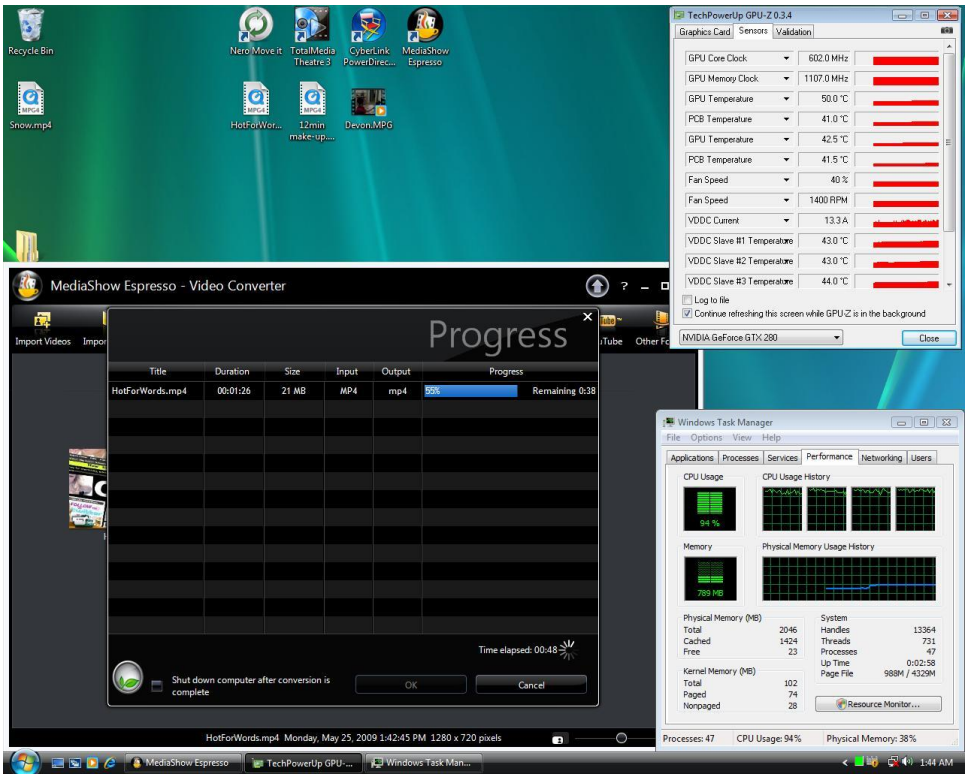

Рисунок 2.11 – Тест CPU

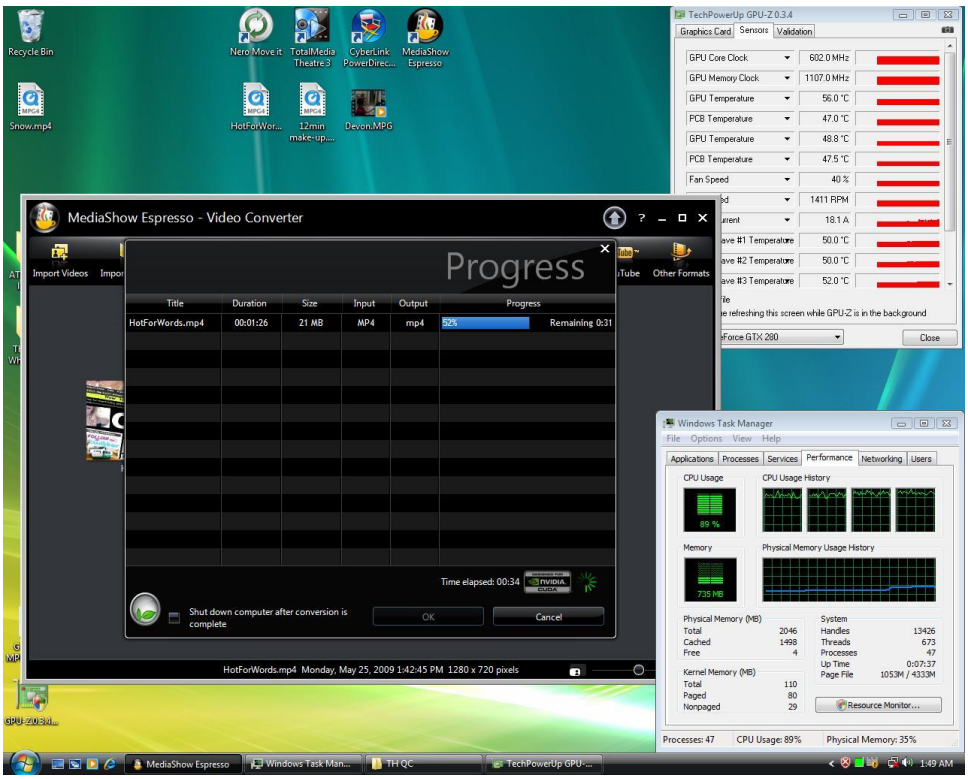

Рисунок 2.12 – Тест з CUDA

Отримаємо 35% зниження часу перекодування з увімкненим прискоренням GPU, так що CUDA дійсно допомагає. Але ось що цікаво: системи AMD і Nvidia показують однаковий час кодування в режимі, де працює тільки CPU, але підтримка Stream дає 108% приріст продуктивності, легко обганяючи перевага CUDA, і при цьому навантаження на CPU виявляється на 40% менше, ніж під CUDA. Як можна буде побачити далі, подібний результат спостерігається не скрізь. Іноді CUDA показує себе краще. Але тести, подібні наведеним вище, доводять, що концепція збалансованої платформи AMD дає реальні переваги, це не просто маркетинговий термін, щоб продавати більше процесорів.

3 ПІДГОТОВКА ДО ПРАКТИЧНОЇ ЧАСТИНИ

У цьому розділі наведено огляд базових понять спеціалізації алгоритмів, а також основних інструментів, що використовуються для проведення спеціалізації. А також провести ще одне дослідження щодо конвертування (здійснення обчислень) відео у різних умовах, щоб більш дослідити роботу GPU, CPU та GPGPU.

3.1 Спеціалізація

Спеціалізація є одним з методів агресивної програмної оптимізації. Суттю методу є автоматичне, за допомогою особливого інструменту - спеціалізатору, породження на основі вхідної процедури, а також деякої заздалегідь певної частини її вхідних параметрів нової, спеціалізованної, процедури. Дана процедура на залишку вхідних параметрів повністю еквівалентна початковій з точки зору поведінки. При цьому застосована техніка спеціалізації забезпечує їй істотно більш високу ефективність, ніж у початкової процедури, за часом виконання [22].

Одним з найбільш відомих прикладів застосування спеціалізації є розгортання зведення числа в заздалегідь відому ступінь.

3.2 Інструменти для спеціалізації на графічному процесорі

Для проведення подальших досліджень спеціалізації алгоритмів на GPU необхідно вибрати спеціалізатор, що володіє переліченими нижче властивостями:

– підтримка спеціалізації CUDAC;

– метою роботи є дослідження застосовності спеціалізації на графічних процесорах, тому від спеціалізатора потребується підтримка даної з найбільш поширених на момент проведення огляду мов для програмування паралельних обчислень на GPU CUDAC;

– швидкодія (оскільки основним призначенням спеціалізації є оптимізація часу виконання, важливо мінімізувати накладні витрати, вироблені спеціалізатором);

– простота використання.

У якості подальшого розвитку теми роботи можливо розробити прикладний програмний продукт, що використовує спеціалізацію на GPU. Тому важливо заздалегідь забезпечити простоту користування спеціалізатором для програміста.

3.3 Бібліотека LLPE

Бібліотека LLPE представляє собою спеціалізатор для біткоду LLVM, розроблений в університеті Іллініоса, США. Виходячи з архітектури інструмента та його сумісності с кодом LLVM мається теоретична можливість роботи с CUDAc [23]. Не дивлячись на наявність гарної документації, дана бібліотека не підходить для вирішення саме цих задач по перерахованим нижче причинам:

– бібліотека заявлена як нестабільна, що обмежує можливість побудови на її базі якісного програмного продукту;

– у відкритому доступі немає інформації про попередні спроби використання спеціалізатора з кодом для відеокарт, тому для досягнення поставлених цілей можуть знадобитися вагомі зусилля з модифікації LLPE.

3.4 Бібліотека AnyDSL

Бібліотека AnyDSL пропонує офлайн-спеціалізацію абстракцій в предметно-орієнтованих бібліотеках. Вона реалізована як щось між LLVM і спеціалізованою бібліотекою, є стабільною і активно підтримується, може обробляти код для GPU. Однак застосування AnyDSL для вирішення поставлених задач так саме обмежено перерахованими нижче причинами.

– AnyDSL використовує окрему мову для написанню коду для спеціалізатора – Impala, що, в сукупності з наявністю додаткових надбудов над LLVM, ускладнює використання спеціалізатору в прикладних задачах;

– наявність складної архітектури у фреймворка потенційно тягне за собою збільшення накладних витрат, а значить, і зменшення швидкодії.

3.5 Інструмент LLVM.mix

Система LLVM.mix [8], розроблена в ІСПРАН, являє собою офлайн-спеціалізатор для проміжного коду LLVM (LLVMIR). Серед її позитивних рис можна виділити відсутність необхідності в специфічній для предметної області мові: конструкції управління реалізовані як атрибути компілятора, інтерпретовані фронтендом для LLVMClang.mix. Крім того, вона реалізована, як прохід для вбудованого оптимізатора LLVM, що забезпечує відносно високу продуктивність. Також, виходячи з цієї особливості реалізації LLVM.mix можна зробити висновок про теоретичну підтримку спеціалізації на CUDA, можливість якої була попередньо підтверджена автором інструменту [24].

3.6 Створення інструменту для проведення замірів

У цьому розділі буде представлена система, використана в даній роботі для проведення замірів продуктивності алгоритмів до і після спеціалізації .

