МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-28

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.28.24.000.КРБ

СЕМКА
МАКСИМА ОЛЕГОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Аналіз ефективності застосування технології CUDA при рішенні математичних задач»**

Проектний матеріал складається з пояснювальної записки на 80 сторінках та графічного (презентаційного) матеріалу на 19 аркушах (слайдах)

Виконавець _____ (Семко М.О.)

Керівник проекту _____ (Кірєєв І.А.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «24» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 4 (добре) 85

Секретар ЕК _____

АНОТАЦІЯ

Метою даної роботи є аналіз ефективності застосування технології CUDA при рішенні математичних задач, її вживання для підвищення продуктивності паралельних обчислень.

Розглянуто застосування та архітектуру технології CUDA, яка дозволяє використовувати обчислювальні ресурси відеоадаптера для підвищення продуктивності паралельних обчислень при рішенні широкого кола неграфічних задач, зокрема у науково-технічних розрахунках.

Розглянуто сфери застосування обчислень за допомогою відеоадаптеру: аналіз і обробка зображень і сигналів, симуляція фізики, обчислювальна математика, обробка звуку, трасування променів, візуалізація.

У роботі проведено дослідження готових реалізацій систем з використанням загальних розрахунків на графічних процесорах, визначено їх переваги та недоліки, розроблено вимоги до власного рішення та обрано оптимальний набір технологій для його реалізації.

Розглянуто побудову та застосування програми для мережі паралельних обчислень на базі технології CUDA. Для демонстрації обчислювальних можливостей було обрано математичний вираз та складено програму мовою програмування C з використанням засобів взаємодії потоків, а саме бар’єрів та атомарних операцій.

Отримано порівняльну характеристику продуктивності виконання програм для різної розмірності елементів та різної кількості CUDA-ядер. Досліджено ефективність технології CUDA у матричних математичних операціях.

Описано заходи з охорони праці та техніки безпеки.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"15" 01 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Семку Максиму Олеговичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Аналіз ефективності застосування технології CUDA при рішенні математичних задач*

затверджена наказом по коледжу від "02" листопада 2023 р. № 244-А2-ОД

2. Термін здачі студентом кваліфікаційної роботи 13.06.2024р.

3. Вихідні дані до роботи *1. Характеристики сучасних центральних процесорів та відеоадаптерів; 2. Специфікації технології NVidia CUDA; 3. Перелік наукових та інженерних задач для прискорення розрахунку за допомогою технології CUDA; 4. Дослідити коефіцієнт прискорення програм в залежності від кількості ядр для математичної задачі обробки матриць*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналіз технології NVIDIA CUDA

2. Дослідження можливостей технологій CUDA у різних галузях

3. Застосування технологій CUDA у прикладних задачах

4. Реалізація на CUDA паралельного математичного алгоритму;

5. Питання охорони праці та техніки безпеки

5. Перелік графічного матеріалу (слайдів мультимедійної презентації) *Традиційна модель програмування GPGPU; Узагальнена структура відеоадаптера; NVidia Структурна схема гібридної ОС; Рівні обробки даних за технологією NVIDIA CUDA; Схема роботи алгоритму рішення задачі з використанням GPU; Групування потоків в моделі програмування CUDA; Модель пам'яті CUDA; Стадії компіляції CUDA-додатку; Структура паралельної комп'ютерної системи; Паралельний математичний алгоритм; Алгоритм взаємодії потоків; Схема процесу вводу-виведення даних; Приклад заповнення вхідного файлу; Графіки зміни коефіцієнту прискорення та ефективності програми в залежності від кількості ядер*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Кірєєв І.А.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 15.01.2024р

Керівник роботи *Кірєєв І.А.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Вступ. Постановка задачі проектування*	8.05.2024	Виконав
2.	*Дослідження технологій GPGPU*	11.05.2024	Виконав
3.	*Можливості NVIDIA CUDA. Переваги і обмеження*	14.05.2024	Виконав
4.	*Апаратне забезпечення технології CUDA. Склад CUDA*	16.05.2024	Виконав
5.	*Основи створення програм на CUDA*	18.05.2024	Виконав
6.	*Модель програмування CUDA. Модель пам'яті CUDA*	22.05.2024	Виконав
7.	*Середовище програмування CUDA*	24.05.2024	Виконав
8.	*Дослідження можливостей практичного застосування*	26.05.2024	Виконав
	технології CUDA для рішення інженерних задач	28.05.2024	
9.	*Реалізація алгоритму Viola-Jones*	1.06.2024	Виконав
10.	*Розробка алгоритму обчислення паралельних матричних*	4.06.2024	Виконав
	операцій за допомогою CUDA		
11.	*Реалізація на CUDA прикладу паралельного*	7.06.2024	Виконав
	математичного алгоритму та аналіз результатів		
12.	*Розробка питань з охорони праці*	8.06.2024	Виконав
13.	*Оформлення слайдів мультимедійної презентації*	13.06.2024	Виконав

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 28. 24 000. 00 КРБ	Випускна робота		
А4			БКС 28. 24 000. 00 КРБ ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 28. 24 000. 00 КРБ		
Розробив		Семко М.О.		8.06	Аналіз ефективності застосування технології CUDA при рішенні математичних задач	Літ.	Аркуш 4 / Аркушів 80
Перевірив		Кірєєв І.А.		8.06		Н Д П	
Н. Контр.		Петрашова В.І.		09.06		ВСП "ОТФК ОНТУ" гр.2БКС-28	
Затверд.		Іванова Л.В.		10.06			

ЗМІСТ

ВСТУП

Задля 3D-видеоприскорювачів ще кілька років через це із'явилися перші моделі неграфічних розрахунків загального призначення GPGPU (General-Purpose computation on GPUs), адже сучасні відеочіпи містять сотні математичних виконавчих модулів, та ця потужність може використовуватися задля значного ускорення безлічі обчислювально-інтенсивних додатків.

Визначення на ГПУ розвивалися та розвиваються сильно швидко. ТА надалі, два основні виробники відеочіпів, NVIDIA та AMD, розробили та анонсували відповідні платформи під назвою CUDA (Compute Unified Device Architecture) та CTM (Close To Metal або AMD Stream Computing), відповідно. На відміну з попередніх моделей програмування ГПУ, ці були виконані із врахуванням прямого доступу до апаратних можливостей відеокарт. Платформи не сумісні поміж собою, CUDA – це розширення мови програмування C, але CTM – віртуальна машина, виконуюча асемблерний код.

Перерахуємо основні додатки, у котрих зараз застосовуються визначення на ГПУ: аналіз та обробка картинок та сигналів, симуляція фізики, обчислювальна математика, обчислювальна біологія, астрономія, обробка звуку, біоінформатика, біологічні симуляції, електромагнітні симуляції, геоінформаційні системи, військові вживання, гірське планерування, молекулярна динаміка, магнітно-резонансна томографія (MRI), нейромережі, океанографічні дослідження, фізика часток, симуляція згортання молекул білка, квантова хімія, трасування променів, візуалізація, радари, гідродинамічне моделювання (reservoir simulation), штучний інтелект, аналіз супутникових даних, сейсмічна розвідка.

В даній роботі розглядаються можливості моделі CUDA при рішенні математичних завдань, її вживання задля підвищення ефективності одночасних розрахунків. Тематика є актуальною в таких областях, як молекулярна динаміка, фізика твердих тіл, сейсмічні дослідження і інші. Пришвидшення цих розрахунків і знаходження альтернативних пристроїв задля розрахунків дозволить затрачати менше коштів і отримувати більш продуктивні системи.

1 ОСНОВНА ЧАСТИНА

1.1 Аналіз різниці поміж ЦП та ГПУ у розрахунках

В відеочіпах N-VIDIA основний блок – це мульти-процесор із вісьма-десятьма ядрами та сотнями ALU у цілому, декількома тисячами регістрів та невеликою кількістю спільної розділяємої сховища даних. Крім того, відеокарта містить швидку глобальну сховище даних із доступом до неї усіх мульти-процесорів, локальну сховище даних у кожному мульти-процесорі, але разом з цим спеціальну сховище даних задля констант.

Найголовніше – ці кілька ядр мульти-процесора у ГПУ є SIMD (одиночний потік команд, безліч стрімів даних) ядрами. Та ці ядра виконують одні та ті ж інструкції одночасно, такий стиль програмування є звичайним задля графічних методів та багатьох наукових завдань, але вимагає специфічного програмування. Зате такий підхід передбачає збільшити кількість виконавчих модулів поза рахунок їх спрощення.

Отже, перерахуємо основні відмінності поміж архітектурою ЦП та ГПУ. Ядра ЦП створені задля реалізації одного потоку послідовних інструкцій із максимальною продуктивністю, але ГПУ проектуються задля швидкого реалізації великого значення паралельно виконуваних стрімів інструкцій. Універсальні процесори оптимізовані задля досягнення високої ефективності єдиного потоку команд, обробляючого та цілі значення та значення із плаваючою крапкою. При цьому доступ до сховища даних випадковий.

Розробники ЦП прагнуть добитися реалізації як можливо більшого значення інструкцій паралельно, задля збільшення ефективності. Задля цього, починаючи із обчислювачів Intel Pentium, із'явилося суперскалярне реалізації, яке забезпечує реалізації двох інструкцій поза такт, але Pentium Pro відрізнявся позачерговим виконанням інструкцій. Але у паралельного реалізації послідовного потоку інструкцій є певні базові обмеження та збільшенням значення виконавчих модулів кратного збільшення швидкості не добитися.

В відеочіпів робота проста та розпаралелена напочатку. Відеочіп приймає

на вході групу полігонів, проводить усі необхідні операції, та на виході видає пікселі. Обробка полігонів та пікселів незалежна, їх можливо обробляти паралельно, окремо один з одного. Через це, через паралельну організацію роботи у ГПУ використовується велика кількість виконавчих модулів, котрі легко завантажити, на відміну з послідовного потоку інструкцій задля ЦП. Крім того, сучасні ГПУ разом з цим спроможні виконувати більше однієї інструкції поза такт (dual issue).

ГПУ відрізняється з ЦП ще та поза принципами доступу до сховища даних. В ГПУ він зв'язаний та легко передбачений – коли із сховища даних читається тексель текстури, то через деякий період прийде період та задля сусідніх текселей. При записі разом з цим – піксель записується в фреймбуфер, та через кілька тактів записуватиметься розташований поряд із ним, та відеочіпу, на відміну з універсальних обчислювачів, просто не потрібна кеш-сховище даних великого розміру.

Робота із пам'яттю у ГПУ та ЦП кілька відрізняється. Так, не усі центральні процесори мають вбудовані контролери сховища даних, але в усіх ГПУ зазвичай є по кілька контролерів, наприклад до восьми 64-бітових каналів у чіпі N-VIDIA GT200. Крім того, на відеокартах застосовується швидша сховище даних, та у результаті відеочіпам доступна у рази більша пропускна спроможність сховища даних, яке разом з цим вельми важливо задля одночасних розрахунків, яке оперують із величезними потоками даних.

В універсальних процесорах великі значення транзисторів та площа чіпа використані задля буферів команд, апаратного передбачення галуження та величезних об'ємів кеш-сховища даних. Відеочіпи використовують транзистори задля масивів виконавчих модулів, управління потоками модулів, розділяємої сховища даних невеликого об'єму та контролерів сховища даних на кілька каналів. Вищеперелічене не прискорює реалізації окремих стрімів, воно передбачає чіпові обробляти кілька тисяч стрімів, яке одночасно виконуються чіпом та яке вимагають високої пропускної спроможності сховища даних.

Універсальні центральні процесори використовують кеш-сховище даних

задля збільшення ефективності поза рахунок зниження затримок доступу до сховища даних, але ГПУ використовують кеш або спільну сховище даних задля збільшення смуги пропускання. ЦП знижують затримки доступу до сховища даних поза допомогою кеш-сховища даних великого розміру, але разом з цим передбачення галужень коду. Відеочіпи обходять проблему затримок доступу до сховища даних поза допомогою одночасного реалізації тисяч стрімів – у той період, коли один із стрімів чекає значення із сховища даних, відеочіп може виконувати визначення іншого потоку без очікування та затримок.

Є безліч відмінностей та у підтримці багатопоточності. ЦП виконує 1-2 потоки розрахунків на одне процесорне ядро, але відеочіпи спроможні підтримувати до 1024 стрімів на кожний мульти-процесор, котрих у чіпі кілька штук. ТА коли перемикання із одного потоку на іншій задля ЦП коштує сотні тактів, то ГПУ перемикає кілька стрімів поза один такт.

Коротко можливо сказати, яке на відміну з сучасних універсальних ЦП, відеочіпи призначені задля одночасних розрахунків із великою кількістю арифметичних операцій. ТА значно більше значення транзисторів ГПУ працює по прямому призначенню – обробці масивів даних, але не управляє виконанням (flow control) численних послідовних обчислювальних стрімів (рис.1.1).

Рисунок 1.1. Розташування логічних модулів у ЦП та ГПУ

В результаті, основою задля ефективного впровадження потужності ГПУ у наукових та інших неграфічних розрахунках є розпаралелювання методів на сотні виконавчих модулів, наявних в відеочіпах. Наприклад, безліч додатків по

молекулярному моделюванню відмінно пристосована задля розрахунків на відеочіпах, вони вимагають великих обчислювальних потужностей та через це зручні задля одночасних розрахунків. АЛЕ впровадження кілька ГПУ дає ще більше обчислювальних потужностей задля вирішення подібних завдань.

Реалізації розрахунків на ГПУ показує відмінні значення у алгоритмах, яке використовують паралельну обробку даних. Тобто, коли одну та ту ж послідовність математичних операцій застосовують до великого об'єму даних. При цьому кращі значення досягаються, коли відношення значення арифметичних інструкцій до звернень до сховища даних достатньо велике. Це пред'являє менші вимоги до управління реалізації (flow control), але висока щільність математики та великий об'єм даних відміняє необхідність в великих кешах, як на ЦП.

У результаті усіх описаних вище відзнак, теоретична продуктивність відеочіпів значно перевершує продуктивність ЦП.

1.2 Аналіз можливостей NVIDIA CUDA

Технологія CUDA – це програмно-апаратна обчислювальна архітектура N-VIDIA, заснована на розширенні мови Сі, котра дає можливість організації доступу до набору інструкцій графічного прискорювача та управління його пам'яттю при організації одночасних розрахунків. CUDA допомагає реалізовувати методи, здійснимі на графічних процесорах відеоприскорювачів Ge-force восьмого покоління та старше, але разом з цим Quadro та Tesla.

Перерахуємо основні характеристики CUDA:

- уніфіковане програмно-апаратне рішення задля одночасних розрахунків на відеочіпах N-VIDIA;
- великий набір підтримуваних рішень, з мобільних до мультичіпових;
- стандартна мова програмування Сі;
- стандартні бібліотеки чисельного аналізу FFT (швидке перетворення Фурье) та BLAS (лінійна алгебра);
- оптимізований обмін даними поміж ЦП та ГПУ;

- взаємодія із графічними API OPENGL та DIRECTX;
- підтримка 32- та 64-бітових операційних систем: Windows, Linux та MAC OS X;
- можливість розробки на низькому рівні.

Середовище розробки CUDA (CUDA Toolkit) включає:

- компілятор nvcc;
- бібліотеки FFT та BLAS;
- профілювальник;
- відладчик gdb задля ГПУ;
- CUDA runtime драйвер у комплекті стандартних драйверів N-VIDIA;
- керівництво по програмуванню;
- CUDA Developer SDK (початковий код, утиліти та документація).

1.3 Аналіз переваг та обмежень CUDA

ІЗ точки зіру програміста, графічний конвеєр є набором стадій обробки. Блок геометрії генерує трикутники, але блок растеризації – пікселі, яке відображуються на моніторі. Традиційна модель програмування GPGPU виглядає таким чином (рис.1.2).

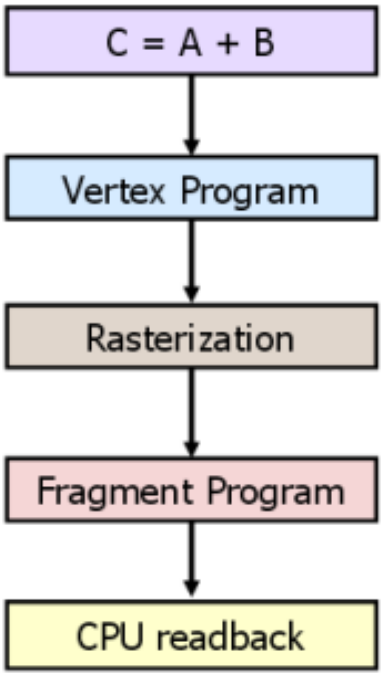


Рисунок 1.2. Традиційна модель програмування GPGPU

Щоб перенести визначення на ГПУ у рамках такої моделі, потрібний спеціальний підхід. Навіть поелементне складання двох векторів зажадає відрисовки фігури на екрані або в позаекранний буфер. Фігура растеризується, колір будь-якого пікселя обчислюється поза заданою програмою (піксельним шейдером). Програма зчитує вхідні значення із текстур задля будь-якого пікселя, складає їх та записує в вихідний буфер. Через це, вживання GPGPU задля розрахунків загального призначення має обмеження в вигляді сильно великої складності навчання розробників. І та інших обмежень вистачає, адже піксельний шейдер – це всього тільки формула залежності підсумкового кольору пікселя з його координати, але мова піксельних шейдерів – мова запису цих формул із Сі-подобним синтаксисом.