LLVM.mix не надає програмісту спеціального універсального інтерфейсу для управління своєю роботою, тому виникла потреба в інструменті, який зв’яже всю інфраструктуру LLVM.mix і бібліотеку для проведення замірів продуктивності разом, а так само надасть програмісту можливість оперативно заміняти алгоритми – об’єкти тестування.

Загальний сценарій взаємодії системи і спеціалізатора зображений на рисунку 3.1, діаграма її компонентів представлена на рисунку 3.2. Для оцінки часу виконання програм використана бібліотека Google Benchmark4 в силу її хорошою підтримки мов C / C++ і їх похідних, а так само в зв’язку з можливістю отримання статистичних даних за результатами оцінки продуктивності з використанням бібліотеки [25].

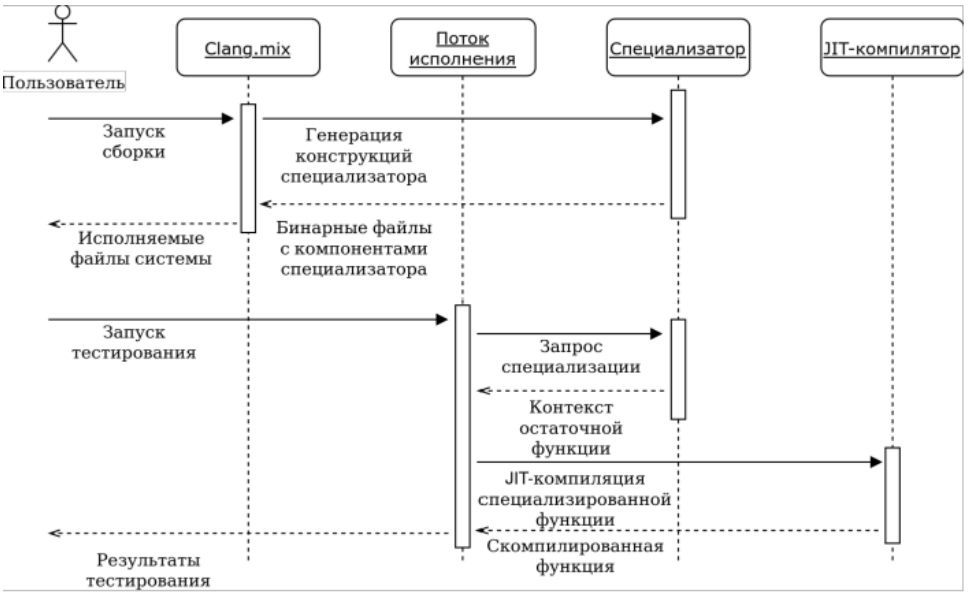

Рисунок 3.1 – Діаграма послідовності, що описує сценарій взаємодії зі спеціалізатором

Тому була створена невелика система, що складається з трьох основних модулів: ядро тестування, інтерфейс і модуль JIT-компіляції.

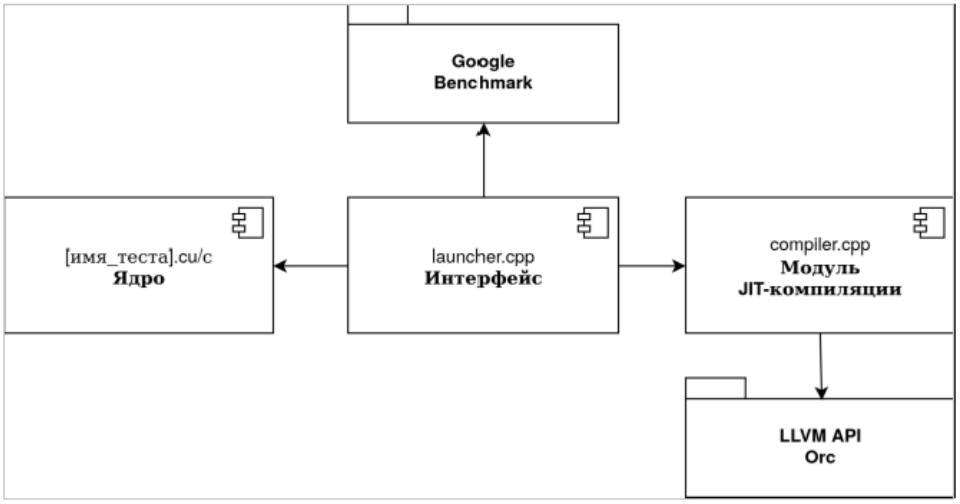

Рисунок 3.2 – Діаграма компонентів системи

Інтерфейс пов’язує всі компоненти системи разом із Google Benchmark, в код ядра перед компіляцією інструменту користувач вносить цільової алгоритм для тестування з розставленими необхідним чином конструкціями управління LLVM.mix, а модуль JIT-компіляції здійснює компіляцію спеціалізованої функції по засобом виклику внутрішніх функцій LLVM і Orc JIT API. Останнє необхідно в силу реалізації LLVM.mix як модуля для LLVM.

Таким чином, перший етап роботи спеціалізатора відбувається під час компіляції інструменту з прописаним програмістом ядром, а другий – безпосередня спеціалізація алгоритму і його запуск – вже під час виконання за допомогою модуля JIT-компіляції.

4 ПРОВЕДЕННЯ ДОСЛІДЖЕНЬ

В даному розділі буде перевірка застосовності та ефективності техніки спеціалізації для оптимізації даних алгоритмів.

4.1 Вибір алгоритмів для подальших досліджень

У відповідності з поставленими завданнями необхідно вибрати кілька алгоритмів, які, з одного боку, з точки зору теорії мають потенціал для оптимізації з використанням техніки спеціалізації і, з іншого боку, добре піддаються розпаралелюванню на графічних процесорах.

4.2 Тензорний добуток операторів з розрідженою матрицею

Даний алгоритм в рамках проведеного дослідження припускає наявність розрідженої матриці, представленої у вигляді координатного списку, тобто списку трійок з координат і значення для всіх елементів матриці з ненульовим значенням; а так само другий матриці довільного виду, причому остання матриця має малу розмірність (до десяти), і оголошується статичної. Такий алгоритм має потенціал для спеціалізації в силу наявності статичних фрагментів в реалізації алгоритму і для розпаралелювання на GPU в силу незалежності ітерацій множення елемента розрідженій матриці на статичну матрицю. Тензорний добуток використовується [26] в такий актуальній на момент проведення дослідження області як графові бази даних: велика розріджена матриця є матрицею суміжності графа, маленька щільна є запит, а їх тензорний добуток – виконання регулярного запиту до графової бази даних.

4.3 Множинне зіставлення шаблонів

Алгоритм в наївною реалізації являє собою по елементне зіставлення шуканого шаблона з кожним суфіксом рядка пошуку з квадратичною складністю. Він може бути успішно використаний як багатопоточними обчисленнями на GPU [27], так і спеціалізацією до алгоритмі Кнута-Морріса-Пратта [28]. Як приклад практичного застосування алгоритмів зіставлення шаблонів можна назвати такі актуальні на момент написання роботи області, як молекулярна біологія [29] і біоінформатика.

4.4 Зіставлення з регулярним виразом

Алгоритм перевіряє належність рядка до множини символьних послідовностей, що породжуються регулярним виразом шляхом по елементного проходу по посиланнях в відповідному висловом детермінованому кінцевому автоматі. Зіставлення з регулярним виразом теоретично може бути піддано спеціалізації і перенесений на CUDA C [30]. Як застосування алгоритму на практиці можна назвати глибоку інспекцію пакетів (DPI) [31] або обробку природної мови [32].

4.5 Згортка зображення

Алгоритм здійснює фільтрацію зображень методом згортки, тобто перетворення кожної точки зображення з урахуванням навколишніх точок. Перетворення задається ядром згортки. Згортка зображень теоретично може бути піддана спеціалізації [33], а так само ефективно перенесений на CUDA C. Даний алгоритм є одним з базових у широкій практичній області обробки зображень.

4.6 Відбір сумісних алгоритмів на CPU

Багатониткове програмування, особливо для GPU, має велику кількість особливостей і, як правило, технічно складніше, ніж однониткове програмування для CPU. Тому, для скорочення кількості зайвих дій при проведенні дослідження спеціалізації коду для графічних процесорів і підвищення точності і змістовності дослідження дуже важливо було відібрати алгоритми, які показують приріст продуктивності і при роботі з центральним процесором. Роботи проводилися на стенді Лабораторії мовних інструментів Jetbrains Research, що має наступну конфігурацію: Ubuntu18.04, Intel®Core ™ i7-6700 (4x4GHz), 64GB RAM, NvidiaGeForceGTX1070. Для замірів використовувалася описана раніше інфраструктура.

Динамічні дані в кожному з тестів генерувалися псевдовипадковим чином з використанням рівномірного розподілу і мали обсяг близько 400 мегабайт (з урахуванням відмінностей розмірів типів даних). Статичні дані, на які проводилася спеціалізація, відносно невеликого розміру вибиралися в кожному тесті окремо, в залежності від завдання. Кожен тест виконувався від десяти до декількох сотень разів відповідно до алгоритмів Google Benchmark, що забезпечує істотне підвищення точності даних при псевдовипадкової ситуації генерації тестів.

5 ТЕСТУВАННЯ АЛГОРИТМІВ

5.1 Тензорний добуток

У даному випадку в якості динамічних даних виступає список трійок - пара координат і значення – для кожного ненульового елемента розрідженій матриці. Розмір розрідженій матриці варіювався від 25 000 рядків до 2 000 000 рядків с заповненням десяти ненульових елементів на рядок. Заповнення статичної матриці у разі її невироджені не впливає помітним чином на кількість ітерацій в силу реалізації алгоритму, тому вибиралася в стандартному Але вигляді розміром десять з псевдовипадкових елементів з рівномірним розподілом.

Тестування показало прискорення спеціалізованого алгоритму тензорного добутку твори близько 18% с відхиленням менше 0,001% на CPU, як показано на рисунку 5.1, а так само сумісність його з LLVM.mix.