Програмно-апаратна архітектура задля розрахунків на ГПУ компанії N-VIDIA відрізняється з попередніх моделей GPGPU тим, яке передбачає писати програми задля ГПУ на справжній мові Сі із стандартним синтаксисом, вказівниками та необхідністю у мінімумі розширень задля доступу до обчислювальних ресурсів відеочіпів. CUDA не залежить з графічних API, та володіє деякими особливостями, призначеними спеціально задля розрахунків загального призначення.

Переваги CUDA перед традиційним підходом до GPGPU-розрахунків:

- інтерфейс програмування додатків CUDA заснований на стандартній мові програмування Сі із розширеннями, яке спрощує процес вивчення та впровадження архітектури CUDA;
- CUDA забезпечує доступ до сховища даних, яке розділяється поміж потоками, розміром у 16 Кб на мульти-процесор, котра може існувати використана задля організації кеша із широкою смугою пропускання, у зрівнянні із вибірками текстур;
- ефективніша передача даних поміж системною та відеопам'яттю;
- відсутність необхідності у графічних API із надмірністю та накладними витратами;
- лінійна адресація сховища даних (gather та scatter), можливість запису поза

довільними адресами;

- апаратна підтримка цілочисельних та бітових операцій.

Основні обмеження CUDA:

- відсутність підтримки рекурсії задля виконуваних процедур;
- блоки стрімів виконуються в вигляді невеликих груп, званих варп (warp), розмір котрих – 32 потоки. Це мінімальний об'єм даних, котрі спроможні оброблятися у мульти-процесорах;
- CUDA передбачає працювати та із блоками, яке містять з 64 до 512 стрімів.

1.4 Огляд апаратного забезпечення моделі CUDA

Усі відеокарти, яке володіють підтримкою CUDA, спроможні допомогти у прискоренні більшості вимогливих завдань, починаючи з аудіо- та відеообробок, та закінчуючи медициною та науковими дослідженнями.

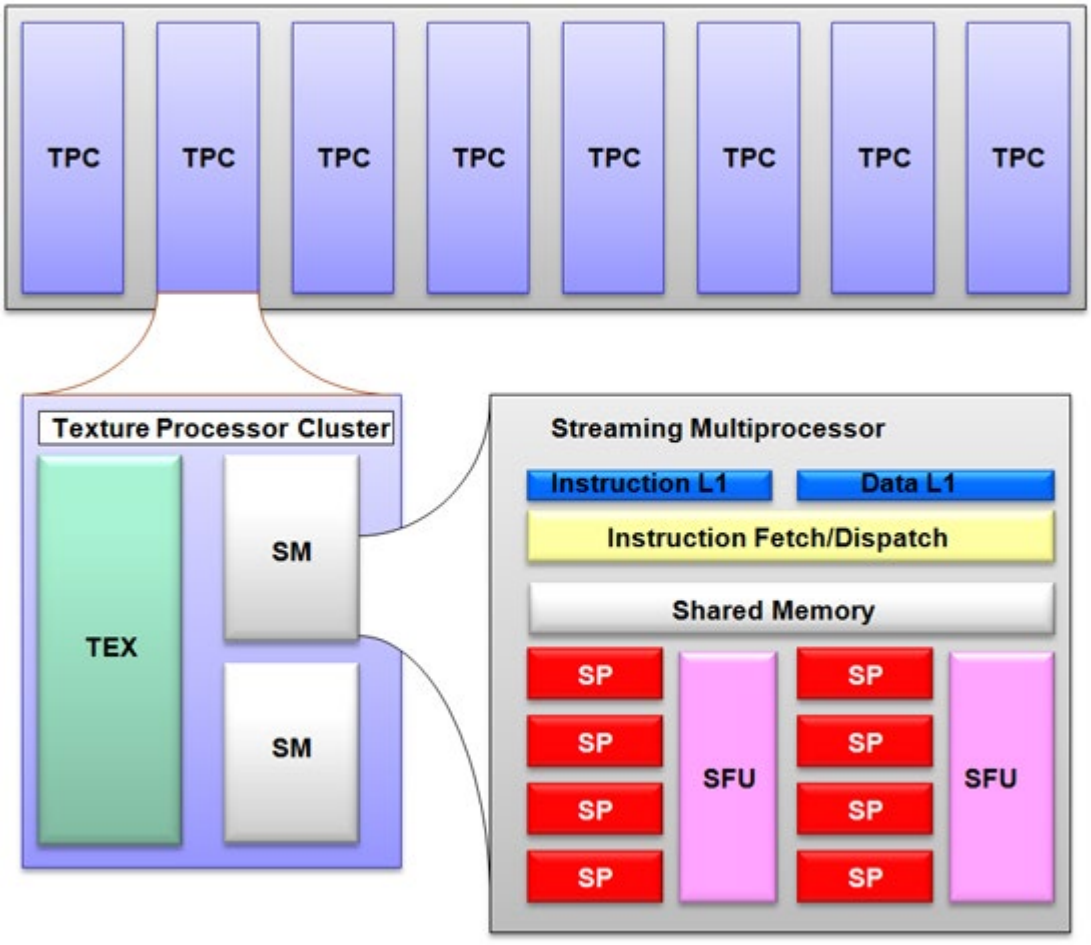


Рисунок 1.3. Узагальнена структура відеоадаптера N-vidia

Єдине реальне обмеження полягає у через це, яке багато CUDA-програм вимагають мінімум 256 мегабайт відеопам'яті, та це – одна із найважливіших технічних характеристик задля CUDA-додатків.

Ядро шейдерів n-vidia складається із кількох кластерів обчислювачів текстур (Texture Processor Cluster, TPC). Кожний кластер, по суті, складається із блоку текстури та двох потокових мульти-процесорів (streaming multiprocessor). Останні включають початок конвеєра (front end), яке виконує читання та декодування інструкцій, але разом з цим відсилання їх на реалізації, та кінець конвеєра (back end), яке складається із восьми обчислювальних пристроїв та двох суперфункціональних пристроїв SFU (Super Function Unit), де інструкції виконуються поза принципом SIMD, тобто одна інструкція застосовується до усіх стрімів в варпі. n-vidia називає такий спосіб реалізації SIMT (single instruction multiple threads, одна інструкція, багато стрімів).

Важливо відзначити, яке кінець конвеєра працює на частоті у два рази більше, ніж його початок. На практиці це означає, яке дана частка виглядає у два рази "ширше", ніж вона є насправді (тобто як 16-канальний блок SIMD замість восьмиканального).

Потокові мульти-процесори працюють таким чином: кожний такт початок конвеєра вибирає варп, готовий до реалізації, та запускає реалізації інструкції. Щоб інструкція застосувалася до усіх 32 потокам в варпі, кінцю конвеєра буде треба чотири такти, але адже він працює на подвоєній частоті у зрівнянні із початком, буде треба тільки два такти (із точки зіру початку конвеєра). Через це, щоб початок конвеєра не простоював такт, але апаратне забезпечення було максимально завантажене, у ідеальному випадку можливо чергувати інструкції кожний такт – класична інструкція у один такт та інструкція задля SFU – у іншій.

Актуальний список підтримуючих CUDA продуктів можливо отримати на вебсайті N-VIDIA. Розрахунки CUDA підтримують усі продукти серій Ge-force 600, 500, 400, 200, Ge-force 9 та Ge-force 8, в через це числі та мобільні продукти, починаючи із Ge-force 8400M, але разом з цим чіпсети Ge-force 8100, 8200 та 8300. Разом з цим підтримкою CUDA володіють сучасні продукти Quadro та все

Tesla: S1070, C1060, C870, D870 та S870. Особливо відзначимо, яке були анонсовані та відповідні рішення задля високопродуктивних розрахунків: Tesla C1060 та S1070 (рис.2.4, 2.5). ГПУ у них застосований GT200, у C1060 він один, у S1070 – чотири.

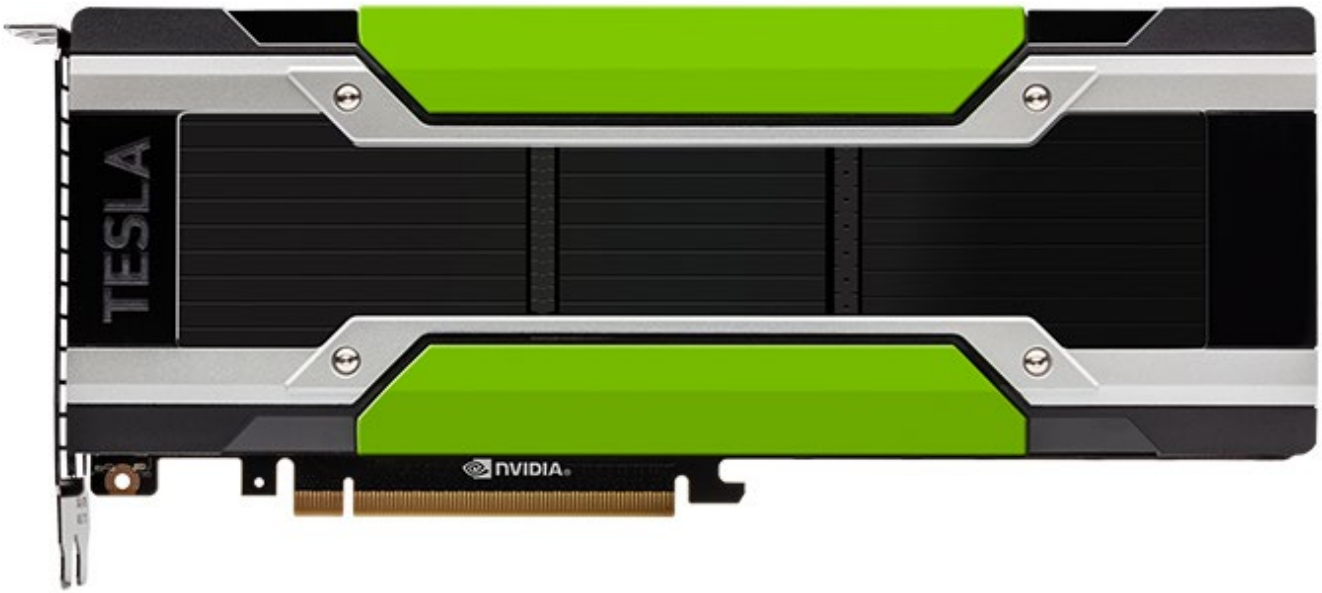


Рисунок 1.4. Відеоадаптер N-vidia сімейства Tesla C1060

Рисунок 1.5. Відеоадаптер N-vidia сімейства Tesla S1070

Впровадження кількох відеоадаптерів у одній обчислювальній системі передбачає ще більш прискорити процес вирішення завдань та оптимізувати їх реалізації. У цьому випадку використовується гібридна обчислювальна система, подібна до рис.1.6.

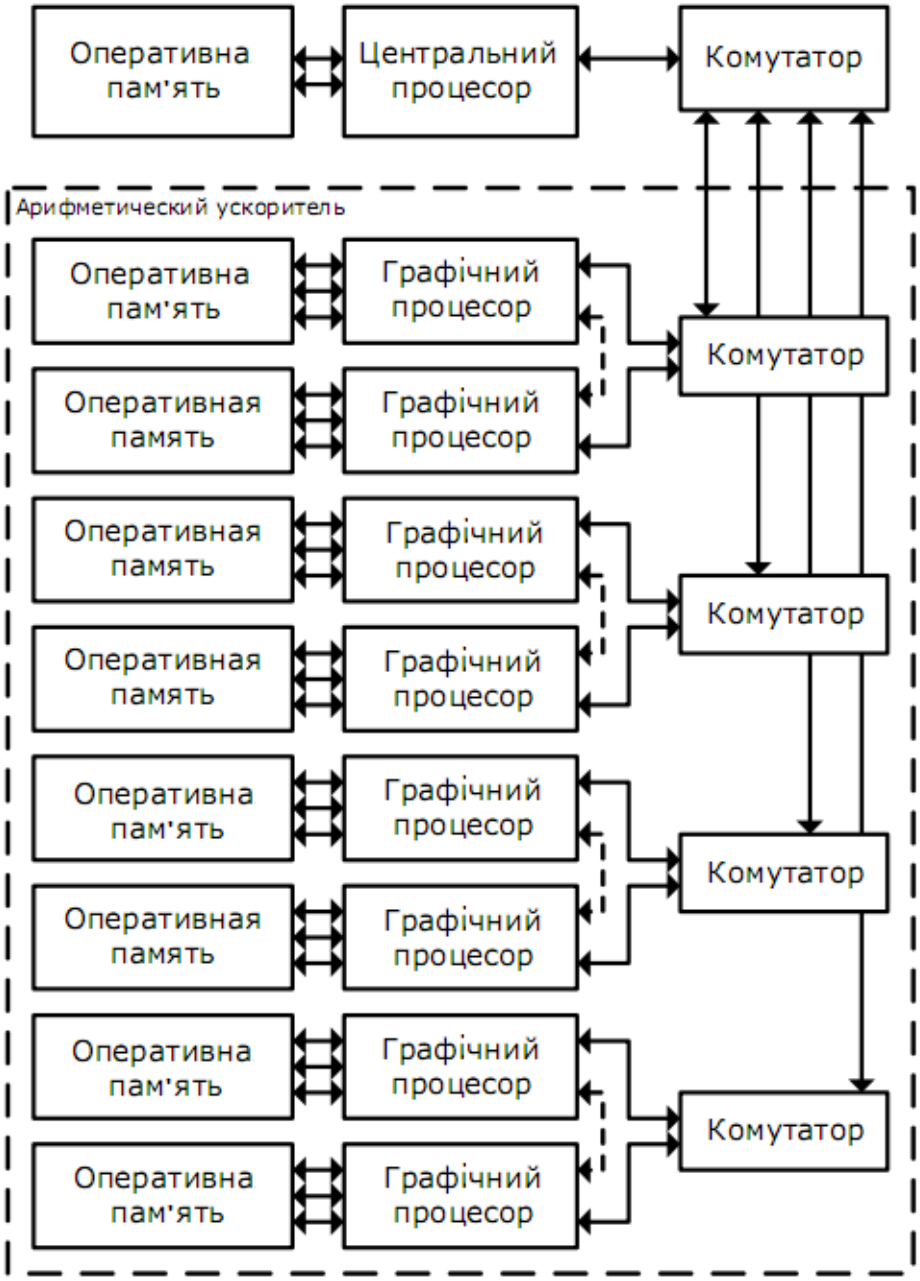


Рисунок 1.6. Структурна схема гібридної обчислювальної системи

1.5 Визначення рівнів обробки даних N-VIDIA CUDA

CUDA включає два API: високого рівня та низького (CUDA Driver API), хоча у одній програмі одночасне впровадження обох неможливо, треба застосовувати або один або інший. Високорівневий працює «зверху» низькорівневого, усі виклики runtime транслюються у прості інструкції, яке обробляються низькорівневим Driver API (рис.1.7). Але навіть «високорівневий» API передбачає знання про пристрій та роботу відеочіпів N-VIDIA, сильно високого рівня абстракції там немає.

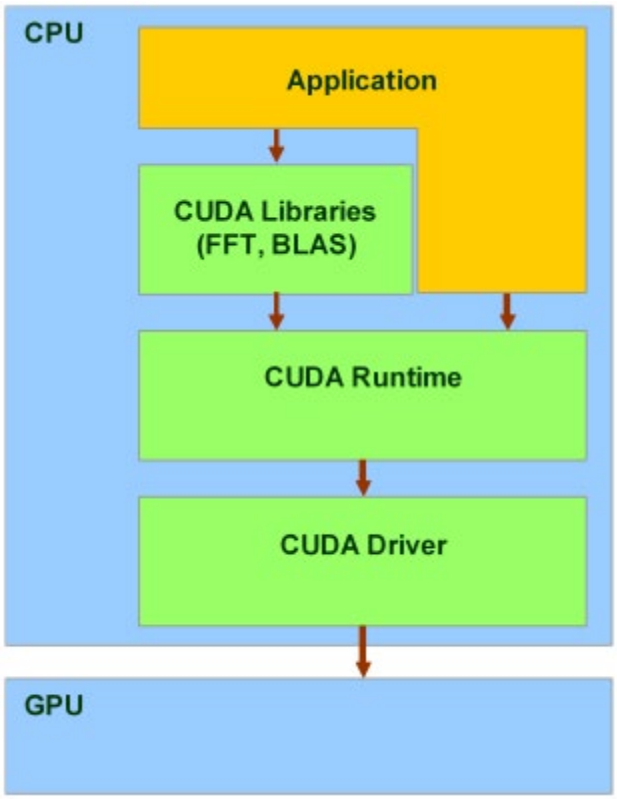


Рисунок 1.7. Рівні обробки даних поза технологією N-VIDIA CUDA

Існує ще один рівень, вищий – дві бібліотеки:

CUBLAS – CUDA варіант BLAS (Basic Linear Algebra Subprograms), призначений задля розрахунків завдань лінійної алгебри та використовуючий прямий доступ до ресурсів ГПУ;

CUFFT – CUDA варіант бібліотеки Fast Fourier Transform задля обчислення швидкого перетворення Фурьє, широко використовуваного при обробці сигналів. Підтримуються наступні типи перетворень: complex-complex (C2C), real-complex (R2C) та complex-real (C2R).

Розглянемо ці бібліотеки докладніше. CUBLAS – це перекладені мовою CUDA стандартні методи лінійної алгебри, на даний момент підтримується тільки певний набір основних процедур CUBLAS. Бібліотеку сильно легко застосовувати: треба створити матрицю та векторні об'єкти у сховища даних відеокарти, заповнити їх даними, викликати необхідні процедури CUBLAS, та завантажити значення із відеопам'яті назад у системну. CUBLAS містить спеціальні процедури задля створення та знищення модулів у сховища даних ГПУ, але разом з цим задля читання та запису даних у цю сховище даних.

Підтримувані процедури BLAS: рівні 1, 2 та 3 задля дійсних чисел, рівень 1 CGEMM задля комплексних. Рівень 1 – це векторно-векторні операції, рівень 2 – векторно-матричні операції, рівень 3 – матричні задля матриці операції.

CUFFT – CUDA варіант процедури швидкого перетворення Фурьє – широко використовуваної та сильно важливої при аналізі сигналів, фільтрації та через це подібне. CUFFT надає простий інтерфейс задля ефективного визначення FFT на відеочіпах виробництва N-VIDIA без необхідності у розробці власного варіанту FFT задля ГПУ. CUDA варіант FFT підтримує 1D, 2D, та 3D перетворення комплексних та дійсних даних, пакетного реалізації задля кількох 1D трансформацій у паралелі, розміри 2D та 3D трансформацій спроможні існувати у межах [2…16384], задля 1D підтримується розмір до 8 мільйонів елементів.

1.6 Визначення методу рішення задачі із на базі ГПУ

Розглянемо базові архітектурні особливості відеочіпів N-VIDIA. ГПУ складається із кількох кластерів модулів текстур (Texture Processing Cluster). Кожний кластер складається із укрупненого блоку вибірок текстур та двох-трьох потокових мульти-процесорів, кожний із котрих складається із восьми обчислювальних пристроїв та двох суперфункціональних модулів (рис.1.8). Усі інструкції виконуються поза принципом SIMD, коли одна інструкція застосовується до усіх стрімів в warp (термін із текстильної промисловості, у CUDA це група із 32 стрімів – мінімальний об'єм даних, яке обробляються мульти-процесорами). Цей спосіб реалізації назвали SIMT (single instruction multiple threads – одна інструкція та багато стрімів).

Кожний із мульти-процесорів має певні ресурси. Так, є спеціальна розділяємо сховище даних об'ємом 16 кілобайт на мульти-процесор. Ця розділяєма сховище даних передбачає обмінюватися інформацією поміж потоками одного блоку. Важливо, яке усі потоки одного блоку завжди виконуються одним та тим же мульти-процесором, але потоки із різноманітних модулів обмінюватися даними не спроможні, та треба пам'ятати це обмеження.

Рисунок 1.8. Архітектура відеочіпу (ГПУ) N-VIDIA

Розділяєма сховище даних часто буває корисною, окрім тих випадків, коли кілька стрімів звертаються до одного банку сховища даних. Мульти-процесори спроможні звертатися та до відеопам'яті, але із великими затримками та гіршою пропускною спроможністю. Задля ускорення доступу та зниження частоти звернення до відеопам'яті, у мульти-процесорів є по 8 кілобайт кеша на константи та значення текстур.

Мульти-процесор використовує 8192-16384 (задля G8x/G9x та GT2xx, відповідно) регістрів, спільних задля усіх стрімів усіх модулів, яке виконуються на ньому. Максимальне значення модулів на один мульти-процесор задля G8x/G9x дорівнює восьми, але значення warp – 24 (768 стрімів на один мульти-процесор). Всього відеокарти серій Ge-force 8 та 9 спроможні обробляти до 12288 стрімів одночасно. Ge-force GTX 280 на основі GT200 пропонує до 1024 стрімів на мульти-процесор, у ньому є 10 кластерів по три мульти-процесори, оброблюючих до 30720 стрімів. Знання цих обмежень передбачає оптимізувати методи під доступні ресурси.

Першим кроком при перенесенні існуючого додатку на CUDA є його профілізація та визначення ділянок коду, яке є вузьким місцем, яке гальмує роботу. Коли серед таких ділянок є відповідні задля швидкого паралельного реалізації, ці процедури переносяться на Сі-розширення CUDA задля реалізації на ГПУ. Програма компілюється поза допомогою компілятора, яке поставляється, N-VIDIA, який генерує код та задля ЦП, та задля ГПУ. При виконанні програми, центральний процесор виконує свої порції коду, але ГПУ виконує CUDA код із найбільш важкими паралельними обчисленнями. Ця частка, призначена задля ГПУ, називається ядром (kernel). В ядрі визначаються операції, котрі будуть виконані над даними.

Відеочіп отримує ядро та створює копії задля будь-якого елементу даних. Ці копії називаються потоками (thread). Потік містить лічильник, регістри та стан. Задля великих об'ємів даних, таких як обробка картинок, запускаються мільйони стрімів. Потоки виконуються групами по 32 штуки (warp). Warp'ам призначається реалізації на певних потокових мульти-процесорах. Кожний мульти-процесор складається із восьми ядр – потокових обчислювачів, котрі виконують одну інструкцію MAD поза один такт. Задля реалізації одного 32-потокового warp'але треба чотири такти роботи мульти-процесора (мова йде про частоту shader domain, котра дорівнює 1.5 Ггц та вище).

Мульти-процесор не є традиційним багатоядерним процесором, він відмінно пристосований задля багатопоточності, підтримуючи до 32 warp'ів одночасно. Кожний такт апаратне забезпечення обирає, який із warp'ів виконувати, та перемикається з одного до іншого без втрат у тактах. Коли проводити аналогію із центральним процесором, це схоже на одночасного реалізації 32 програм та перемикання поміж ними кожний такт без втрат на перемикання контексту. Реально ядра ЦП підтримують одночасне реалізації однієї програми та перемикаються на інші із затримкою у сотні тактів.

Здійснення методу розроблюваної програми в вигляді CUDA-ядра передбачає мінімізувати пересилки даних через глобальну сховище даних відеоадаптеру (рис.1.9).

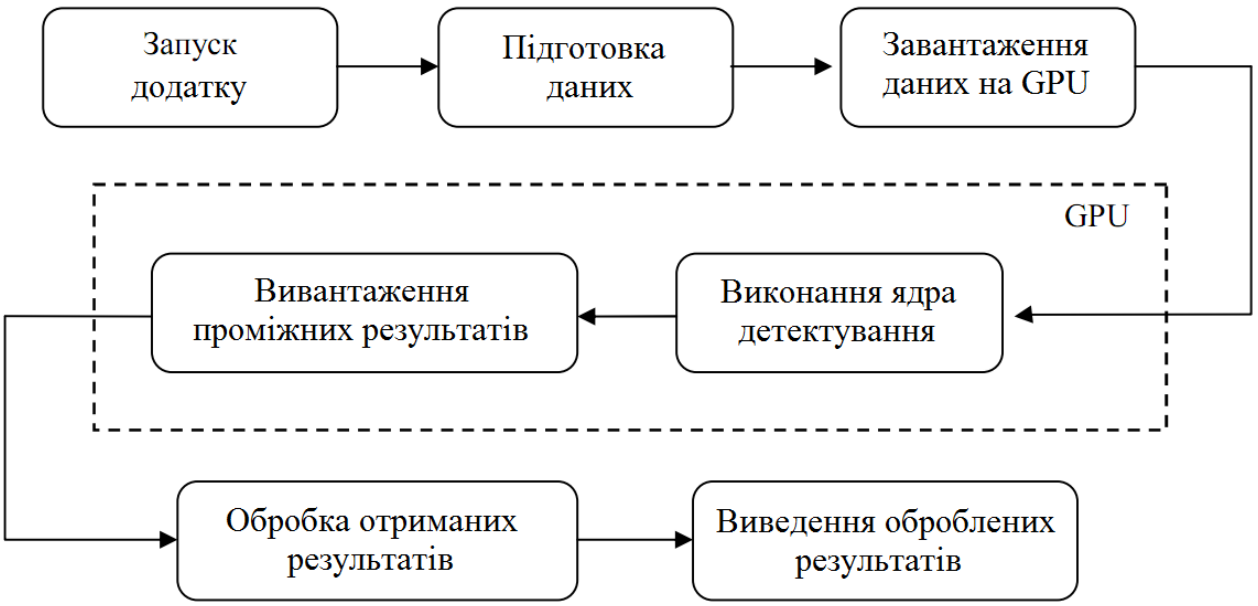


Рисунок 1.9. Схема роботи методу рішення задачі із використанням ГПУ

1.7 Аналіз моделі програмування CUDA

CUDA використовує паралельну модель розрахунків, коли кожний із SIMD обчислювачів виконує ту ж інструкцію над різними елементами даних паралельно. ГПУ є обчислювальним пристроєм, співпроцесором (device) задля центрального процесора (host), яке володіє власною пам'яттю та обробляє паралельно велику кількість стрімів. Ядром (kernel) називається функція задля ГПУ, яке виконується потоками (аналогія із 3D графіки – шейдер). Кожний потік скалярний, не вимагає упаковки даних у 4-компонентні вектори, яке зручніше задля більшості завдань. Кількість логічних стрімів та модулів стрімів перевершує кількість фізичних виконавчих пристроїв, яке дає добру масштабованість задля всього модельного ряду рішень компанії.

Модель програмування у CUDA передбачає групування стрімів. Потоки об'єднуються у блоки стрімів (thread block) – одновимірні або двовимірні сітки стрімів, яке взаємодіють поміж собою поза допомогою розділяємої сховища даних, та точок синхронізації. Програма (ядро, kernel) виконується над сіткою (grid) модулів стрімів (thread blocks), див. рис 1.10. Одночасно виконується одна сітка. Кожний блок може існувати одно-, двох- або тривимірним поза формою, та може складатися із 512 стрімів на поточному апаратному забезпеченні.

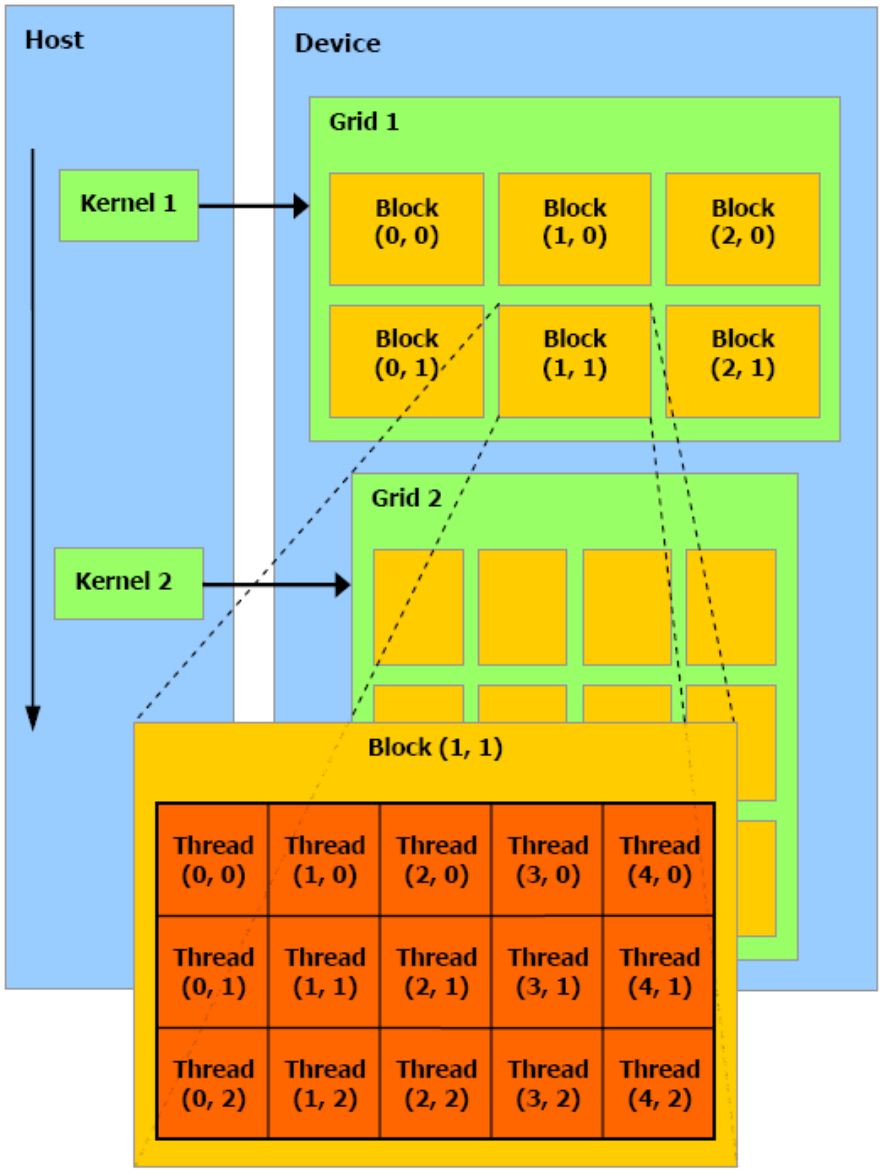


Рисунок 1.10. Групування стрімів у моделі програмування CUDA

Угрупування модулів у сітки передбачає піти з обмежень та застосувати ядро до більшого значення стрімів поза один виклик. Це допомагає та при масштабуванні. Коли у ГПУ недостатньо ресурсів, він виконуватиме блоки послідовно. В зворотному випадку, блоки спроможні виконуватися паралельно, яке важливо задля оптимального розподілу роботи на відеочіпах різного рівня, починаючи з мобільних та інтегрованих.

1.8 Аналіз моделі сховища даних CUDA

Модель сховища даних у CUDA (рис.1.11) відрізняється можливістю побайтової адресації, підтримкою як gather, так та scatter.

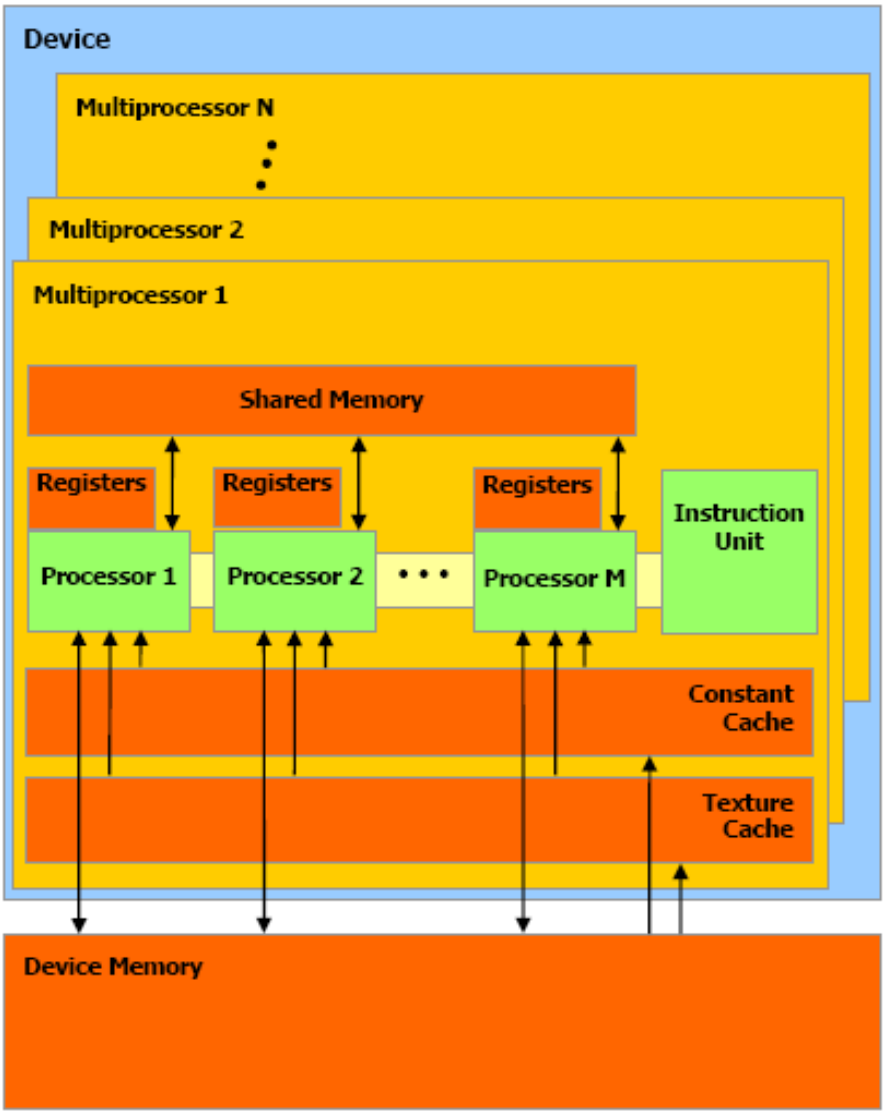


Рисунок 1.11. Модель сховища даних CUDA

Доступна достатньо велика кількість регістрів на кожний потоковий процесор, до 1024 штук. Доступ до них сильно швидкий, зберігати у них можливо 32-бітові цілі або значення із плаваючою крапкою.