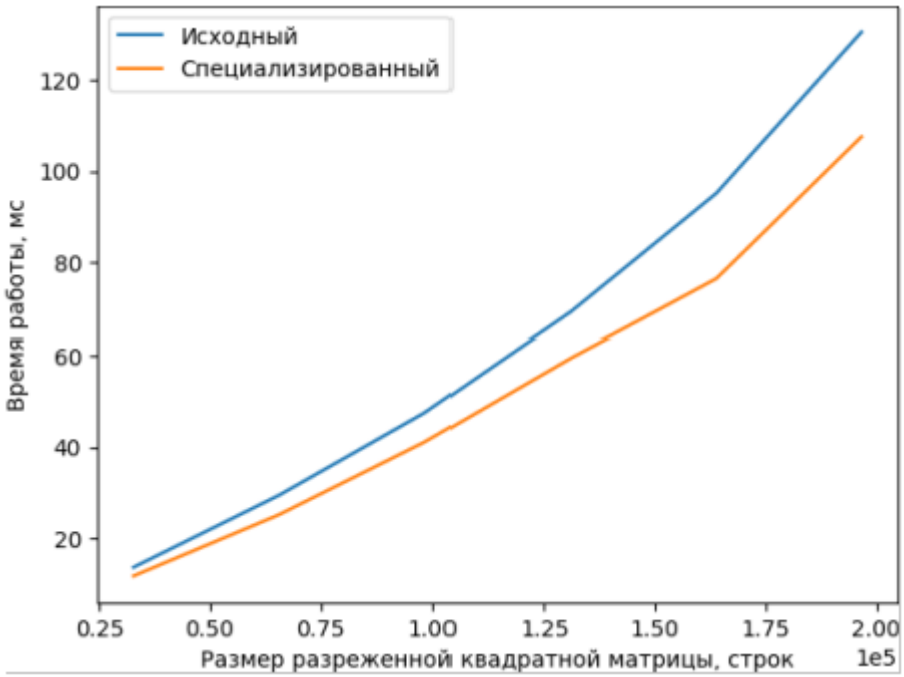

Рисунок 5.1 – Результати тестування алгоритму тензорного добутку

5.2 Згортка зображень

Динамічними даними при роботі алгоритму згортки зображень виступають квадратні матриці з невід'ємних цілих чисел розміром від 1000 до 7500 рядків. Статичними даними виступали квадратні матриці-ядра розміром п'ять, мають невироджених конфігурацію (нулі не становлять більшості елементів): медіанний фільтр, а також інші отримані вручну з елементами з проміжку від мінус п'яти до п'яти.

Тестування показало прискорення спеціалізованого алгоритму згортки зображень близько 20% з відхиленням менше 0.001% на центральному процесорі, як показано на рисунку 5.2, тобто його сумісність з обраним спеціалізатором.

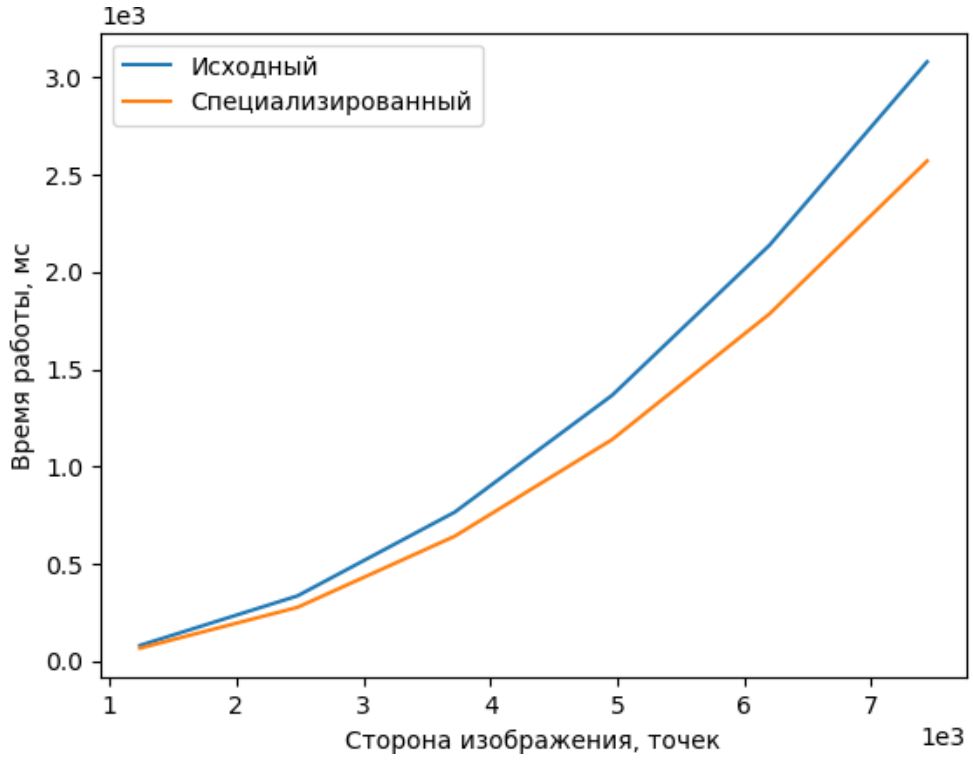

Рисунок 5.2 – Результати тестування алгоритму згортки зображення

5.3 Зіставлення шаблонів із регулярним виразом

В якості динамічних даних в обох тестах виступали рядки довжиною від 5 до 35 мільйонів символів. Статичними даними, на які проводилася спеціалізація, виступили або рядки довжиною 200 символів в першому випадку, або детерміновані кінцеві автомати з десятьма станами в другому, що генеруються псевдовипадково з рівномірним розподілом в обох випадках. І в завданні зіставлення шаблонів, і в завданні зіставлення з регулярним виразом тестування показало негативні результати: уповільнення від двох до трьох разів у порівнянні з вихідними версіями, як показано на рисунку 5.3 і рисунку 5.4. Алгоритми погано спеціалізувалися через наявність великої кількості динамічних конструкцій (умовні переходи, звернення до динамічної частини параметрів та інше), а уповільнення пояснюється суттєвими накладними витратами, здійснюваними спеціалізатором.

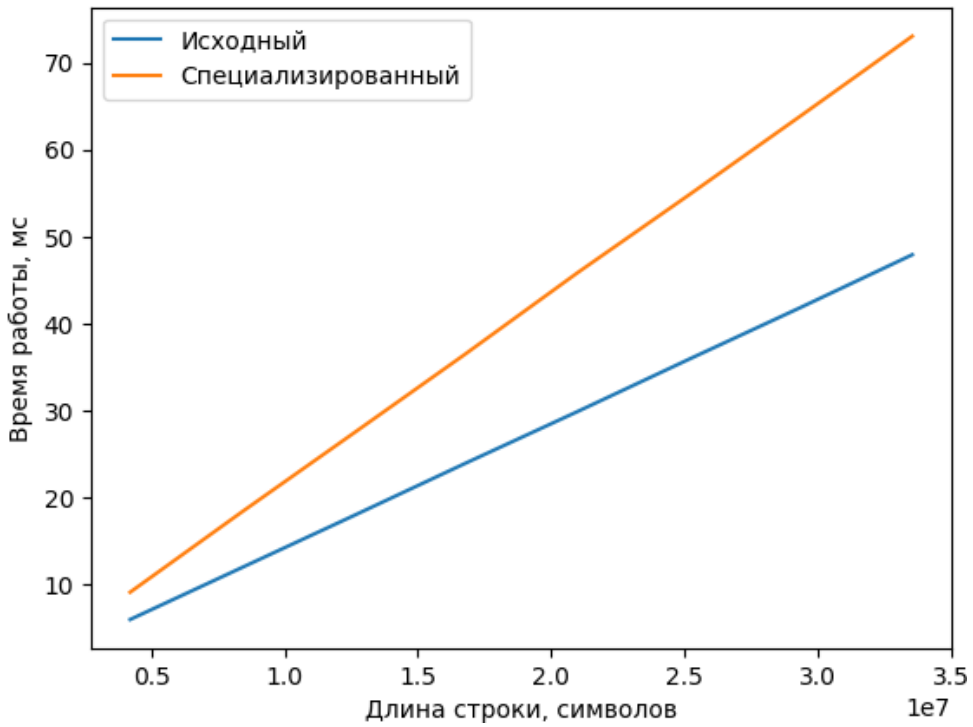

Рисунок 5.3 – Результати тестування алгоритму зіставлення шаблонів

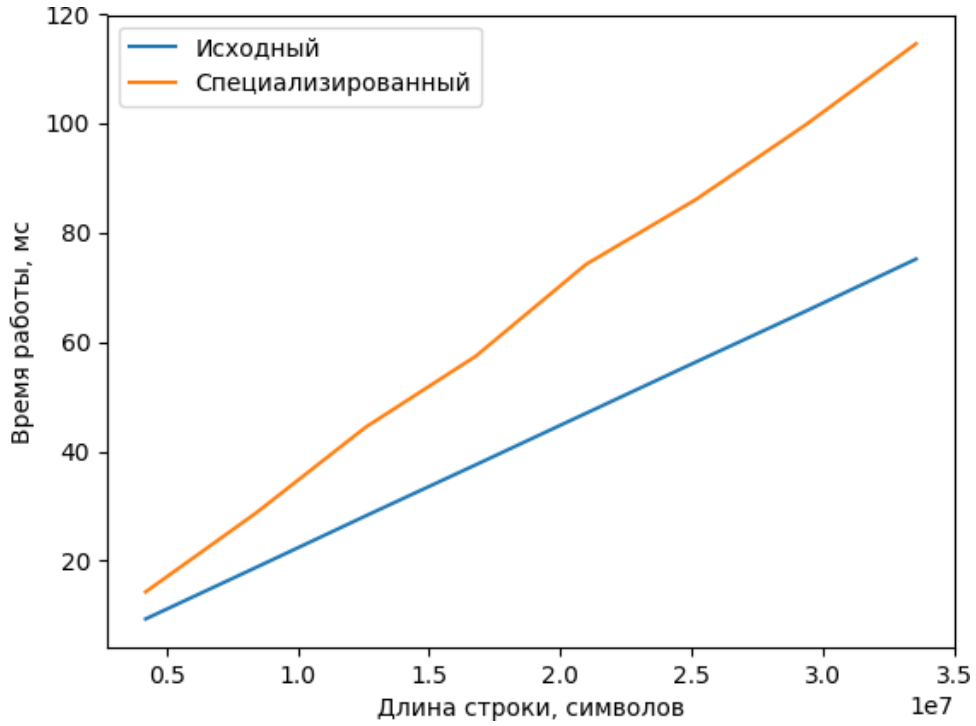

Рисунок 5.4 – Результати тестування алгоритму зіставлення шаблонів

5.4 Результати тестування

За результатами тестування було виявлено два зазначених нижче алгоритму, добре піддаються спеціалізації на CPU, а значить потенційно придатних для подальших експериментів по спеціалізації на GPU:

– тензорне твори (прискорення до 18 ± 0,001%);
– згортка зображень (прискорення до 20 ± 0,001%).