Кожний потік має доступ до наступних типів сховища даних:

Глобальна сховище даних – найбільший об'єм сховища даних, доступний задля усіх мульти-процесорів на відеочіпі, розмір складає з 256 мегабайт до 1.5 гігабайт на поточних рішеннях. Володіє високою пропускною спроможністю, більше 100 гігабайтів/с задля топових рішень N-VIDIA, але сильно великими затримками у кілька сотен тактів. Не кешується, підтримує узагальнені інструкції load та store, та звичайні вказівники на сховище даних.

Локальна сховище даних – це невеликий об'єм сховища даних, до якого

має доступ тільки один потоковий процесор. Вона відносно повільна – така ж, як та глобальна. Розділювана сховище даних – це 16-кілобайтний (в відеочіпах нинішньої архітектури) блок сховища даних із спільним доступом задля усіх потокових обчислювачів у мульти-процесорі. Вона забезпечує взаємодію стрімів, управляється розробником безпосередньо та має низькі затримки. Переваги розділюваної сховища даних: впровадження в вигляді керованого програмістом кеша першого рівня, зниження затримок при доступі виконавчих модулів (ALU) до даних, скорочення значення звернень до глобальної сховища даних. Сховище даних констант – область сховища даних об'ємом 64 кілобайти (задля нинішніх ГПУ), доступна тільки задля читання всіма мульти-процесорами. Вона кешується по 8 кілобайт на кожний мульти-процесор. Достатньо повільна – затримка у кілька сотень тактів поза відсутності потрібних даних у кеші.

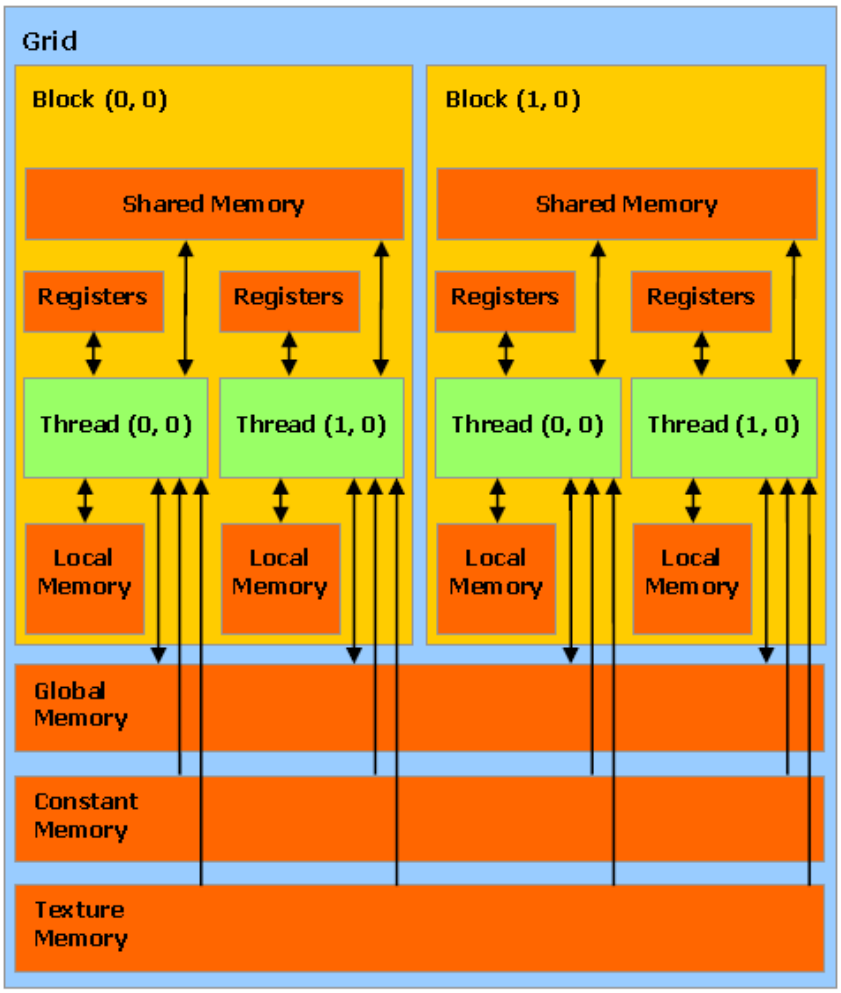


Рисунок 1.12. Моделі доступу до різноманітних типів сховища даних

Сховище даних текстури – блок сховища даних, доступний задля читання всіма мульти-процесорами. Вибірка даних здійснюється поза допомогою модулів текстур відеочіпа, через це надаються можливості лінійної інтерполяції даних без додаткових витрат. Кешується по 8 кілобайт на кожний мульти-процесор. Повільна, як глобальна – сотні тактів затримки поза відсутності даних у кеші.

Природно, яке глобальна, локальна, текстурна та сховище даних констант – це фізично одна та і ж сховище даних, відома як локальна відеопам'ять відеокарти. Їх відзнаки у різноманітних алгоритмах кешування та моделях доступу (рис.1.12). Центральний процесор може оновлювати та запрошувати тільки зовнішню сховище даних: глобальну, константну та текстуру.

ІЗ написаного вище зрозуміло, яке CUDA передбачає спеціальний підхід до розробки, не зовсім такий, як прийнятий у програмах задля ЦП. Треба пам'ятати про різні типи сховища даних, про те, яке локальна та глобальна сховище даних не кешується та затримки при доступі до неї набагато вище, адже вона фізично знаходиться у окремих мікросхемах.

Типовий, але не обов'язковий шаблон рішення завдань:

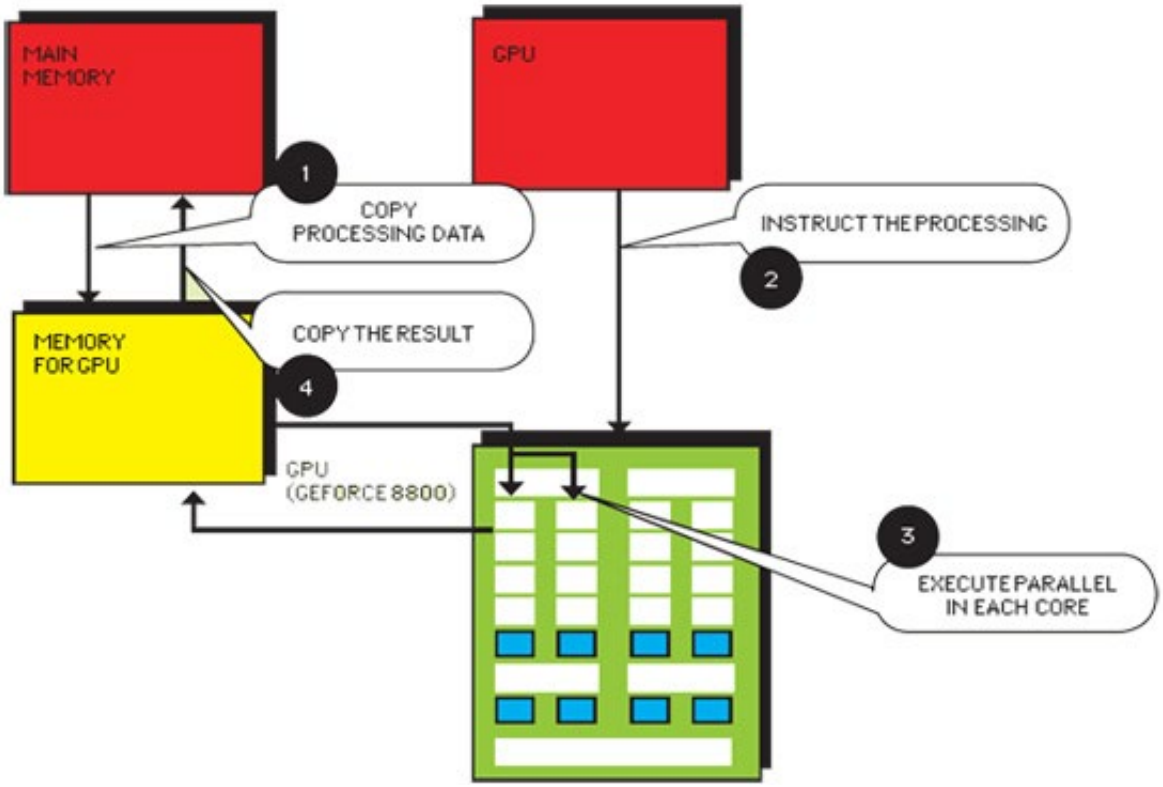
- задача розбивається на підзадачі;
- вхідні значення діляться на блоки, котрі вміщаються у сховище даних;
- кожний блок обробляється блоком стрімів;
- підблок підвантажується у розділяєму сховище даних із глобальної;
- над даними у розділяємій сховища даних проводяться визначення;
- значення копіюються із розділяємої сховища даних назад у глобальну.

1.9 Застосування середовища програмування CUDA

До складу CUDA входять runtime-бібліотеки:

- загальна частина, яке надає вбудовані векторні типи та підмножини викликів RTL, підтримувані на ЦП та ГПУ;
- ЦП-компонента, задля управління одним або декількома ГПУ;
- ГПУ-компонента, яке надає специфічні процедури задля ГПУ.

Основний процес додатка CUDA працює на універсальному процесорі (host), він запускає кілька копій процесів kernel на відеокарті. Код задля ЦП робить наступне: ініціалізує ГПУ, розподіляє сховище даних на відеокарті та системі, копіює константи у сховище даних відеокарти, запускає кілька копій процесів kernel на відеокарті, копіює отриманий результат із відеопам'яті, визволяє сховище даних та завершує роботу (рис.1.13).



(1) КОПІЮЄМО ЗНАЧЕННЯ ІЗ ОСНОВНОЇ СХОВИЩА ДАНИХ У СХОВИЩЕ ДАНИХ ВІДЕОКАРТИ.

(2) ПЕРЕДАЄМО УПРАВЛІННЯ ГПУ.

(3) ГПУ ВИКОНУЄ КОМАНДУ ПАРАЛЕЛЬНО У КОЖНОМУ ЯДРІ.

(4) КОПІЮЄМО РЕЗУЛЬТАТ ІЗ СХОВИЩА ДАНИХ ВІДЕОКАРТИ У СХОВИЩЕ ДАНИХ

Рисунок 1.13. Схема взаємодії поміж ЦП та ГПУ

У якості прикладу розглянемо ЦП-код задля складання векторів, представлений у CUDA:

```
//Розмір вектора у елементах
const int N = 1048576;
//Розмір вектора у байтах
const int dataSize = N * sizeof(float);
//Виділення сховища даних ЦП
```

```
float *h_A = (float *)malloc(dataSize);
float *h_B = (float *)malloc(dataSize);
float *h_C = (float *)malloc(dataSize);
//Виділення сховища даних ГПУ
float *d_A, *d_B, *d_C;
cudaMalloc((void **)&d_A, dataSize));
cudaMalloc((void **)&d_B, dataSize));
cudaMalloc((void **)&d_C, dataSize));
//Ініціалізувати h_A[], h_B[]…
//Скопіювати вхідні значення у ГПУ задля обробки
cudaMemcpy(d_A, h_A, dataSize, cudaMemcpyHostToDevice) );
cudaMemcpy(d_B, h_B, dataSize, cudaMemcpyHostToDevice) );
//Запустити ядро із N / 256 модулів по 256 стрімів
//Передбачено, яке N кратне 256
vectorAdd<<<N / 256, 256>>>(d_C, d_A, d_B);
//Зчитати значення ГПУ

cudaMemcpy(h_C, d_C, dataSize, cudaMemcpyDeviceToHost) );
```

Процедури, яке виконуються відеочіпом, мають наступні обмеження: відсутня рекурсія, немає статичних змінних усередині процедур та змінного значення аргументів. Підтримується два види управління пам'яттю: лінійна сховище даних із доступом по 32-бітовим вказівникам, та CUDA-масиви із доступом тільки через процедури вибірки текстури.

Програми на CUDA спроможні взаємодіяти із графічними API: задля рендеринга даних, згенеруваних у програмі, задля зчитування результатів рендеринга та їх обробки засобами CUDA (наприклад, при реалізації фільтрів постобробки). Задля цього ресурси графічних API спроможні існувати відображені (із отриманням адреси ресурсу) у простір глобальної сховища даних CUDA. Підтримуються наступні типи ресурсів графічних API: Buffer Objects (PBO / VBO) у OPENGL, вершинні буфери та текстури (2D, 3D та кубічні карти) Direct3D9.

Стадії компіляції CUDA-додатку показані на рис.1.14.

Компіляція виконується в кілька етапів. Спочатку витягується код, яке відноситься до ЦП, який передається стандартному компілятору. Код, призначений задля ГПУ, спочатку перетворюється у проміжну мову PTX. Вона подібна до асемблера та передбачає вивчати код в пошуках потенційних неефективних ділянок.

Файли початкового коду на CUDA-Сі компілюються поза допомогою програми NVCC, котра є оболонкою над іншими інструментами, та викликає їх: cudacc, g++, cl та ін. NVCC генерує: код задля центрального процесора, який компілюється разом із рештою частин додатку, написаною на чистому Сі. Виконувані файли із кодом на CUDA у обов'язковому порядку вимагають наявність бібліотек CUDA runtime library (cudart) та CUDA core library (cuda).

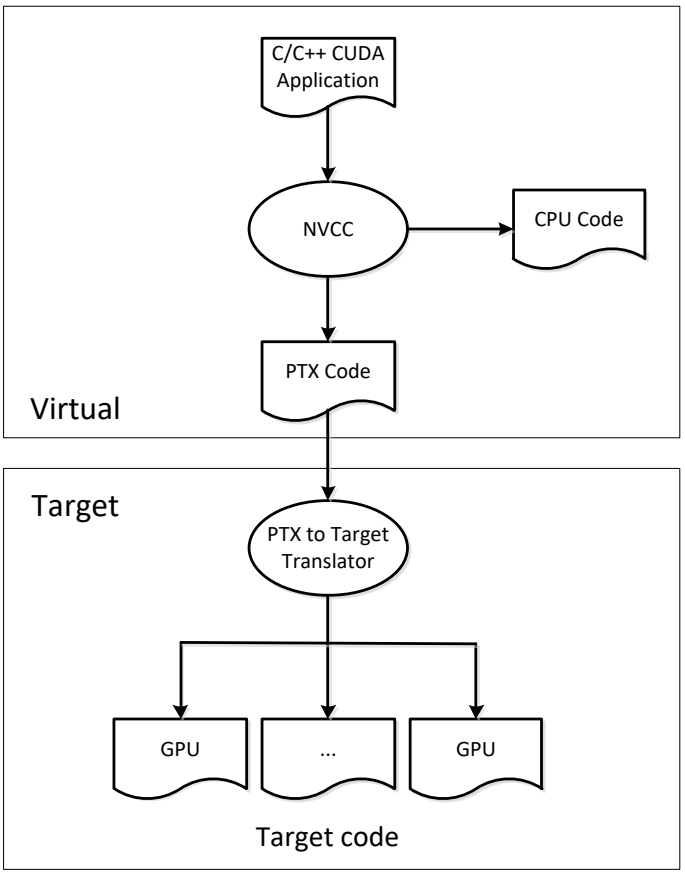


Рисунок 1.14. Стадії компіляції CUDA-додатку

1.10 Оптимізація програм на CUDA

Природно, у рамках даної роботи неможливо розглянути серйозні питання оптимізації у CUDA-програмуванні. Через це просто коротко розглянемо базові речі. Задля ефективного впровадження можливостей CUDA треба забути про звичайні методи написання програм задля ЦП, та застосовувати ті методи, котрі добре розпаралелюються на тисячі стрімів. Разом з цим важливо знайти оптимальне місце задля зберігання даних (регістри, сховище даних та через це подібне), мінімізувати передачу даних поміж ЦП та ГПУ, застосовувати

буферизацію. В загальних рисах, при оптимізації програми CUDA треба намагатися добитися оптимального балансу поміж розміром та кількістю модулів. Більша кількість стрімів у блоці знизить вплив затримок сховища даних, але знизить та доступне значення регістрів. Крім того, блок із 512 стрімів неефективний, сама N-VIDIA рекомендує застосовувати блоки по 128 або 256 стрімів, як компромісне значення задля досягнення оптимальних затримок та значення регістрів.

Серед основних моментів оптимізації програм CUDA: як можливо активніше впровадження розділяємої сховища даних, адже вона значно швидша поза глобальну відеопам'ять відеокарти; операції читання та запису із глобальної сховища даних мають існувати об'єднані (coalesced) по можливості. Задля цього треба застосовувати спеціальні типи даних задля читання та запису відразу по 32/64/128 біта даних однією операцією. Коли операції читання важко об'єднати, можливо спробувати застосовувати вибірки текстур.

1.11 Здійснення методу Viola-Jones задля N-VIDIA CUDA

Комплекс методів задля виявлення модулів, представлений у 2001 році Полом Віолою (Paul Viola) та Майклом Джонсом (Michael Jones) [1], логічно складається із двох компонентів: навчання класифікатора та безпосередньо виявлення модулів на зображенні. На практиці основні вимоги до швидкості роботи пред'являються до другого методу. В даному розділі розглядається здійснення цього методу задля архітектури N-VIDIA CUDA, котра передбачає у кілька разів прискорити його роботу у зрівнянні із реалізацією на центральному процесорі у бібліотеці Open-cv [7].

Метод Viola-Jones виявлення модулів добре відомий. Через це перерахуємо тільки найбільш значущі задля даної теми його особливості:

1. Велика кількість звернень до сховища даних. Задля визначення відгуку одного класифікатора треба з 11 до 15 звернень до сховища даних;
2. Визначення інтегрального відтворення – особливе представлення вихідного відтворення, яке передбачає швидке визначення характерних рис (особливостей), котрі використовуються при класифікації

відтворення. Кожний піксель інтегрального відтворення являє собою суму значень усіх пікселів вихідного відтворення, розташованих вище та лівіше поточного пікселя;

3. Впровадження каскаду класифікаторів, яке представляє собою об'єднання у ланцюжок класифікаторів поза таким принципом, яке вихід одного є входом задля іншого. Через це, коли ділянка відтворення не проходить один класифікатор, він відразу відкидається та не йде по всьому ланцюжку.

Задля ефективної реалізації методу необхідно разом з цим врахувати особливості архітектури N-VIDIA CUDA, із якими можливо ознайомитися у [5]. Розглянутий метод реалізований як одне CUDA-ядро. Це дозволило мінімізувати пересилання даних через глобальну сховище даних графічного прискорювача (див. рис.1.9).

Правильний вибір розміру блоку стрімів та значення використовуваних регістрів передбачає досягти значення коефіцієнта впровадження мульти-процесора порядку 2/3, яке є достатнім задля нівелювання впливу латентності при роботі із пам'яттю.

Вихідні задля методу значення, але саме каскад класифікаторів та аналізоване відтворення, розміщуються в текстурній та константній сховища даних графічного прискорювача. Каскад класифікаторів є незмінним, необхідний всім потокам одночасно, яке передбачає ефективно застосовувати особливості роботи константної сховища даних. Розміщення вихідного відтворення в текстурній сховища даних передбачає реалізувати ефективний метод його масштабування шляхом вибірки із текстури із апаратною білінійною фільтрацією.

Через велику кількість звернень до сховища даних при роботі даного методу необхідно застосовувати розділяєму сховище даних графічного прискорювача як керований L1-кеш. Усі потоки мульти-процесора аналізують ділянки відтворення, яке частково перекриваються, через це у розділяєму сховище даних копіюється частина вихідного відтворення, достатня задля

забезпечення усіх стрімів даними задля класифікації.

Вузьким місцем реалізації методу на графічному прискорювачі є необхідність пересилання даних поміж хостом та прискорювачом. Задля пошуку зразків різного розміру метод Viola-Jones передбачає аналіз відтворення у різноманітних масштабах. Впровадження моделі «zero copy», реалізованої у усіх сучасних графічних прискорювачах, передбачає скоротити накладні витрати при пересиланні даних практично до нуля поза рахунок їх асинхронного реалізації.

Як згадувалося вище, масштабування відтворення реалізовано, використовуючи можливості текстурної сховища даних. Масштабоване відтворення зчитується безпосередньо у розділяєму сховище даних, де та здійснюється його аналіз. Це виключає додаткові звернення до глобальної сховища даних, яке позитивно позначається на часі роботи методу. Проведене порівняння із алгоритмом масштабування відтворення, реалізованим у бібліотеці примітивів NPP, це підтверджує (рис.1.15).

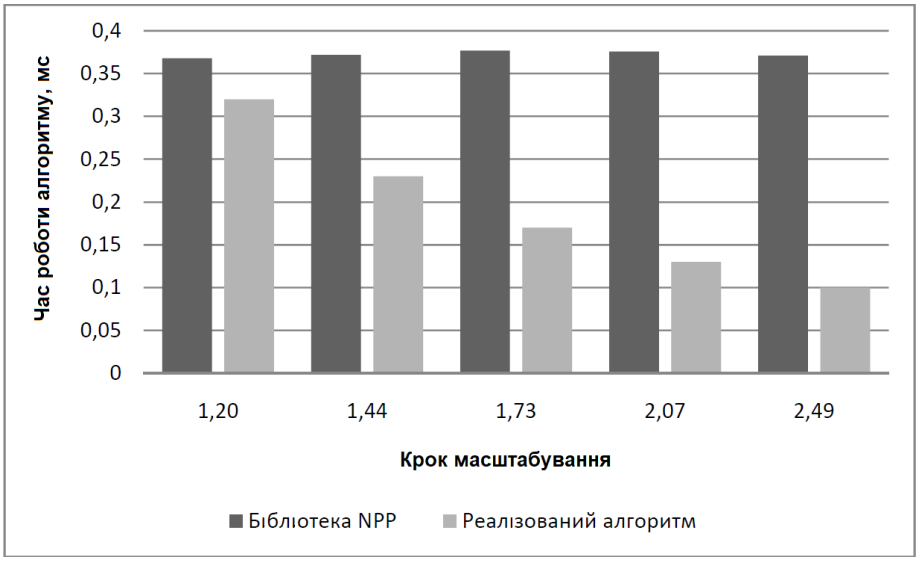


Рисунок 1.15. Порівняння методів масштабування відтворення

Визначення інтегрального відтворення є ключовим кроком в роботі методу Viola-Jones. Процес визначення інтегрального відтворення добре описаний та вивчений. Найочевиднішим способом розпаралелювання цього процесу є розбиття його на два етапи. Однак задля графічного прискорювача здійснення такого методу ускладняється через необхідність синхронізації

результатів роботи окремих мульти-процесорів. Задля полегшення процесу обчислення інтегрального відтворення у даній роботі пропонується застосовувати часткове інтегральне відтворення, яке обчислюється незалежно задля ділянок вихідного відтворення, вже розміщених у сховища даних. Це стає можливим, через це яке у розділюваній сховища даних будь-якого мульти-процесора міститься достатньо даних задля усіх його стрімів. Завдяки цьому робота окремих мульти-процесорів стає незалежною, яке виключає витрати часу на синхронізацію їх роботи. На рис. 1.16 наводиться порівняльна характеристика отриманого методу визначення інтегрального відтворення із аналогічним алгоритмом, яке міститься у бібліотеці NPP.

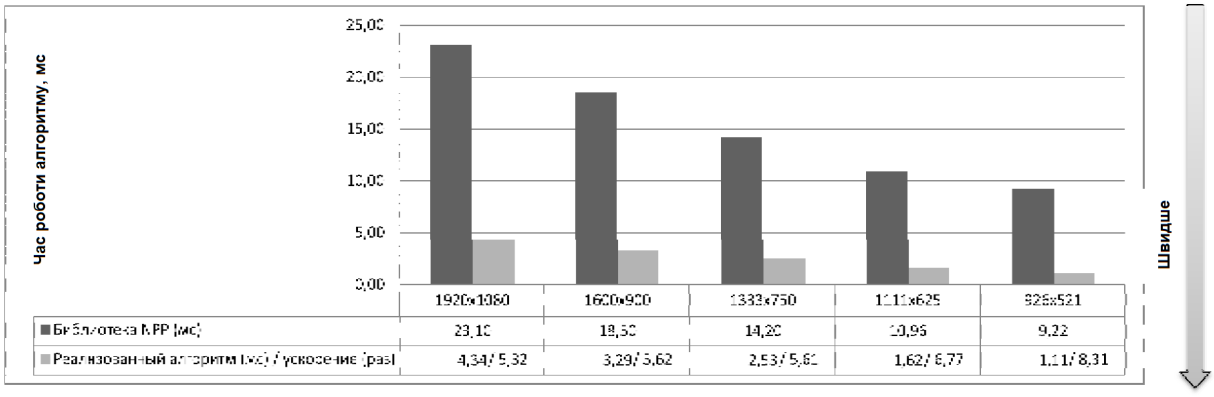


Рис унок 1.16. Порівняння ефективності методів визначення інтегрального відтворення

Тестування отриманої реалізації методу Viola-Jones проводилося на зображеннях трьох розмірів: 1920x1080, 1080x720 та 640x360. Задля тестування застосовувався каскад класифікаторів, запропонований у роботі [2]. Здійснювався пошук картинок із мінімальним розміром 20x20 із кроком масштабування вихідного відтворення 1,2.

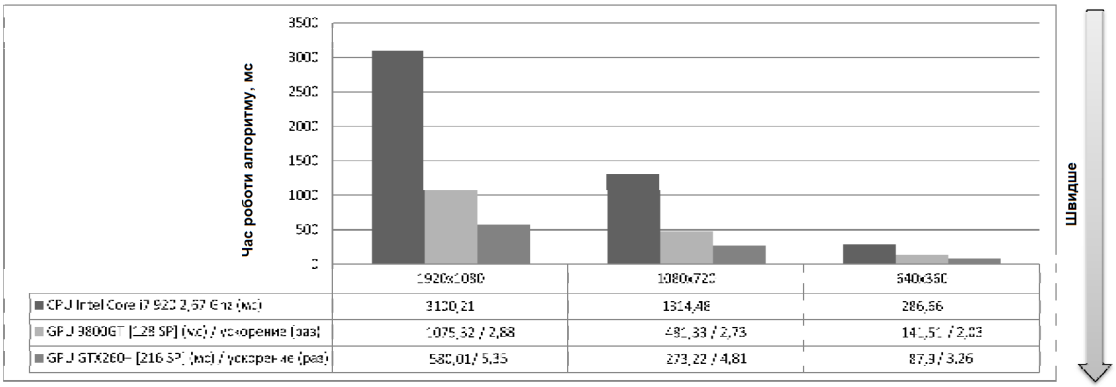


Рисунок 1.17. Значення вимірювання ефективності методу Viola-Jones

Рис.1.17 містить значення вимірювання ефективності методу у цілому на різноманітних графічних прискорювачах. Задля порівняння приведена продуктивність методу, реалізованого у бібліотеці Open-cv.

Можливо відзначити ускорення роботи методу на графічному прискорювачі з 2 до 5 разів в зрівнянні із однопоточними варіантами, яке працюють на центральному процесорі. Такий результат у кілька разів перевищує отриманий автором роботи [4]. Важливим є добра масштабованість отриманої реалізації методу. Це передбачає розраховувати на ще більше ускорення на більш сучасних графічних прискорювачах.

Слід зазначити, яке сам метод не сильно пристосований задля реалізації на графічній карті. Велике значення звернень до сховища даних, простої стрімів при роботі каскаду класифікаторів негативно позначаються на ефективності. Але навіть при цьому отримана здійснення методу на сучасних графічних прискорювачах працює у кілька разів швидше, ніж аналоги на центральному процесорі.

1.12 Програмування задля N-VIDIA CUDA задачі розповсюдження світла

У даний період лазерні методи набули широкого поширення у безконтактній неруйнуючій діагностиці внутрішньої структури різноманітних оптично неоднорідних модулів, зокрема, вони знаходять впровадження в медицині, біофізиці, науках про матеріали, фізиці атмосфери та інших областях.

Задля підвищення ефективності сучасних методів лазерної діагностики, але разом з цим задля розробки нових методів, необхідно докладне вивчення особливостей процесу поширення світла у різноманітних середовищах, включаючи біологічні тканини. Однак, рішення даної задачі утруднено тим, яке на даний момент не існує точної теорії задля опису поширення світла у структурно-неоднорідних середовищах, але експериментальні дослідження ускладнені труднощами підтримання сталості їх структурно динамічних параметрів. В зв'язку із цим все більшу роль набуває комп'ютерне моделювання цього процесу (див., наприклад, [1]). Даний підхід передбачає більш ретельно

вивчити особливості процесу поширення лазерного пучка у модельних середовищах, але разом з цим дослідити залежність отриманих результатів з різноманітних параметрів вимірювальної системи та досліджуваного об'єкта, яке буває достатньо важко у експерименті.

При моделюванні поширення світла в багатошаровому середовищі будемо вважати, яке задача вирішується в тривимірному просторі, але середовище складається із набору плоскопаралельних шарів. Кожний шар є нескінченно широким, описується товщиною та низкою оптичних характеристик.

Результатами моделювання є значення наступних фізичних величин:

- просторовий розподіл інтенсивності розсіяного назад променя на поверхні середовища;
- просторовий розподіл інтенсивності розсіяного вперед променя на задньому кордоні середовища;
- об'ємний розподіл поглиненої інтенсивності у середовищі.

Практичне розв'язання поставленої задачі із використанням сучасних обчислювачів вимагає значного часу розрахунків. В зв'язку із цим актуальним є задача ускорення цього процесу

1.12.1 Загальний опис методу розв'язання задачі

Одним із підходів до вирішення задачі поширення світла у багатошарових середовищах є метод статистичного моделювання Монте-Карло [1, 4]. Під методом Монте-Карло розуміється сукупність прийомів, яке дозволяють отримувати необхідні рішення поза допомогою багаторазових випадкових випробувань. Оцінки шуканої величини виводяться статистичними шляхом.

Стосовно до задачі поширення променя у багатошаровому середовищі метод Монте-Карло полягає у багаторазовому повторенні обчислення траєкторії руху фотона у середовищі, виходячи із заданих параметрів середовища. Загальний метод вирішення задачі представлений на схемі [1] (рис. 1.18). Задля будь-якого фотона вищевказані дії виконуються незалежно. Значення фотонів у реальних експериментах може існувати достатньо великим (10^8 – 10^{10}), із чого

можливо припустити, яке дана задача може існувати ефективно реалізована при використанні архітектури графічних адаптерів (ГПУ). В роботі описується здійснення представленого вище методу на графічному адаптері із використанням моделі CUDA [2].

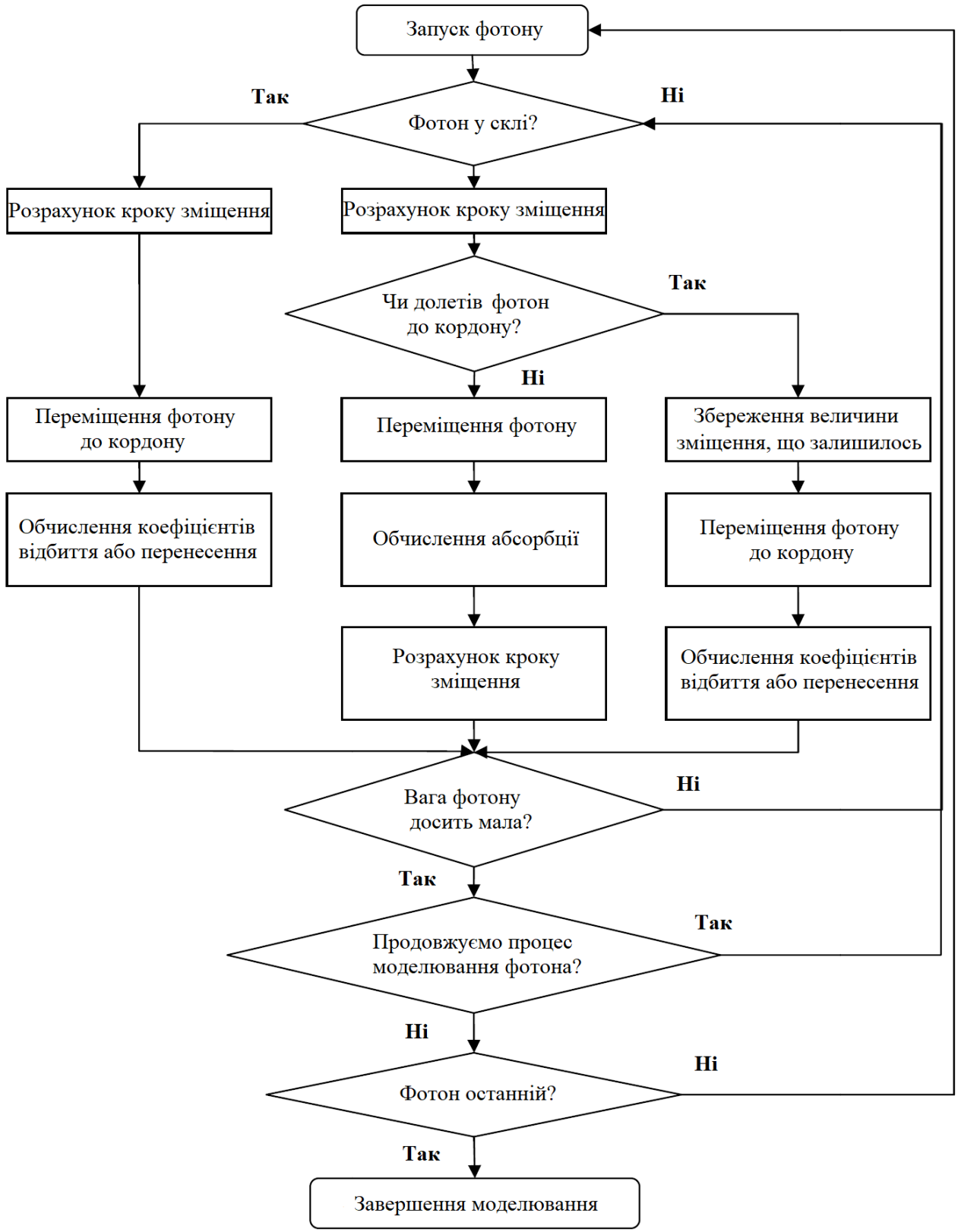


Рисунок 1.18. Метод моделювання поширення світла методом Монте-Карло

1.12.2 Схема розпаралелювання та оптимізація

В розглянутій задачі необхідні задля моделювання дії виконуються незалежно над кожним фотоном, яке передбачає ефективно застосувати схему розпаралелювання, при якій кожний потік обчислює траєкторію руху свого фотона. Адже значення фотонів велике, запропонована схема передбачає завантажити усі доступні обчислювальні ядра графічного адаптера.