5.5 Експерименти на графічному процесорі

Відповідно до списку завдань, необхідно було провести експериментальне дослідження спеціалізації обраних алгоритмів (тензорний добуток з розрідженою матрицею і згортком зображення) на графічному процесорі. Їх адаптація для GPU проводилася шляхом переписування на мову

CUDA C з використанням універсального типу пам’яті графічного процесора.

У процесі постановки експерименту були виявлені критичні проблеми, що стосуються роботи спеціалізатора LLVM.mix з кодом на CUDA C. Їх суть описана нижче.

1. Виявлено складність при передачі ядра з атрибутами CUDA і LLVM.mix в інтерфейс розробленого інструменту для тестування. Для запуску процесу спеціалізації потрібно передати в LLVM.mix по засобом вбудованих функцій LLVM і Orc, обгорнутих в високорівнева інтерфейс в модулі JIT-компілятора розробленого інструменту, покажчик на контекст спеціалізованої функції-об’єкта службового класу LLVM. Складність полягає в необхідності при передачі покажчика на функцію з модуля ядра включити в код на C ++ код на CUDAC з характерними атрибутами, що викликає помилку компіляції. В якості вирішення можна використовувати проксі-об’єкт, що передає покажчик на функцію між об’єктними файлами на різних мовах, що потенційно може призвести до втрати керуючих атрибутів спеціалізатора. Так ж можливо повне переписування інструменту для тестування на низькорівневий проміжний IR-код LLVM, що повністю вирішує проблему , але значно підвищує трудомісткість роботи.

2. При виконанні коду на CUDA разом з LLVM.mix з використанням проксі-об’єктів або дописування керуючих конструкцій спеціалізатора в проміжний код LLVM на будь-якому випробуваному коректному тесті виникає помилка часу виконання Segmentation fault, викликана некоректною роботою JIT-компілятора LLVM на коді CUDA. Єдиним адекватним вирішенням даної проблеми була визнана істотна модифікація JIT-компілятора LLVM.

Схематично проблемні місця відображені на рисунку 5.5. Жирною жовтою стрілкою на даній схемі зображено першу проблему зі списку, пов’язану з труднощами при передачі покажчика функції між CUDAC і C++,

а жирною перекресленою – друга, критична, проблема, пов’язана з некоректною JIT-компіляції коду для CUDA.

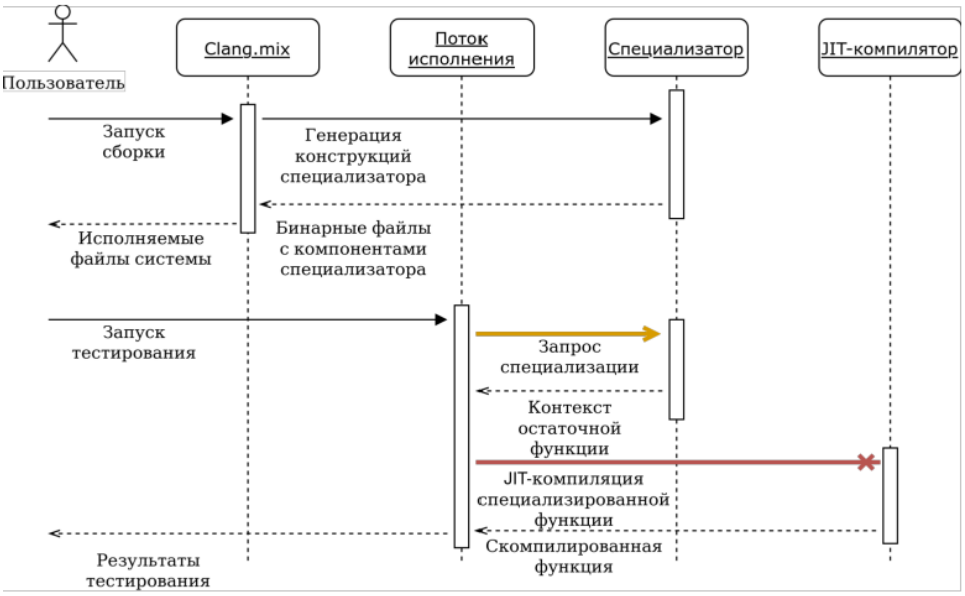

Рисунок 5.5 – Діаграма послідовності із зазначенням проблемних місць

Таким чином, в результаті проведеного дослідження було зроблено висновок про непридатність обраного інструменту LLVM.mix для спеціалізації коду на CUDAC без проведення радикальних правок в спеціалізаторі, що виходить за рамки дипломної роботи.

В якості вирішення зазначених проблем в подальшому можливе використання інструменту Clang.JIT [34], який був випущений на початку 2020 року. Ця система, виходячи з документації, не володіє виявленими недоліками, при цьому пропонує коректну JIT-компіляцію коду CUDA. Поява Clang.JIT додатково підкреслює динамічний розвиток досліджуваної предметної області, а також актуальність роботи.

5.6 NVIDIA CUDA та AMD APP

Коли почали щось робити перші спроби реалізувати неграфічні обчислення на GPU (General Purpose GPU, GPGPU), виник компілятор BrookGPU. До його створення розробникам доводилося отримувати доступ до ресурсів відеокарти через графічні API OpenGL або Direct3D, що значно ускладнювало процес програмування, так як вимагало специфічних знань – доводилося вивчати принципи роботи з 3D-об'єктами (шейдерами, текстурами і т.п.). Це стало причиною досить обмеженого застосування GPGPU в програмних продуктах. BrookGPU став своєрідним «перекладачем». Ці потокові розширення до мови Сі приховували від програмістів тривимірний API і при його використанні потреба в знаннях 3D-програмування практично відпала. Обчислювальні потужності відеокарт стали доступні програмістам у вигляді додаткового співпроцесора для паралельних розрахунків. Компілятор BrookGPU обробляв файл з кодом Сі і розширеннями, вибудовуючи код, прив'язаний до бібліотеки з підтримкою DirectX або OpenGL.

Багато в чому завдяки BrookGPU, компанії NVIDIA і ATI (нині AMD) звернули увагу на зароджується технологію обчислень загального призначення на графічних процесорах і почали розробку власних реалізацій, що забезпечують прямий і більш прозорий доступ до обчислювальних блокам 3D-прискорювачів.

В результаті компанія NVIDIA розробила програмно-апаратну архітектуру паралельних обчислень CUDA (Compute Unified Device Architecture). Архітектура CUDA дозволяє реалізувати неграфічні обчислення на графічних процесорах NVIDIA.

Реліз публічної бета-версії CUDA SDK відбувся в лютому 2007 року. В основі API CUDA лежить спрощений діалект мови Сі. Архітектура CUDA SDK забезпечує програмістам реалізацію алгоритмів, здійсненних на графічних процесорах NVIDIA, і включення спеціальних функцій в текст

програми на Сі. Для успішної трансляції коду на цій мові до складу CUDA SDK входить власний Сікомпілятор командного рядка nvcc компанії NVIDIA.

Компанія AMD (ATI) також розробила свою версію технології GPGPU, яка раніше називалася ATI Stream, а тепер – AMD Accelerated Parallel Processing (APP). Основу AMD APP становить відкритий індустріальний стандарт OpenCL (Open Computing Language). Стандарт OpenCL забезпечує паралелізм на рівні інструкцій і на рівні даних і є реалізацією техніки GPGPU. Це повністю відкритий стандарт, його використання не обкладається ліцензійними відрахуваннями. Відзначимо, що AMD APP і NVIDIA CUDA несумісні один з одним, проте, остання версія NVIDIA CUDA підтримує і OpenCL.

Отже, для реалізації GPGPU на графічних процесорах NVIDIA призначена технологія CUDA, а на графічних процесорах AMD – API APP. Як уже зазначалося, використання неграфічних обчислень на GPU доцільно тільки в тому випадку, якщо можна вирішити завдання може бути распараллелена на безліч потоків. Однак більшість призначених для користувача додатків не задовольняють цим критерієм. Втім, є і деякі виключення. Наприклад, більшість сучасних відеоконвертер підтримують можливість використання обчислень на графічних процесорах NVIDIA і AMD.

Для того щоб з’ясувати, наскільки ефективно використовуються обчислення на GPU в призначених для користувача відеоконвертер, відібрано три популярних рішення: Xilisoft Video Converter Ultimate 7.7.2, Wondershare Video Converter Ultimate 6.0.3.2 і Movavi Video Converter 10.2.1. Ці конвертери підтримують можливість використання графічних процесорів NVIDIA і AMD, причому в налаштуваннях відеоконвертер можна відключити цю можливість, що дозволяє оцінити ефективність застосування GPU. Для відеоконвертірованія застосовувано три різних відеоролика. Перший відеоролик мав тривалість три хвилини та тридцять п’ять секунд і

розмір 1,05 Гбайт. Він був записаний в форматі зберігання даних (контейнер) mkv і мав такі характеристики:

а) відео:

- формат – MPEG4 Video (H264);
- дозвіл – 1920 x 1080;
- режим бітрейта – Variable;
- середній відеобітрейт – 42,1 Мбіт / с;
- максимальний відеобітрейт – 59,1 Мбіт / с;
- частота кадрів – 25 fps;

б) аудіо:

- формат – MPEG-1 Audio;
- аудіобітрейт – 128 Кбіт / с;
- кількість каналів – 2;
- частота семплірованія - 44,1 кГц.