Слід зазначити, яке задля підвищення ефективності будь-якого потоку має сенс проводити визначення відразу над групою фотонів (у поточній реалізації розмір групи фотонів дорівнює 1000).

Задля забезпечення збіжності методу моделювання поширення світла необхідно, по-перше, наявність довгої послідовності різноманітних випадкових чисел, по-друге, організація роботи будь-якого конкретного обчислювального потоку зі своєї підпослідовністю, причому усі ці підпослідовності не повинні перетинатися.

Генератор випадкових чисел, реалізований у стандартній бібліотеці мови програмування *C* (функція *rand*) породжує недостатньо довгу послідовність випадкових чисел, та до того ж не може існувати використаний при програмуванні на ГПУ. Через це було прийнято рішення адаптувати метод генератора MCG59 задля роботи у кілька стрімів на графічний адаптер. В поточній реалізації усі потоки працюють із однією послідовністю випадкових чисел, але при цьому кожний потік використовує свої *(id + i *N)* елементи цієї послідовності (тут *id* – ідентифікатор гілки, *N* – загальне значення стрімів). Важливою умовою ефективності програм на ГПУ є правильне впровадження наявних типів сховища даних. Зокрема задля вирішення даної задачі необхідні наступні набори даних: інформація про середовище (зберігається в розділяємій сховища даних), поточні характеристики фотона (зберігаються в регістрах) та результуючі значення (мають великий обсяг, зберігаються в глобальній сховища даних). Розділяєма сховище даних та регістри є швидкими типами сховища даних, але мають невеликий обсяг, у зв'язку із чим застосовувати їх задля роботи із результуючими даними не представляється можливим. Глобальна сховище

даних, навпаки, передбачає зберігати великі обсяги даних, але є відносно повільною. Однак у даній задачі звернень до результуючих даних небагато, через це істотного впливу на продуктивність впровадження повільної сховища даних не робить.

Значення, котрі стосуються результатів, є загальними задля усіх фотонів, звідки виникає необхідність звернення до цих даних одночасно із різноманітних стрімів, але відповідно, потрібна синхронізація. В поточній реалізації задля цього використовуються атомарні операції, яке працюють із 64-бітними цілими числами (задля їх впровадження версія compute capability графічного адаптера повинна існувати 1.2 та вище).

Впровадження механізмів синхронізації зазвичай сильно зменшує продуктивність додатків. Вже згадана задача не є винятком, проте тут все залежить з розмірності результуючих масивів, котра є вхідним параметром методу. При малій розмірності (100 елементів у кожному масиві) одночасних звернень до одних та тих же даних багато, швидкість роботи програми низька. Але коли збільшити значення елементів будь-якого масиву до 10 000 (тим самим підвищивши роздільну здатність задачі), то вплив синхронізації на продуктивність програми буде мінімальним.

Додатково слід зазначити, яке задля вирішення необхідної задачі достатньо одинарної точності, але впровадження операцій із одинарної точністю на ГПУ разом з цим істотно прискорює додаток. До того ж у даному випадку можливе впровадження більш швидких, але менш точних математичних процедур, яке надаються вбудованою бібліотекою CUDA.

1.12.3 Впровадження кількох ГПУ

Технологія CUDA передбачає застосовувати одночасно кілька графічних адаптерів, встановлених у рамках одного обчислювального вузла. Кожний ГПУ має свій унікальний ідентифікатор, поза допомогою якого можливо запускати реалізації задачі на вибраному пристрої. При цьому задля одночасної роботи кількох ГПУ необхідно створити на ЦП певне значення одночасних стрімів, щоб кожний потік працював із власним графічним адаптером. Спосіб створення

стрімів на центральному процесорі може існувати будь-яким, зокрема у даному дослідженні використовувалася технологія OpenMP.

ІЗ метою організації паралельної роботи кількох ГПУ, встановлених на різноманітних обчислювальних вузлах (наприклад, у рамках обчислювального кластера), наведений вище підхід може існувати узагальнено. Задля цього необхідно додатково застосувати одну із технологій паралельного програмування задля систем із розподіленою пам'яттю, наприклад, MPI [3].

Задля впровадження кількох графічних адаптерів треба організувати коректну роботу із наявними даними:

- перед початком реалізації методу створюються керуючі структури, котрі зберігають значення задля роботи генератора випадкових чисел;
- перед початком реалізації методу робиться кілька копій структури із інформацією про модельоване середовище (вхідні значення задачі), кожна копія використовується на своєму ГПУ;
- структура стану фотона створюється та використовується тільки у рамках обчислювального потоку на графічному адаптері, через це не вимагає реалізації додаткових дій при переході на кілька ГПУ;
- проміжні значення накопичуються у сховища даних будь-якого із використовуваних графічних адаптерів, але після закінчення основного методу підсумовуються на центральному процесорі.

Таким чином, розглянутий метод передбачає створити працюючу на кількох ГПУ програму поза невелику кількість достатньо простих кроків.

1.12.4 Значення розрахунків

В табл. 1.1 наведено період роботи програми (у секундах) у залежності з значення фотонів та обчислювального пристрою. Характеристики обладнання:

- Процесор: Intel Xeon E5520 2.27 ГГц, 4 ядра;
- Кількість ядр в процесорі: 4;
- Кількість обчислювачів: 2;
- Об'єм оперативної сховища даних: 12 Гб;
- Графічний адаптер: N-vidia Tesla C1060.

Таблиця 1.1. Період роботи методу (у секундах) у залежності з значення фотонів та обчислювального пристрою

Обчислювальний пристрій	*Значення фотонів*				
	10^4	10^5	10^6	10^7	10^8
Xeon E5520, 1 ядро (1 core)	0,50	4,90	49,3	494	4951
2 x Xeon E5520, 8 ядр (8 cores)	0,11	0,94	7,60	74	730
Tesla C1060 (1 ГПУ)	0,06	0,11	0,60	5,13	50,7

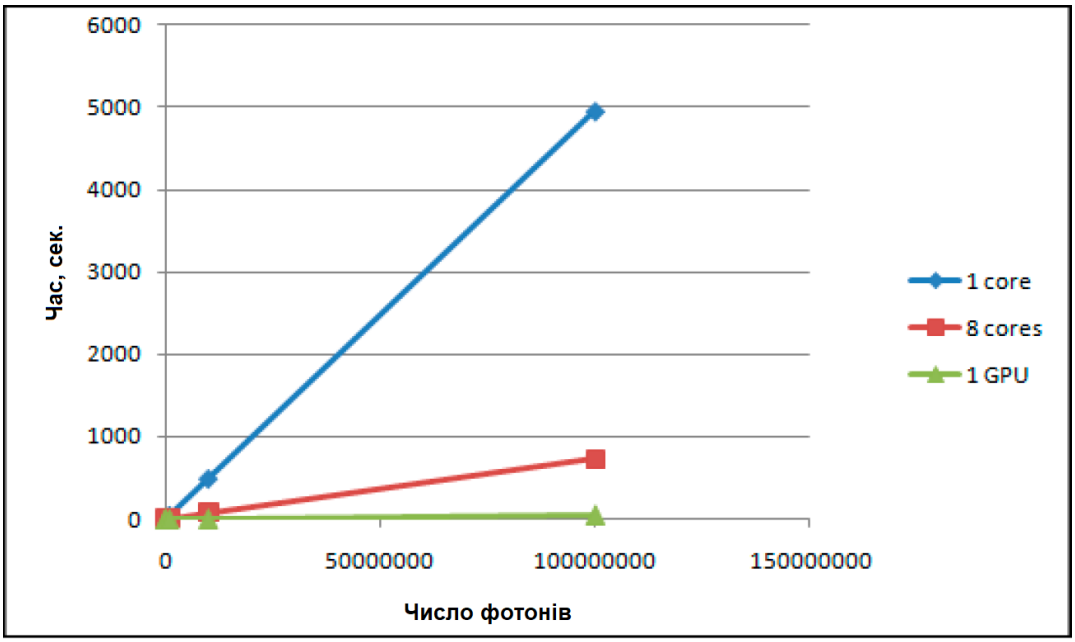


Рисунок 1.19. Період роботи методу у залежності з значення фотонів та обчислювального пристрою

Таблиця 1.2. Період роботи методу (у секундах) у залежності з значення використовуваних ГПУ, значення фотонів дорівнює 10^8

Обчислювальний пристрій	*Значення використовуваних ГПУ*			
	1	*2*	*3*	*4*
Tesla C1060	50.73	25.76	18.14	12.84

Значення проведених експериментів (табл. 1.1 та рис. 1.19) показують, яке впровадження графічного адаптера та моделі CUDA істотно підвищує продуктивність програми, але через це є обґрунтованим, але впровадження кількох ГПУ задля обробки даних (табл. 1.2 та рис. 1.20) передбачає додатково зменшити період, необхідний задля реалізації програми.

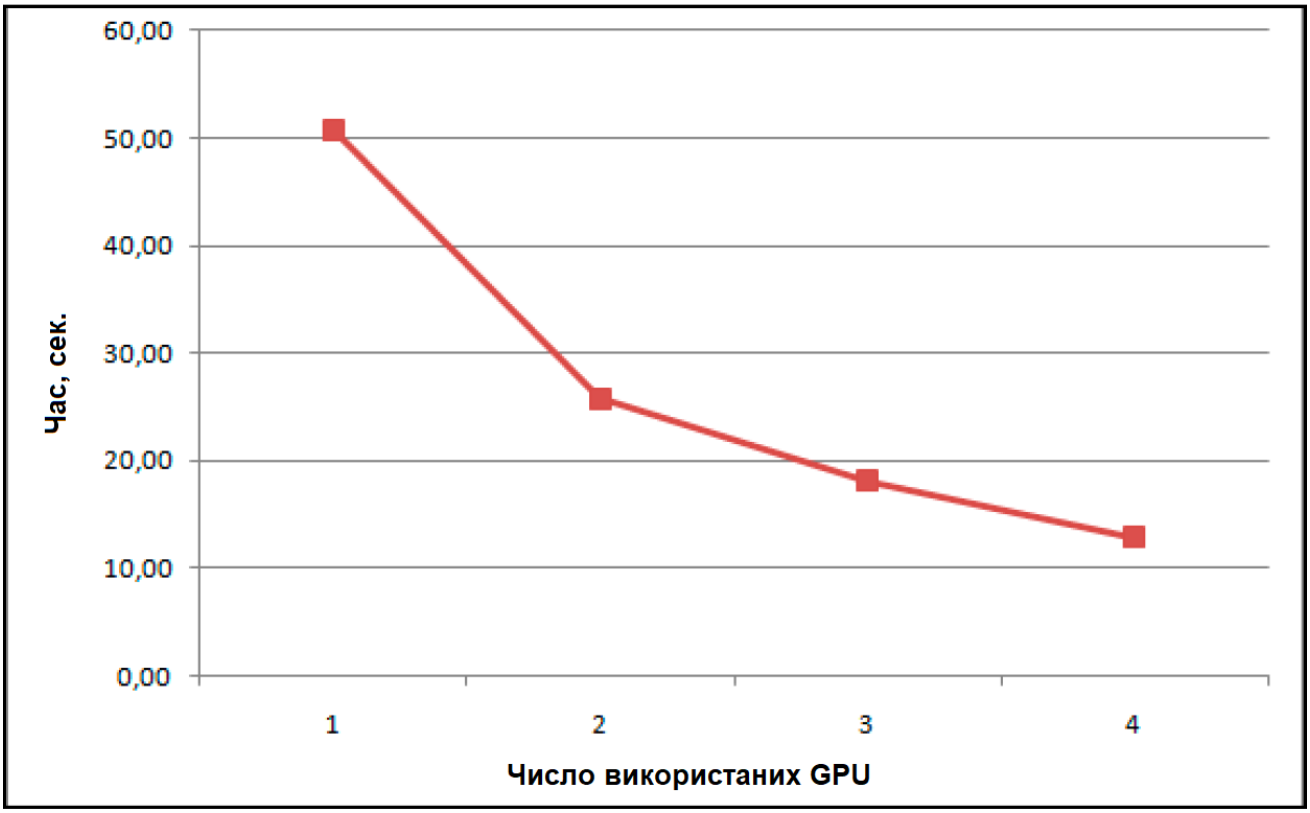


Рисунок 1.20. Період роботи методу у залежності з значення використаних ГПУ, значення фотонів дорівнює 10^8

Проведені експерименти показують добру масштабованість задачі при одночасній роботі на кількох графічних адаптерах.

Удосконалення розробленого програмного коду задля обчислення поширення променя у середовищах зі складною геометрією дозволить у перспективі ефективно застосовувати його як задля моделювання роботи різноманітних систем оптичної біомедицинської діагностики, так та задля планування променевої терапії.

1.13 Здійснення методу Монте-Карло на N-VIDIA CUDA задля завдань оптичної біомедицинської діагностики

Останнім часом методи оптичної біомедицинської діагностики отримали широкий розвиток, та багато із них переходять із класу експериментальних у значення застосовуваних у клінічній практиці. Серед переваг методів оптичної діагностики та візуалізації слід зазначити їх неінвазивність, безпечність та економічність (у зрівнянні із традиційними рентгенівськими методами та магнітно-резонансною томографією), та висока просторова роздільна здатність

(у зрівнянні із методами УЗД). Ряд методів оптичної діагностики при застосуванні вимагають відновлення тривимірного відтворення поза результатами реєстрації розсіяного об'єктом променя, але разом з цим інтерпретації отриманих картинок. Інтегро-диференційне рівняння переносу променя, яке описує поширення променя у розсіюючих середовищах є достатньо складним, та у загальному випадку не має аналітичного рішення, яке вимагає впровадження різноманітних наближень або його рішення чисельними методами.

Одним із ефективних методів чисельного моделювання поширення променя є метод Монте-Карло (ММК) [1]. Стосовно до даної задачі він полягає у багаторазовому повторенні обчислення випадкової траєкторії руху фотона у середовищі виходячи із заданих параметрів середовища (геометрії та оптичних властивостей) та подальшому статистичному аналізі отриманих даних. Перевагою даного методу перед різними аналітичними приближеннями є можливість обчислення інтенсивності розсіяного променя як поблизу джерела, так та на великих відстанях при будь-якої заданої геометрії середовища, у той період як область впровадження аналітичних наближень істотно обмежена.

Єдиним недоліком методу Монте-Карло є його ресурсовитратність (так, задля досягнення задовільної точності при моделюванні поширення променя у зразку біотканини товщиною близько 1 см зазвичай треба близько 10^8 випадкових траєкторій фотонів). Однак це обмеження може існувати ефективно подолано поза допомогою впровадження графічних обчислювачів, яке дозволяють здійснювати багатопотокові паралельні визначення [2].

Адже задля визначення випадкові траєкторій фотонів є незалежними одна з одної та визначаються оптичними і геометричними характеристиками об'єкта, їх розрахунок може здійснюватися паралельно без обміну даними поміж потоками. Цей факт робить реалізацію ММК із застосуванням графічних обчислювачів сильно вигідною.

В даній роботі метод Монте-Карло (ММК) застосовується задля:

- Обчислення характеристик світлового поля на довжинах

хвиль збудження та флуоресценції в задачі дифузійної флуоресцентної томографії (ДФТ);

- Моделювання картинок біотканин, одержуваних методом оптичної когерентної томографії;
- Обчислення максимальної глибини візуалізації структури біотканин методом багатофотонної флуоресцентної мікроскопії.