Другий відеоролик мав тривалість чотири хвилини та двадцять п’ять секунд і розмір 1,98 Гбайт. Він був записаний в форматі зберігання даних (контейнер) MPG і мав такі характеристики:

а) відео:

- формат – MPEG-PS (MPEG2 Video);
- дозвіл – 1920 * um * 1080;
- режим бітрейта – Variable;
- середній відеобітрейт – 62,5 Мбіт / с;
- максимальний відеобітрейт – 100 Мбіт / с;
- частота кадрів – 25 fps;

б) аудіо:

- формат – MPEG-1 Audio;
- аудіобітрейт – 384 Кбіт / с;
- кількість каналів – 2;
- частота семплірованія – 48 кГц.

Третій відеоролик мав тривалість три хвилини сорок сім секунд і розмір 197 Мбайт. Він був записаний в форматі зберігання даних (контейнер) MOV і мав такі характеристики:

а) відео:
- формат – MPEG4 Video (H264);
- дозвіл – 1920 * um * 1080;
- режим бітрейта – Variable;
- відеобітрейт – 7024 Кбіт / с;
- частота кадрів – 25 fps;

б) аудіо:
- формат – AAC;
- аудіобітрейт – 256 Кбіт / с;
- кількість каналів – 2;
- частота семплірованія – 48 кГц.

Всі три тестових відеоролика конвертувалися з використанням відеоконвертер в формат зберігання даних MP4 (кодек H.264) для перегляду на планшеті iPad 2. Дозвіл вихідного відеофайлу становило 1280 * 720. Абсолютно однакові настройки конвертації у всіх трьох конвертерах використовувати не коректно. Так, в відеоконвертер для конвертації застосовувався пресет iPad 2 – H.264 HD Video. У цьому пресеті використовуються наступні настройки кодування:
- кодек – MPEG4 (H.264);
- дозвіл – 1280 * um * 720;
- частота кадрів – 29,97 fps;
- відеобітрейт – 5210 Кбіт / с;
- аудіокодек – AAC;
- аудіобітрейт – 128 Кбіт / с;
- кількість каналів – 2;
- частота семплірованія – 48 кГц.

У відеоконвертер Wondershare Video Converter Ultimate 6.0.3.2 використовувався пресет iPad 2 з наступними додатковими опціями:

- кодек – MPEG4 (H.264);
- дозвіл – 1280 * um * 720;
- частота кадрів – 30 fps;
- відеобітрейт – 5000 Кбіт / с;
- аудіокодек – AAC;
- аудіобітрейт – 128 Кбіт / с;
- кількість каналів – 2;
- частота семплірованія – 48 кГц.

У конвертері Movavi Video Converter 10.2.1 застосовувався пресет iPad (1280 * 720, H.264) (* .mp4) з наступними додатковими опціями:

- формат відео – H.264;
- дозвіл – 1280 * um * 720;
- частота кадрів – 30 fps;
- відеобітрейт – 2500 Кбіт / с;
- аудіокодек – AAC;
- аудіобітрейт – 128 Кбіт / с;
- кількість каналів – 2;
- частота семплірованія – 44,1 кГц.

Конвертація кожного вихідного відеоролика проводилося по п’ять разів на кожному з відеоконверторів, причому з використанням як графічного процесора, так і тільки CPU. Після кожного конвертації комп’ютер перезавантажувався.

Кожен відеоролик конвертувався десять разів в кожному відеоконверторі. Для автоматизації цієї рутинної роботи була написана спеціальна утиліта з графічним інтерфейсом, що дозволяє повністю автоматизувати процес тестування.

5.7 Конфігурація стенду для тестування

Стенд для тестування мав наступну конфігурацію:

- процесор – Intel Core i7-3770K;
- материнська плата – Gigabyte GA-Z77X-UD5H;
- чіпсет системної плати – Intel Z77 Express;
- пам’ять – DDR3-1600;
- обсяг пам’яті – 8 Гбайт (два модуля GEIL по 4 Гбайт);
- режим роботи пам’яті – двоканальний;
- відеокарта – NVIDIA GeForce GTX 660Ti (відеодрайвер 314.07);
- накопичувач – Intel SSD 520 (240 Гбайт).

На стенді встановлювалася операційна система Windows 7 Ultimate (64-bit). Спочатку було проведено тестування в штатному режимі роботи процесора і всіх інших компонентів системи. При цьому процесор Intel Core i7-3770K працював на штатній частоті 3,5 ГГц с активованим режимом Turbo Boost (максимальна частота процесора в режимі Turbo Boost становить 3,9 ГГц). Потім повторювався процес тестування, але при розгоні процесора до фіксованої частоти 4,5 ГГц (без використання режиму Turbo Boost). Це дозволило виявити залежність швидкості конвертації від частоти процесора (CPU). На наступному етапі тестування було прийнято рішення повернутися до штатних налаштувань процесора і повторти тестування вже з іншими відеокартами:

- NVIDIA GeForce GTX 280 (драйвер 314.07);
- NVIDIA GeForce GTX 460 (драйвер 314.07);
- AMD Radeon HD6850 (драйвер 13.1).

Таким чином, відеоконвертіроване проводилося на чотирьох відкритих різної архітектури. Старша відеокарта NVIDIA GeForce 660Ti заснована на однойменному графічному процесорі з кодовим позначенням GK104 (архітектура Kepler), виробленому за 28-нм техпроцесу. Цей графічний

процесор містить 3,54 млрд. транзисторів, а площа кристала становить 294 мм.

Нагадаю, що графічний процесор GK104 включає чотири кластери графічної обробки (Graphics Processing Clusters, GPC). Кластери GPC є незалежними пристроями в складі процесора і здатні працювати як окремі пристрої, оскільки володіють усіма необхідними ресурсами: растеризатор, геометричними двигунами і текстурними модулями.

Кожен такий кластер має два потокових мультипроцессора SMX (Streaming Multiprocessor), але в процесорі GK104 в одному з кластерів один мультипроцессор заблокований, тому всього є сім мультипроцессоров SMX.

Кожен потоковий мультипроцессор SMX містить 192 потокових обчислювальних ядра (ядра CUDA), тому в сукупності процесор GK104 налічує 1344 обчислювальних ядра CUDA. Крім того, кожен SMX-мультипроцессор містить 16 текстурних модулів (TMU), 32 блоки спеціальних функцій (Special Function Units, SFU), 32 блоки завантаження і зберігання (Load-Store Unit, LSU), движок PolyMorph і багато іншого.

Відеокарта GeForce GTX 460 заснована на графічному процесорі з кодовим позначенням GF104 на базі архітектури Fermi. Цей процесор виробляється по 40-нм техпроцесу і містить близько 1,95 млрд транзисторів.

Графічний процесор GF104 включає два кластери графічної обробки GPC. Кожен з них має чотири потокових мультипроцессора SM, але в процесорі GF104 в одному з кластерів один мультипроцессор заблокований, тому існує всього сім мультипроцессоров SM.

Кожен потоковий мультипроцессор SM містить 48 потокових обчислювальних ядра (ядра CUDA), тому в сукупності процесор GK104 налічує 336 обчислювальних ядра CUDA. Крім того, кожен SM-мультипроцессор містить вісім текстурних модулів (TMU), вісім блоків спеціальних функцій (Special Function Units, SFU), 16 блоків завантаження і зберігання (Load-Store Unit, LSU), движок PolyMorph і багато іншого.

Графічний процесор GeForce GTX 280 відноситься до другого покоління уніфікованої архітектури графічних процесорів NVIDIA і за своєю архітектурою сильно відрізняється від архітектури Fermi і Kepler.

Графічний процесор GeForce GTX 280 складається з кластерів обробки текстур (Texture Processing Clusters, TPC), які, хоч і схожі, але в той же час сильно відрізняються від кластерів графічної обробки GPC в архітектурі Fermi і Kepler. Всього таких кластерів в процесорі GeForce GTX 280 налічується десять. Кожен TPC-кластер включає три потокових мультипроцессора SM і вісім блоків текстурної вибірки і фільтрації (TMU). Кожен мультипроцессор складається з восьми потокових процесорів (SP). Мультіпроцесори також містять блоки вибірки і фільтрації текстур даних, використовуваних як в графічних, так і в деяких розрахункових задачах.

Таким чином, в одному TPC-кластері – 24 потокових процесора, а в графічному процесорі GeForce GTX 280 їх вже 240.

Графічний процесор AMD Radeon HD6850, що має кодове найменування Barts, виготовляється по 40-нм техпроцесу і містить 1,7 млрд транзисторів.

Архітектура процесора AMD Radeon HD6850 є уніфіковану архітектуру з масивом загальних процесорів для потокової обробки численних видів даних.

Процесор AMD Radeon HD6850 складається з 12 SIMD-ядер, кожне з яких містить по 16 блоків суперскалярні потокових процесорів і чотири текстурних блоки. Кожен суперскалярний потоковий процесор містить п’ять універсальних потокових процесорів. Таким чином, всього в графічному процесорі AMD Radeon HD6850 налічується 960 універсальних потокових процесорів.

Частота графічного процесора відеокарти AMD Radeon HD6850 становить 775 МГц, а ефективна частота пам’яті GDDR5 – 4000 МГц. При цьому обсяг пам’яті становить 1024 Мбайт.

5.8 Результати тестування

Починаючи з першого тесту, коли використовується відеокарта NVIDIA GeForce GTX 660Ti і штатний режим роботи процесора Intel Core i7-3770K.

На рис. 5.6 – рис. 5.8 показані результати конвертації трьох тестових відеороликів трьома конвертерами в режимах із застосуванням графічного процесора і без.

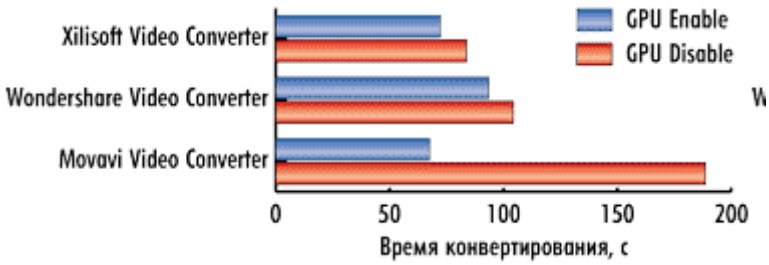

Рисунок 5.6 – Результати конвертації першого тестового відеоролика в штатному режимі роботи процесора

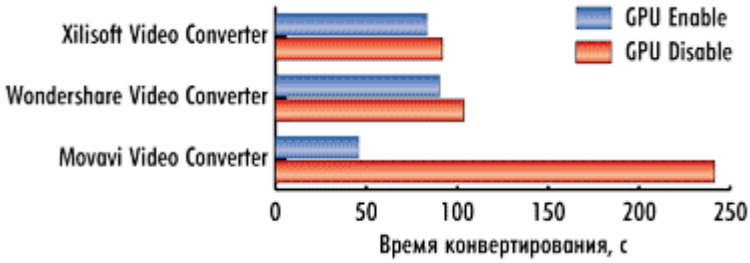

Рисунок 5.7 – Результати конвертації другого тестового відеоролика в штатному режимі роботи процесора

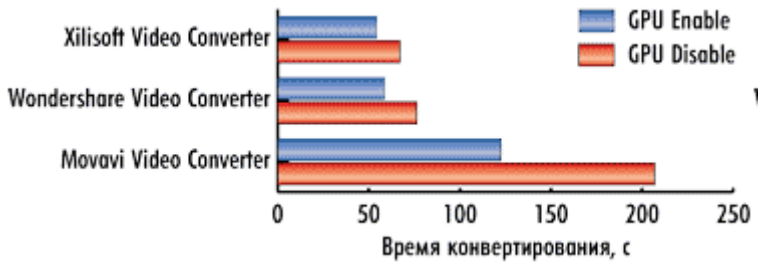

Рисунок 5.8 – Результати конвертації третього тестового відеороліка в штатному режимі роботи процесора

Як видно за результатами тестування, ефект від використання графічного процесора в наявності.

Для відеоконвертера Xilisoft Video Converter Ultimate 7.7.2 у разі застосування графічного процесора час конвертації скорочується на 14%, 9% і 19% для першого, другого і третього відеоролика відповідно.

Для відеоконвертера Wondershare Video Converter Ultimate 6.0.32 використання графічного процесора дозволяє скоротити час конвертації на 10%, 13% і 23% для першого, другого і третього відеоролика відповідно.

Але найбільше від застосування графічного процесора виграє конвертер Movavi Video Converter 10.2.1. Для першого, другого і третього відеоролика скорочення часу конвертації складає 64%, 81% і 41% відповідно.

Зрозуміло, що виграш від використання графічного процесора залежить і від вихідного відеоролика, і від налаштувань відеоконвертірованія, що, власне, і демонструють отримані нами результати.

Тепер подивимося, яким буде виграш за часом конвертації при розгоні процесора Intel Core i7-3770K до частоти 4,5 ГГц. Якщо вважати, що в штатному режимі все ядра процесора при конвертації завантажені і в режимі Turbo Boost працюють на частоті 3,7 ГГц, то збільшення частоти до 4,5 ГГц відповідає розгону по частоті на 22%.

На рис. 5.9 – рис. 5.11 показані результати конвертації трьох тестових відеороликів при розгоні процесора в режимах з використанням графічного процесора і без. В даному випадку застосування графічного процесора дозволяє отримати виграш за часом конвертації.

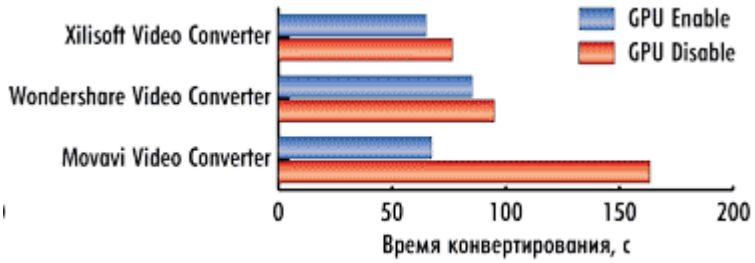

Рисунок 5.9 – Результати конвертації першого тестового відеоролика в режимі розгону процесора

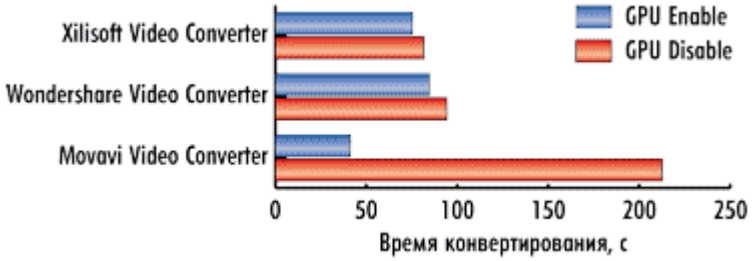

Рисунок 5.10 – Результати конвертації другого тестового відеоролика в режимі розгону процесора

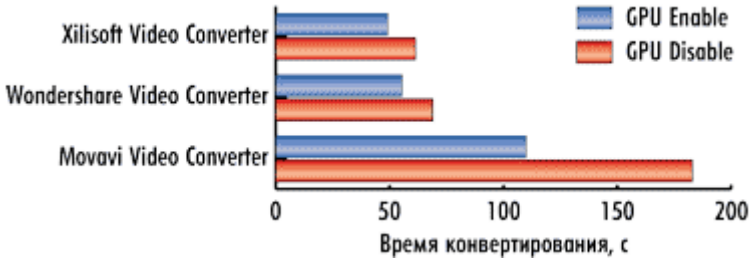

Рисунок 5.11 – Результати конвертації другого тестового відеоролика в режимі розгону процесора

Для відеоконвертера Xilisoft Video Converter Ultimate 7.7.2 у разі застосування графічного процесора час конвертації скорочується на 15%, 9% і 20% для першого, другого і третього відеоролика відповідно.

Для відеоконвертера Wondershare Video Converter Ultimate 6.0.32 використання графічного процесора дозволяє скоротити час конвертації на 10%, 10% і 20% для першого, другого і третього відеоролика відповідно.

Для конвертера Movavi Video Converter 10.2.1 застосування графічного процесора дозволяє скоротити час конвертації на 59%, 81% і 40% відповідно.

Цікаво подивитися, як розгін процесора дозволяє зменшити час конвертації при використанні графічного процесора і без нього.

На рис. 5.12 – рис. 5.14 представлені результати порівняння часу конвертації відеороликів без використання графічного процесора в штатному режимі роботи процесора і в режимі розгону. Оскільки в даному випадку конвертування проводиться тільки засобами CPU без обчислень на GPU, очевидно, що збільшення тактової частоти роботи процесора призводить до скорочення часу конвертації (збільшення швидкості конвертації). Настільки ж очевидно, що скорочення швидкості конвертації повинно бути приблизно однаково для всіх тестових відеороликів. Так, для відеоконвертера Xilisoft Video Converter Ultimate 7.7.2 при розгоні процесора час конвертації скорочується на 9%, 11% і 9% для першого, другого і третього відеоролика відповідно. Для відеоконвертера Wondershare Video Converter Ultimate 6.0.32 час конвертації скорочується на 9%, 9% і 10% для першого, другого і третього відеоролика відповідно. Ну а для відеоконвертера Movavi Video Converter 10.2.1 час конвертації скорочується на 13%, 12% і 12% відповідно.

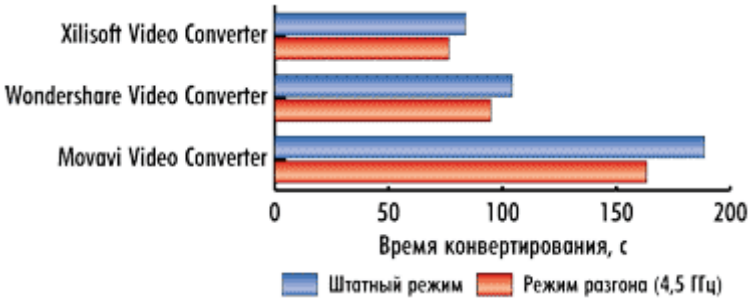

Рисунок 5.12 – Результати порівняння часу конвертації першого відеоролика без використання графічного процесора в штатному

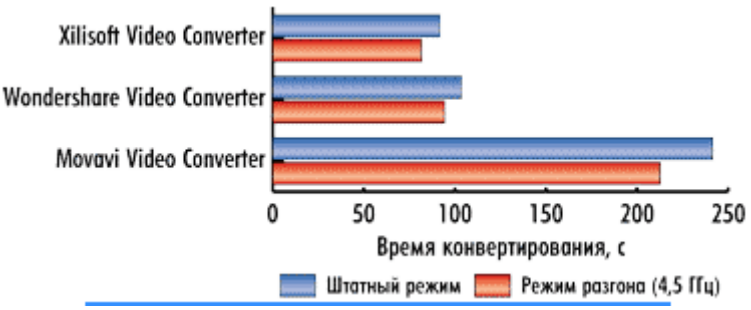

Рисунок 5.13 – Результати порівняння часу конвертації другого відеоролика без використання графічного процесора в штатному

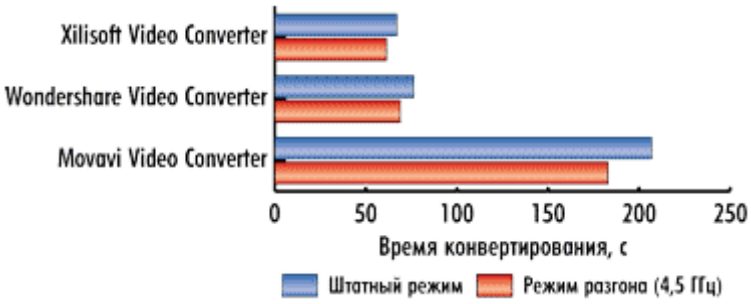

Рисунок 5.14 – Результати порівняння часу конвертації третього відеоролика без використання графічного процесора в штатному режимі роботи процесора і в режимі розгону

Таким чином, при розгоні процесора по частоті на 20% час конвертації скорочується приблизно на 10%.