Дифузійна флуоресцентна томографія є сучасним оптичним методом візуалізації внутрішньої структури оптично-неоднорідних середовищ та модулів із використанням флуоресцентних маркерів [3]. Флуоресцентні маркери різної природи використовуються задля діагностики онкологічних захворювань, досліджень молекулярних процесів, типових задля канцерогенезу, дослідження метастазування та відповіді на протипухлинну терапію [4]. Задля точного визначення місця розташування та реальних розмірів флуоресціюючої пухлини, яке знаходиться в лабораторній тварині на великій глибині (глибше шкірного покриву), використовують методи ДФТ [5]. В методі ДФТ опромінення досліджуваного об'єкта відбувається на довжині хвилі збудження флуоресціюючих речовин (маркерів), але детектування сигналу – в спектрі їх флуоресценції. Задля реконструкції розподілу флуорофору в тканини у даний період використовуються спеціальні методи, яке враховують дифузне поширення світла [6,7]. Метод Монте-Карло стосовно ДФТ може застосовуватися задля вирішення наступних завдань: розрахунок поширення світла всередині об'єкта зі «складною» геометрією; проведення чисельних експериментів (моделювання результатів експерименту) задля апробації методів відновлення; відновлення оптичних властивостей середовища (коефіцієнтів поглинання, розсіювання, фактору анізотропії). Оптична когерентна томографія (ОКТ), заснована на принципах низькокогерентної інтерферометрії [8], передбачає отримувати відтворення поверхневих шарів біотканин на глибинах до 2 мм із вищою порівняно із традиційними (УЗД, рентген) методами просторовим роздільною здатністю (до одиниць мікрон). Формування ОКТ-картинок засноване на візуалізації дво- або тривимірного розподілу оптичних

властивостей всередині біоткани, яке передбачає реєструвати не тільки морфологічні, але та функціональні зміни в біотканинах, пов'язані із локальною зміною їх оптичних властивостей. Метод Монте-Карло застосовується задля моделювання ОКТ-картинок різноманітних модулів [9], яке передбачає інтерпретувати експериментальні діагностичні значення.

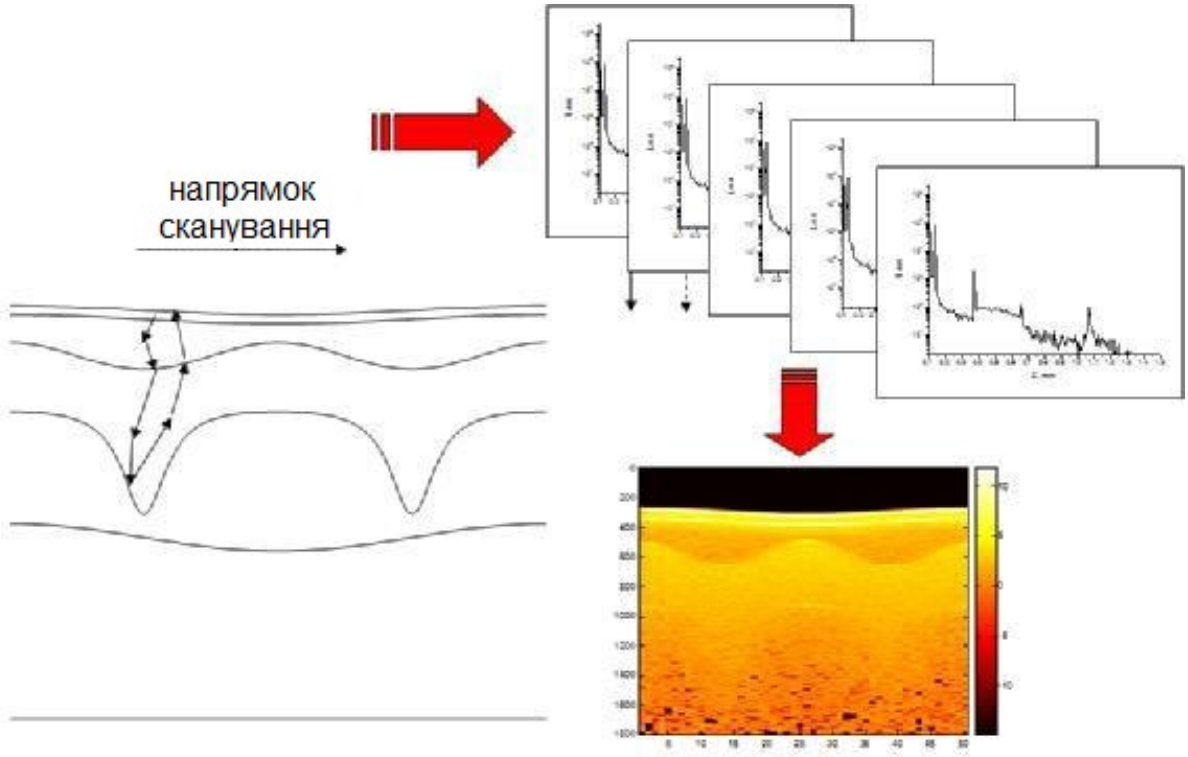


Рисунок 1.21. Формування ОКТ-картинок

Двохфотонна мікроскопія (ДФМ) – вид лазерної скануючої мікроскопії, яке використовує двохфотонно-збуджуєму флуоресценцію задля візуалізації внутрішніх структур біологічних тканин [10]. МК моделювання стосовно ДФМ застосовувалося задля обчислення поширення сфокусованих гаусових пучків у розсіюючих середовищах. Такі розрахунки використовувалися задля перевірки знов розроблених аналітичних моделей розсіювання пучка у розсіюючих середовищах, але разом з цим задля оцінки ефективності збудження сигналу двохфотонної флуоресценції у залежності з глибини фокусування променя накачування [11]. Нижче представлені характерні часи, необхідні задля проведення обчислення методу Монте-Карло на ГПУ (N-vidia Ge-force 470) із використанням моделі CUDA та ЦП (AMD Phenom II x4 920, 2.7 ГГц, однопотокова здійснення).

Таблиця 1.3. Порівняння часу обчислення при реалізації ММК на ГПУ та ЦП

Кількість траєкторій	*Період обчислення на ГПУ, сек.*	*Період обчислення на ЦП, сек.*
10^7	1.2	354
10^8	10.2	3614
10^9	101.8	36895

Особливістю реалізації методу є те, яке при використанні моделі CUDA поза один виклик ядра ГПУ прораховується близько $6 \cdot 10^6$ різноманітних випадкових траєкторій фотонів. Таким чином, було показано, яке здійснення методу Монте-Карло із використанням моделі CUDA передбачає істотно прискорити визначення, яке відкриває можливості задля більш ефективного впровадження цього методу у задачах біомедичної оптичної діагностики.

1.14 Впровадження N-VIDIA CUDA задля рішення завдань комп'ютерного зіру

Комп'ютерний зір є однією із областей, у котрих можливо ефективно застосовувати ГПУ задля широкого кола актуальних завдань, включаючи детектування осіб та пішоходів, відновлення тривимірних поверхонь, розпізнавання модулів задля робототехніки і візуальної одометрії. Багато методів такого роду можливо ефективно розпаралелювати поза даними та виконувати на ГПУ із високою швидкістю.

Область комп'ютерного зіру достатньо велика, в ній діє велика кількість наукових та промислових колективів, яке вирішують найрізноманітніші задачі на різноманітних рівнях. Ключовим фактором, яке визначає те, наскільки активно будуть використовуватися графічні процесори у майбутньому, є наявність зручних програмних інструментів. Необхідна програмна інфраструктура, на основі якої створювалися б нові розробки у області комп'ютерного зіру. У якості такого інструменту пропонується бібліотека Open-cv із підтримкою ГПУ.

Раніше вже робилися спроби застосовувати відеокарти задля вирішення

завдань у області комп'ютерного зіру. Прикладом спроможні служити бібліотеки CUVI Lib [2], GPU4VISION [3], GPUCV [4], openVidia [5], NPP [6]. Крім того, існує певна кількість незалежних реалізацій окремих методів. Здебільшого ці розробки ведуться ентузіастами, ресурси котрих обмежені. Багатьом із них доводиться виконувати дублюючу роботу. Через це існує потреба у професійному інструменті, який об'єднав би у собі кращі у своєму класі методи. Open-cv – це кросплатформена бібліотека методів комп'ютерного зіру, яке є стандартом де-факто у індустрії [7]. В 2010 році компанією Itseez поза підтримки N-vidia виконано проект розробки ГПУ-модуля задля Open-cv. Цей модуль являє собою набір процедур та класів, яке дозволяють фахівцям у області комп'ютерного зіру швидко почати застосовувати можливості відеокарт.

1.14.1 Підтримка ГПУ у Open-cv

ГПУ-модуль Open-cv містить у собі кілька рівнів функціональності. На нижньому рівні знаходяться реалізації службових операцій, таких як ініціалізація та управління ГПУ, робота із пам'яттю, механізм синхронних викликів. Рівнем вище реалізовано широкий набір базових процедур обробки картинок: різні методи фільтрації, пошук максимуму, афінні перетворення, визначення різниці картинок у різноманітних нормах та т.д. Список цих методів постійно оновлюється, у цілому вони покликані полегшити розробку методів комп'ютерного зіру найвищого рівня. До останніх можливо віднести кілька існуючих у Open-cv реалізацій:

- Стереозір:
    - BlockMatching [8];
    - BeliefPropagation [9];
    - Constant Space BeliefPropagation [10];
- Histograms of Oriented Gradients та лінійний SVM [11] – детектування пішоходів та інших модулів;
- Viola-Jonescascadeclassifier [12] – детектування людських обличь;
- Speed-uprobustfeatures (SURF) [13] – пошук ключових точок на зображенні задля склеювання картинок, розпізнавання модулів та т.д.

У Open-cv контейнером задля зберігання даних (у через це числі картинок) служить клас *Mat.* Поза аналогією, у ГПУ-модулі реалізований клас *GpuMat* практично із тією ж функціональністю, але зберігаючий значення в відеопам'яті. Бібліотека Open-cv надає зручні засоби обміну інформацією поміж ГПУ та ЦП.

Основну частину ГПУ-модуля складають процедури, котрі мають інтерфейс, ідентичний ЦП-частині Open-cv із тією тільки відмінністю, яке вони приймають на вхід об'єкт типу GpuMat.

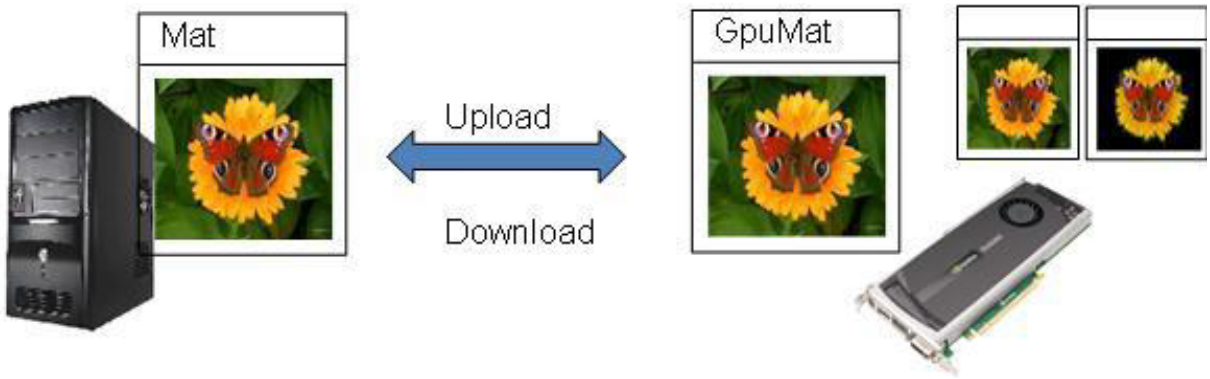


Рисунок 1.22. Зберігання картинок на відеокарті

Програмний інтерфейс модуля (API) створений максимально близьким до інтерфейсу ЦП-частини Open-cv, яке полегшує перенесення існуючого коду на ГПУ. Завантаживши відтворення на відеокарту, можливо застосовувати різні процедури обробки, котрі будуть проводитися цілком на відеокарті.

Нижче наводиться приклад коду, який отримує відтворення із камери та запускає на ньому метод пошуку пішоходів. У якості методу використовується HistogramsofOrientedGradients (HOG) [11]. Відмінності поміж ГПУ та ЦП версіями виділені жирним шрифтом:

```
Mat frame;
VideoCapturecapture(camera);
cv::HOGDescriptor hog;
hog.setSVMDetector(cv::HOGDescriptor::
getDefaultPeopleDetectorector());
capture>> frame;


vector<Rect> found;
hog.detectMultiScale(frame, found,
    1.4,  Size(8, 8), Size(0, 0), 1.05, 8);
```

```
Mat frame;
VideoCapturecapture(camera);
cv::gpu::HOGDescriptor hog;
hog.setSVMDetector(cv::HOGDescriptor::
getDefaultPeopleDetectorector());
capture>> frame;

GpuMatgpu_frame;
gpu_frame.upload(frame);
vector<Rect> found;
hog.detectMultiScale(gpu_frame, found,
    1.4, Size(8, 8), Size(0, 0), 1.05, 8);
```

Рисунок. 1.23. Приклад вихідного коду задля методу пошуку пішоходів

Задля тестів ефективності використовувалася наступна конфігурація: 4-ядерний Intel Core i5-750 2.66ГГц та NVidiaGe-force GTX470 с 448 ядрами на архітектурі Fermi. Отримані ускорення на ГПУ в зрівнянні із ЦП-версіями представлені у таблиці нижче.

Таблиця 1.4. Ускорення на ГПУ в зрівнянні із ЦП

Stereo Block Matching (BM)	7x
Stereo Belief Propagation (BP)	10-20x
Stereo Constant Space BP (CSBP)	50-100x
HOG	3.5-7x
SURF	12x
CascadeClassifier	6-10x

При зрівнянні використовувалися ГПУ- та ЦП- версії методів із Open-cv, крім BP та CSBP, задля котрих порівняння велося із авторським кодом. Open-cv містить реалізації методів, котрі тривалий період оптимізувались професійними розробниками, яке забезпечує коректність порівняння. Бібліотека Open-cv була зібрана із підтримкою багатопоточності (на основі Intel TBB), через це усі ядра процесора були задіяні.

Слід зазначити, яке приріст ефективності базових процедур становить у середньому ~ 50-х. Пояснення цьому полягає у через це, яке прості методи спроможні існувати більш ефективно розпаралелені поза даними. Задля отримання докладної інформації про швидкість роботи примітивів та декотрих процедур високого рівня, слід запустити додаток performance_гпу із набору прикладів Open-cv.

Відомо, яке не усі визначення спроможні існувати успішно розпаралелені. Існують чисто послідовні методи, ефективна здійснення котрих на ГПУ неможлива. Прикладом може служити фільтрація малих пов'язаних компонент на зображенні (speckle filtering). Подібні операції доводиться виконувати на центральному процесорі. Однак, у цьому випадку виникають накладні витрати при передачі даних поміж ЦП та ГПУ. У декотрих випадках, частина накладних витрат може існувати сильно істотною.

Задля мінімізації подібних витрат у Open-cv пропонується механізм

асинхронних викликів. При використанні асинхронної процедури вона повертає керування відразу, у той період як результат може існувати готовий тільки через деякий період. Щоб дізнатися про готовність результату слід застосовувати клас Stream. Подібний засоби дозволяють достатньо просто застосовувати ГПУ та ЦП одночасно. Можливість впровадження кількох відеокарт разом з цим є достатньо затребуваною. Однак, у Open-cv реалізації методів орієнтовані на впровадження одного ГПУ. Щоб застосовувати кілька ГПУ, користувач повинен сам розподіляти роботу поміж ними, використовуючи задля цих цілей клас MultiGpuManager. Коли є велика база картинок, то легко можливо розбити всю роботу поза кількома відеокартами. В разі ж одного відтворення розпаралелювання роботи не завжди можливе. Майже задля усіх примітивних процедур період роботи методу достатньо малий та накладні витрати на пересилку даних становитимуть суттєву частину часу, яке робить таке розпаралелювання неефективним. В разі високорівневих методів застосовувати кілька ГПУ задля роботи над одним зображенням є цілком допустимим, адже період обробки набагато більше часу пересилання. Наприклад, задля методу Stereo Block Matching можливо розрізати картинку навпіл із невеликим перекриттям, обчислити результат на кожному ГПУ незалежно, але потім склеїти значення. Продуктивність задля двох ГПУ поза результатами запусків становить 180% з варіанта із одним ГПУ.