Порівняємо час конвертації з використанням графічного процесора в штатному режимі роботи і в режимі розгону (рис. 5.15 – рис. 5.17).

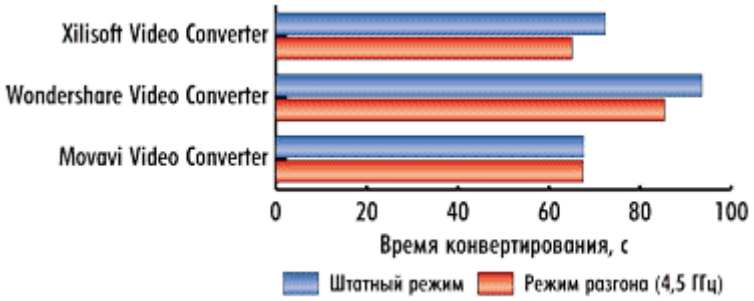

Рисунок 5.15 – Результати порівняння часу конвертації першого відеоролика з використанням графічного процесора в штатному режимі і в режимі розгону

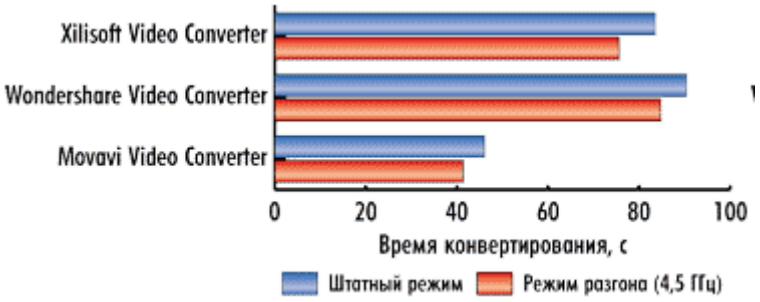

Рисунок 5.16 – Результати порівняння часу конвертації другого відеоролика з використанням графічного процесора в штатному режимі і в режимі розгону

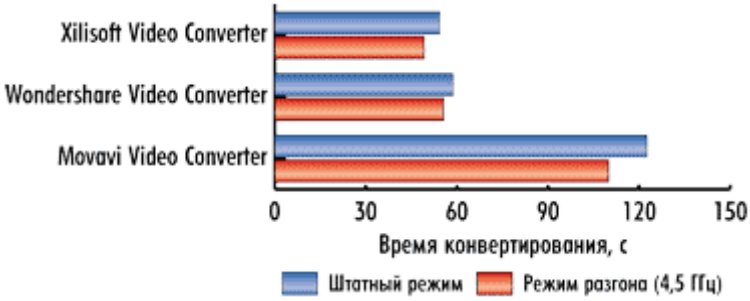

Рисунок 5.17 – Результати порівняння часу конвертації третього відеоролика з використанням графічного процесора в штатному режимі роботи процесора і в режимі розгону

Для відеоконвертера Xilisoft Video Converter Ultimate 7.7.2 при розгоні процесора час конвертації скорочується на 10%, 10% і 9% для першого,

другого і третього відеоролика відповідно. Для відеоконвертера Wondershare Video Converter Ultimate 6.0.32 час конвертації скорочується на 9%, 6% і 5% для першого, другого і третього відеоролика відповідно. Ну а для відеоконвертера Movavi Video Converter 10.2.1 час конвертації скорочується на 0,2%, 10% і 10% відповідно.

Як бачимо, для конвертерів Xilisoft Video Converter Ultimate 7.7.2 і Wondershare Video Converter Ultimate 6.0.32 скорочення часу конвертації при розгоні процесора приблизно однаково як при використанні графічного процесора, так і без його застосування, що логічно, оскільки ці конвертери не дуже ефективно використовують обчислення на GPU. А ось для конвертера Movavi Video Converter 10.2.1, який ефективно використовує обчислення на GPU, розгін процесора в режимі використання обчислень на GPU мало позначається на скороченні часу конвертації, що також зрозуміло, оскільки в даному випадку основне навантаження лягає на графічний процесор.

Тепер подивимося результати тестування з різними відеокартами.

Здавалося б, чим потужніший відеокарта і чим більше в графічному процесорі ядер CUDA (або універсальних потокових процесорів для відеокарт AMD), тим ефективніше повинно бути відеоконвертіроване в разі застосування графічного процесора. Але на практиці виходить не зовсім так.

Що стосується відеокарт на графічних процесорах NVIDIA, то ситуація наступна. При використанні конвертерів Xilisoft Video Converter Ultimate 7.7.2 і Wondershare Video Converter Ultimate 6.0.32 час конвертації практично ніяк не залежить від типу використовуваної відеокарти. Тобто для відеокарт NVIDIA GeForce GTX 660Ti, NVIDIA GeForce GTX 460 і NVIDIA GeForce GTX 280 в режимі використання обчислень на GPU час конвертації виходить один і той же (рис. 5.18 – рис. 5.20).

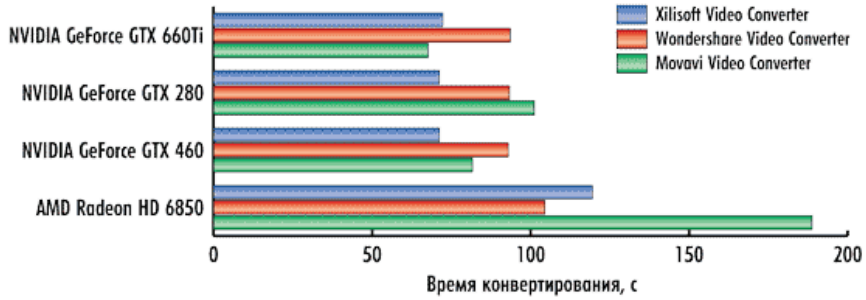

Рисунок 5.18 – Результати порівняння часу конвертації першого відеоролика на різних відкритих в режимі використання графічного процесора

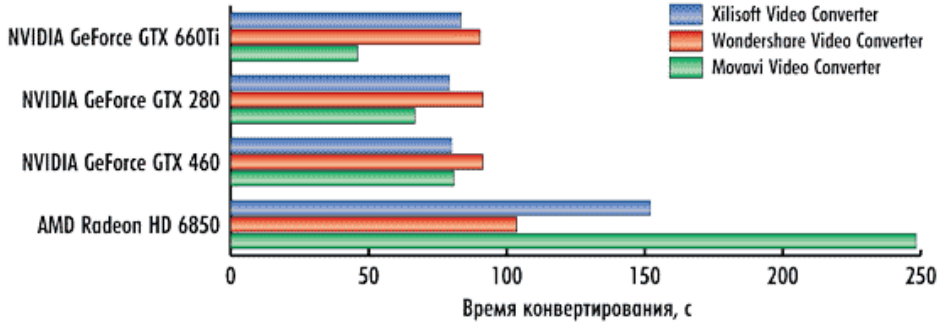

Рисунок 5.19 – Результати порівняння часу конвертації другого відеоролика на різних відкритих в режимі використання графічного процесора

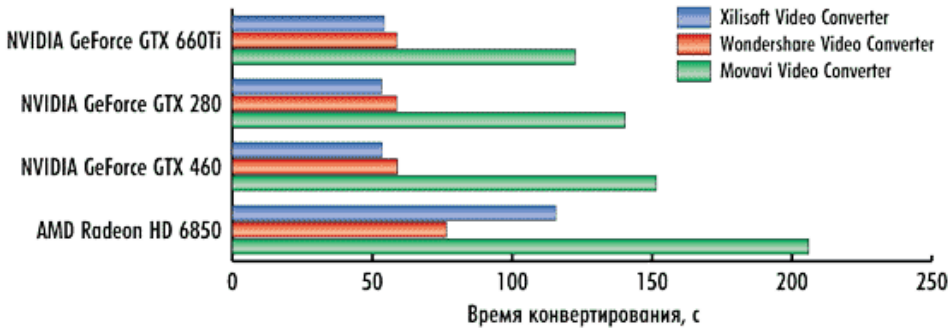

Рисунок 5.20 – Результати порівняння часу конвертації третього відеоролика на різних відкритих в режимі використання графічного процесора

Пояснити це можна лише тим, що алгоритм обчислень на графічному процесорі, реалізований в конвертерах Xilisoft Video Converter Ultimate 7.7.2 і Wondershare Video Converter Ultimate 6.0.32, просто неефективний і не дозволяє активно задіяти всі графічні ядра. До речі, саме цим пояснюється і той факт, що для цих конвертерів різниця за часом конвертації в режимах використання GPU і без використання невелика.

У конвертері Movavi Video Converter 10.2.1 ситуація дещо інша. Цей конвертер здатний дуже ефективно використовувати обчислення на GPU, а тому в режимі використання GPU час конвертації залежить від типу використовуваної відеокарти.

А ось з відеокартою AMD Radeon HD 6850 все як завжди. Чи то драйвер відеокарти «кривий», то чи алгоритми, реалізовані в конвертерах, потребують серйозного доопрацювання, але в разі застосування обчислень на GPU результати або не поліпшуються, або погіршуються.

Якщо говорити більш конкретно, то ситуація наступна. Для конвертера Xilisoft Video Converter Ultimate 7.7.2 при використанні графічного процесора для конвертації першого тестового відеоролика час конвертації збільшується на 43%, при конвертації другого ролика – на 66%.

Причому, конвертер Xilisoft Video Converter Ultimate 7.7.2 характеризується ще й нестабільністю результатів. Розкид за часом конвертації може досягати 40%! Саме тому треба було повторювати всі тести по десять разів і розраховували середній результат.

А ось для конвертерів Wondershare Video Converter Ultimate 6.0.32 і Movavi Video Converter 10.2.1 при використанні графічного процесора для конвертації всіх трьох відеороликів час конвертації не змінюється взагалі! Ймовірно, що конвертери Wondershare Video Converter Ultimate 6.0.32 і Movavi Video Converter 10.2.1 або не використовують технологію AMD APP при конвертації, або відеодрайвер AMD просто «кривий», в результаті чого технологія AMD APP не працює.

ВИСНОВКИ

Таким чином, за підсумками проведеного дослідження були отримані перераховані нижче результати.

1. Виконано огляд існуючих програмних інструментів для спеціалізації програм з підтримкою графічних процесорів. Вибраний інструмент LLVM.mix.

2. Виділено такі алгоритми з перспективних областей, теоретично піддаються спеціалізації, а так само распаралелюванню для відеокарт:

– тензорний добуток з розрідженій матрицею;

– згортка зображення;

– зіставлення шаблонів;

– зіставлення шаблону з автоматом.

3. Проведено експериментальне дослідження ефективності спеціалізації виділених алгоритмів з LLVM.mix на центральному процесорі. Знайдено 2 перерахованих нижче алгоритма, на яких було отримано позитивний ефект:

– тензорне добуток (прискорення до 18%);

– згортка зображень (прискорення до 20%).

4. Досліджено застосовність і ефективність спеціалізаії обраних алгоритмів на GPU з застосуванням LLVM.mix. Показана неможливість застосування обраного спеціалізатора для задач узагальненого програмування для графічних процесорів без його суттєвого виправлення.

Щодо тестування GPU: на підставі проведеного тестування можна зробити наступні важливі висновки. В сучасних відеоконвертер дійсно може застосовуватися технологія обчислень на GPU, що дозволяє підвищити швидкість конвертації. Однак це зовсім не означає, що всі обчислення цілком переносяться на GPU і CPU залишається незадіяним. Як показує тестування, при використанні технології GPGPU центральний процесор залишається завантаженим, а значить, застосування потужних, багатоядерних

центральних процесорів в системах, які використовуються для конвертації відео, залишається актуальним. Винятком з цього правила є технологія AMD APP на графічних процесорах AMD. Наприклад, при використанні конвертера Xilisoft Video Converter Ultimate 7.7.2 з активованою технологією AMD APP навантаження на CPU дійсно знижується, але це призводить до того, що час конвертації не скорочується, а, навпаки, збільшується.

Взагалі, якщо говорити про конвертування відео з додатковим використанням графічного процесора, то для вирішення цього завдання доцільно застосовувати відеокарти з графічними процесорами NVIDIA. Як показує практика, тільки в цьому випадку можна домогтися збільшення швидкості конвертації. Причому потрібно пам’ятати, що реальний приріст в швидкості конвертації залежить від дуже багатьох чинників. Це вхідний і вихідний формати відео, і, звичайно ж, сам відеоконвертер. Конвертери Xilisoft Video Converter Ultimate 7.7.2 і Wondershare Video Converter Ultimate 6.0.32 для цього завдання підходять погано, а ось конвертер і Movavi Video Converter 10.2.1 здатний дуже ефективно використовувати можливості NVIDIA GPU.

Що ж стосується відеокарт на графічних процесорах AMD, то для задач відеоконвертірованія їх не варто застосовувати взагалі. У кращому випадку ніякого приросту в швидкості конвертації це не дасть, а в гіршому – можна отримати її зниження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. JonesNeil D., Gomard Carsten K., Sestoft Peter (1993). Partial Evaluationand Automatic Program Generation.

2. AndersenPeter (1996)**.** Holst. Partial evaluation applied to ray tracing Software Engineering im Scientific Computing**.–**Springer, P.78–85.

3. E.Yu.Sharygin,R.A.ButchatskiyR.A.Zhuykov,A.R.Sher (2017).Query compilation in Postgre SQL by specialization of the DBMSsourcecode // Programming and Computer Software, P.353–365.

4. NickollsJohn, Dally WilliamJ (2010).The GPU computing era // IEEE micro. Vol.30, no. 2.–P.56–69.

5. International Conference on Management of Data (2018). Cypher: An Evolving Query Language for Property Graphs /NadimeFrancis, Alastair Green, Paolo Guagliardo et al.–SIGMOD ‘18.–NewYork,NY, USA : ACM, 2018.–P.1433–1445.

6. Ershov A.P. (1982). Mixed computation: potential applications and problems for study // Theoretical Computer Science.–Vol.18, no. 1.–P.41 –67.

7. Roland Lei, Klaas Boesche, Sebastian Hacket al (2018). AnyDSL: A Partial Evaluation Frameworkfor Programming High- performance Libraries Oct.–Vol.2, no. OOPSLA.–P.119:1–119:30.

8. Eugene Sharygin (2019). Multi-stage compiler-assisted specialize generator built on LLVM, FOSDEM.

9. Mashtalir, S., Mashtalir, V., & Stolbovyi, M. (2018).Representative Based Clustering of Long Multivariate Sequences with Different Lengths. In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP) (pp. 545-548). IEEE.

10. Bodyanskiy, Y., Kobylin, I., Rashkevych, Y., Vynokurova & Peleshko, (2018).Hybrid fuzzy-clustering algorithm of unevenly and asynchronously spaced time series in computer engineering. In 2018 14th International Conference on

Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET) (pp. 930-935). IEEE.

11. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016).Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks. Information Technology and Management Science, 19(1), 23-28.

12. Schafer, J. L., & Graham, J. W. (2002). Missing data: our view of the state of the Mashtalir, S. V., Stolbovyi, M. I., & Yakovlev, S. V. (2019). Clustering Video Sequences by the Method of Harmonic k-Means. Cybernetics and Systems

13. Analysis, 55(2), 200-206. art. Psychological methods, 7(2), 147.

14. Mashtalir, V., Ruban, I., & Levashenko, V. (Eds.). (2019). Advances in Spatio-Temporal Segmentation of Visual Data (Vol. 876). Springer Nature.

15. Kobylin, O., & Lyashenko, V. (2016). Contrast Modification as a Tool to Study the Structure of Blood Components.

16. Mashtalir, S., Mashtalir, V., & Stolbovyi, M. (2017) Video shot boundary detection via sequential clustering. International Journal “Information Theories and Applications, 24(1), 50-59.

17. Bodyanskiy, Y. V., Tyshchenko, O. K., & Mashtalir, S. V. (2019, June). Fuzzy Clustering High-Dimensional Data Using Information Weighting. In International Conference on Artificial Intelligence and Soft Computing (pp. 385-395). Springer, Cham.

18. Oleg, K., Sergii, M., & Mykhailo, S. (2017, October). Video Clustering 70 via Multidimensional Time-Series Analysis. In Proceedings of the 9th International Conference on Information Management and Engineering (pp. 60-63). ACM.

19. Hu, Z., Bodyanskiy, Y. V., Tyshchenko, O. K., & Shafronenko, A.(2019) Fuzzy clustering of incomplete data by means of similarity measures. In 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON) (pp. 957-960). IEEE.

20. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020) The application of non-parametric statistics methods in image classifiers based on structural description components, Telecommunications and Radio Engineering, 79(10), pp. 855-863.

21. Leutenegger S., Chli M., and Siegwart R. (2011) BRISK: Binary Robust Invariant Scalable Keypoints, Proceedings of 2011 IEEE International Conference on Computer Vision (ICCV), pp. 2548-2555.

22. Rublee E., Rabaud V., Konolige K., and Bradski G. (2011) ORB: an efficient alternative to SIFT or SURF, Proceedings of 2011 IEEE International Conference on Computer Vision (ICCV), pp. 2564-2571.

23. Nong Ye. (2013) Data Mining: Theories, Algorithms, and Examples, Florida, USA: CRC Press, 349 p.

24. Sonka M., Hlavac V., and Boyle R. (2014) Image Processing, Analysis, and Machine Vision, Atlanta, USA: Thomson-Engineering, 920 p.

25. Duda R.O., Hart P.E., and Stork D.G. (2000) Pattern classification, Hoboken, USA: John Wiley & Sons, 738 p.

26. Flah P. (2015) Machine learning. The science and art of building algorithms that extract knowledge from data, Moscow, Russia: DMK Press, 400p., (in Russian).

27. Daradkeh Y.I., and Tvoroshenko I. (2020) Technologies for Making Reliable Decisions on a Variety of Effective Factors using Fuzzy Logic, International Journal of Advanced Computer Science and Applications, 11(5), pp.43-50.

28. Kido, S., Hirano, Y., & Hashimoto, N. (2018, January). Detection and classification of lung abnormalities by use of convolutional neural network (CNN) and regions with CNN features (R-CNN). In 2018 International Workshop on Advanced Image Technology (IWAIT) (pp. 1-4). IEEE.

29. Yousef Ibrahim Daradkeh, and Iryna Tvoroshenko (2020) Application of an Improved Formal Model of the Hybrid Development of Ontologies in Complex Information Systems, Applied Sciences, 10(19). p. 6777.

30. Abdulkareem I., Ammar A. Shubber, and Sabah A. (2018) Multi-criteria decision making to select the best monorail route, Global Journal of Engineering Science and Research Management, pp. 16-32.

31. S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," (2018) International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, 2018,pp. 1-10.

32. Baggio, D. L., Emami, S., Escriva, D. M., Ievgen, K., Saragih, J., & Shilkrot, R. (2017). Mastering OpenCV 3. Packt Publishing Ltd.

33. Peters J.F. (2017) Foundations of computer vision: Computational Geometry, Visual Image Structures and Object Shape Detection, Cham, Switzerland: Springer International Publisher, 417 p.

34. Szeliski R. Computer vision: algorithms and applications / Springer Science & Business Media, 2010.

35. Mikolajczyk K., Schmid C. A performance evaluation of local descriptors // IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE. 2005.