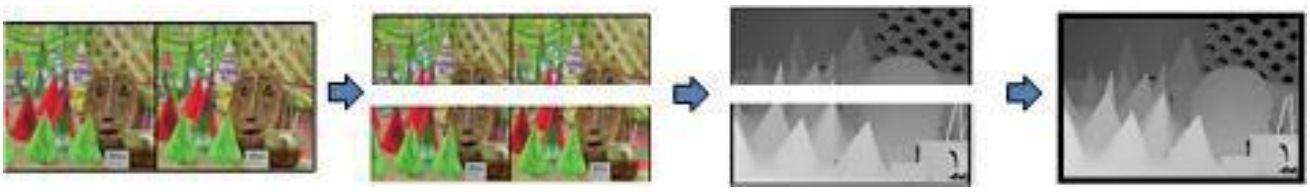


Рисунок 1.24. Впровадження двох ГПУ у алгоритмі стереозору

Ще один приклад – задача детектування пішоходів. Адже люди на кадрі спроможні існувати різного розміру у залежності з відстані до камери, метод запускається на різноманітних масштабах відтворення. Кожний масштаб обробляється незалежно, після чого значення об'єднуються. Таким чином, обробку набору масштабів можливо розділити поміж декількома ГПУ, тоді отримане ускорення пропорційне значення використовуваних ГПУ.

1.14.2 Впровадження ГПУ в задачі стереозору

Задача стерео-зіставлення полягає в пошуку відповідностей поміж точками на двох зображеннях (лівого та правого), отриманих поза допомогою двох камер або однієї стереокамери. Знайшовши відповідність задля будь-якого пікселя в вигляді вектора зміщення, можливо обчислити відстань до модулів на сцені, поза умови, яке відомо взаємне розташування камер та їх параметри. Задача пошуку пари задля точки лівого відтворення може існувати зведена до задачі пошуку уздовж горизонтальної прямої на правому зображенні. Тоді задля будь-якого пікселя достатньо знати тільки величину зсуву вздовж осі абсцис (disparity). Оптимальна відповідність поміж точками визначається на основі порівняння околиць пікселів лівого та правого картинок. Класичним методом вирішення даної задачі є метод Stereo Block Matching (block-matching). Даний метод достатньо добре вивчений, детальна інформація може існувати знайдена у роботах [8], [14].

Рисунок 1.25. Впровадження кількох ГПУ у алгоритмі детектування пішоходів

Важливо відзначити, яке метод block-matching був вже неодноразово портованим на ГПУ [11] [12] [13] [14]. Дана здійснення ґрунтується на одній із ранніх робіт та являє собою модифіковану версію методу, описаного у [13]. Деталі та загальний опис методу наводяться у оригінальному джерелі, через це тут у основному будуть розглянуті відмінності з оригінальної версії. Головними удосконаленнями є такі моменти: більш ефективне визначення кореляційних сум, забезпечення кеш-ефективності, фільтрація низькотекстурованих областей. Незначні удосконалення, такі як більш економічний формат зберігання, у даній статті не описуються, проте спроможні існувати знайдені в вихідних кодах бібліотеки Open-cv. Далі розглянемо кожне із удосконалень окремо.

Визначення кореляційних сум – це центральна частина методу, метою якої є визначення схожості двох модулів пікселів. У оригінальній версії методу визначення, вироблені різними потоками, значно перекривалися. Фактично, сусідні потоки на ГПУ підсумовували елементи двох однакових векторів, яке відрізняються тільки одним елементом. В запропонованій реалізації потоки ділять вектор на кілька частин, та кожний підсумовує тільки свою ділянку, але частину, яке залишилася, отримує в інших стрімів.

Метод пошуку відповідностей на ГПУ влаштований таким чином, яке блоки стрімів читають області відтворення, сильно пересічні один із одним. Дана обставина наводить на думку про те, яке повторні читання повинні проводитися вже із кеша L1 на ГПУ. Однак, згідно із показаннями CUDAVisualProfiler, доступ до глобальної сховища даних був головним вузьким місцем описаної реалізації. Подібна ситуація пояснюється тим, яке ми змушені зберігати у кеші одночасно кілька рядків відтворення. Це призводить до того, яке при додаванні нових рядків раніше прочитані виявляються викинутими із кеша. Таким чином, перехід до нових ітерацій у циклі поза значенням зміщення (d) означає повторне читання рядків відтворення із глобальної сховища даних. Тривіальні розрахунки підтверджують цю гіпотезу (Рис. 1.26). Розмір частини карти зсувів, яке обчислюється блоком стрімів, дорівнює blockWidth x RowPerThread = 128x21. Тоді задля визначення кореляційних сум пікселів цієї частини, розмір шматків вхідних картинок, необхідних при кожному значенні лічильника циклу по d, дорівнює:

*(blockWidth + winSize - 1) * RowPerThread * 2 * 8 blocks permultiprocessor = = (128 + 18) * 21 * 2 * 8 = 49kb> 48kb of cache per multiprocessor*

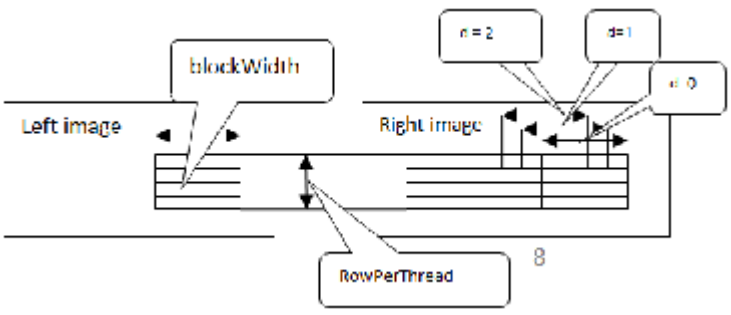


Рисунок 1.26. Обчислювальна схема методу

Значення розрахунки наведені задля відеокарти N-vidia Ge-force GTX470, побудованої на архітектурі Fermi. Задля поліпшення кеш-ефективності було використано розгортання циклу по зсувах, у якому визначається оптимальна відповідність пікселів лівого та правого картинок. Була використана глибина розгортки, рівна 8, таким чином, відбувався одночасних розрахунок 8 кореляційних сум, після чого вибиралося найкраще значення та відбувалося порівняння із рекордом, яке зберігається у глобальній сховища даних. Глибина розгортки 8 була отримана експериментально, її зменшення погіршує кеш-ефективність, але отже продуктивність. Збільшення ж веде до зниження значення запускаємих модулів через брак реєстрової сховища даних. Природно, яке обрані значення спроможні існувати не оптимальні задля інших моделей ГПУ.

Метод StereoBlockMatching працює найкраще на сценах, у котрих об'єкти мають текстуру, тобто містять безліч дрібних деталей. Деталізація та унікальність малюнка передбачає краще зіставляти пікселі лівого та правого кадрів. У областях, де текстури мало, можливо бачити часті помилки методу. Так, наприклад, проблеми виникають при визначенні відстані до однорідних стін (рис. 1.27). Ідея методу фільтрації достатньо проста: задля околиці будь-якого пікселя відтворення визначається міра її текстурування. Задля цього до відтворення застосовується згортка із ядром:

$$\begin{matrix} -2 & 0 & 2 \\ -1 & 0 & 1 \\ -2 & 0 & 2 \end{matrix}$$

Даний оператор є похідною по X. Потім відбувається підсумовування цих похідних у околиці будь-якого пікселя.

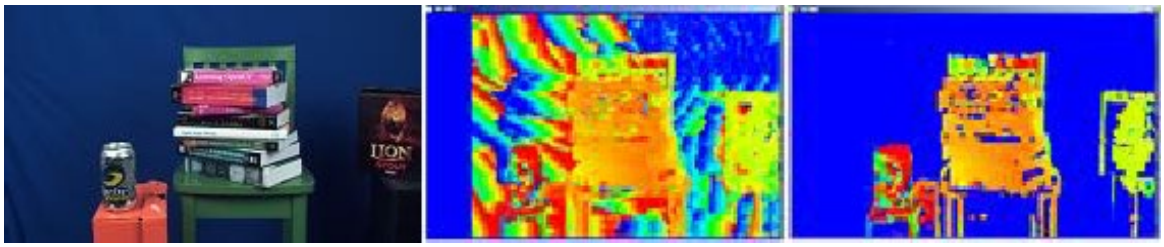


Рисунок 1.27. Фільтрація областей зі слабкою текстурою

Після цього, коли сумарне значення менше, ніж заданий поріг, то область

вважається низькотекстурованою та величина зсуву виставляється у «не визначене». Задля ефективного визначення міри текстурування тут використовується ідея сум, яке біжать по Y.

Фінальна продуктивність методу StereoBlockMatching склала 18 кадрів у секунду на відеокарті Ge-force GTX470 (Fermi) та роздільній здатності FullHD (1920x1080) задля діапазону пошуку зміщення 0..256. Це приблизно у 7 разів швидше оптимізованої ЦП-версії методу на 4-ядерних IntelCore i5-750 2.66Ghz. На сьогоднішній день це найшвидша здійснення методу block-matching із відомих автору.

Підводячи підсумки, ще раз зазначимо, яке впровадження графічних обчислювачів добре підходить задля завдань комп'ютерного зіру, дозволяючи значно прискорити існуючі методи та розробляти нові, орієнтовані на масивно-паралельні архітектури. Розглянутий модуль достатньо простий в використанні, передбачає отримувати приріст ефективності у 50-100 разів задля примітивних процедур, та у 5-10 разів задля складних високорівневих методів.

Важливо відзначити, яке бібліотека Open-cv поширюється по вільній ліцензії типу BSD, через це кожний може ознайомитися та скористатися результатами даної роботи.

1.15 Здійснення на CUDA математичного методу

В даному підрозділі розглянуто побудову і впровадження програми задля мережі одночасних розрахунків на базі моделі CUDA із структурою, представленою на рис 1.28.

Задля демонстрації обчислювальних можливостей було обрано математичний вираз $MX = e * MC * MZ * MK + max(R) * MT$. Програма була складена на мові програмування C із використанням засобів взаємодії стрімів, але саме бар’єрів і атомарних операцій.

1.15.1 Опис паралельного математичного методу

Паралельний математичний метод можливо представити в вигляді трьох етапів:

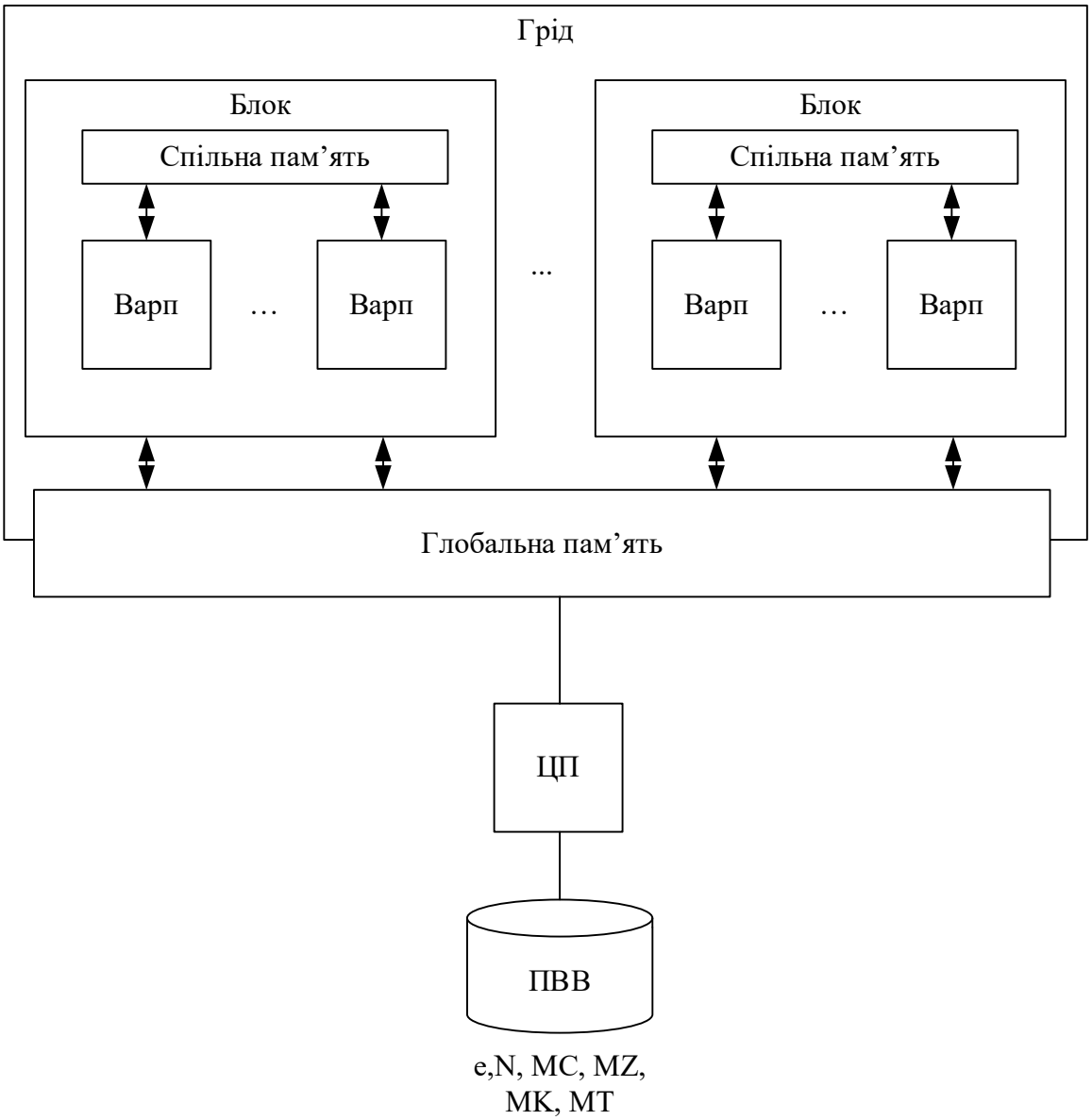


Рисунок 1.28. Структура паралельної комп’ютерної системи (ПКС)

1. $m_i = \max(R_H), i = \overline{1,P}$
2. $m = \max(m, m_i), i = \overline{1,P}$
3. $MX_H = e * (MC_H * MZ * MK) + m * MT_H$

де

- e, m, MZ, MK – спільні ресурси;
- P – кількість стрімів;
- N – розмірність векторів і матриць;
- H = N/P;
- R_H – H елементів вектору R;
- MC_H – рядків матриці MC;

- MT_H – рядків матриці MT;
- MX_H – рядків матриці MX;

1.15.2 Здійснення методу взаємодії стрімів

Адже розроблювана програма є масштабованою та працює на системі із кількістю обчислювачів P≥2, то задля реалізації було складено єдиний метод задля усіх завдань, який зображено в табл 1.5. Разом з цим необхідно враховувати особливість роботи із CUDA-потоками, адже базовий спосіб бар'єрної синхронізації працює тільки задля стрімів із одного блоку, через це із метою можливості масштабування необхідно додатково додати синхронізацію стрімів із різноманітних модулів.

Таблиця 1.5. Метод взаємодії стрімів

Назва	*Бар'єри, КД*
1. Введення e, R, MC, MZ, MK, MT	
2. Передача e, R, MC, MZ, MK, MT в сховище даних	
3. Коли tid = 1, надіслати сигнал про введення e, R, MC, MZ, MK, MT	$S_{2\text{-}P\ -\ 1}$
4. Коли tid != 1, очікувати сигнал про введення e, R, MC, MZ, MK, MT	$W_{2\text{-}P\ -\ 1}$
5. Визначення $m_i = max(R_H)$	
6. Визначення $m=max(m, m_i)$	
7. Надіслати сигнал про обрахування m	$S_{1\text{-}(tid\text{-}1),(tid+1)\text{-}P\ -\ tid}$
8. Очікувати сигнал про обрахування m	$W_{1\text{-}(tid\text{-}1),(tid+1)\text{-}P\ -\ tid}$
9. Копіювати $e_i = e$, $m_i = m$, $MZ_i = MZ$, $MK_i = MK$	КД
10. Визначення $MX_H=e_i*(MC_H*MZ_i*MK_i) + m_i*MT_H$	
11. Коли tid != 1, надіслати сигнал про закінчення	$S_{1\ -\ 2\text{-}P}$
12. Коли tid = 1, очікувати сигнали про закінчення MX_H	$W_{2\text{-}P\ -\ 1}$
13. Коли tid = 1, передати результат в ОЗП	
14. Виведення результату MX	

В ході аналізу було обрано спосіб із використанням системи прапорців в глобальній сховища даних. В схемі взаємодії стрімів застосовано наступні методи:

- atomicCopy() – атомарна операція задля копіювання спільних ресурсів із глобальної сховища даних;

- atomicMax() – атомарна операція, котра забезпечує послідовний доступ до змінної m при записі результатів знаходження елементу;
- __syncthread() – функція синхронізації стрімів.

Задля реалізації паралельного реалізації програми було використано:

- метод main(), який викликає реалізації одночасних стрімів;
- фукнція vectMax(), котра виконує пошук максимального вектору;
- фукнція matrSum(), котра виконує складання матриць;
- функція scalOnMatr(), котра виконує множення значення на матрицю;
- фукнція matrOnMatr(), котра виконує множення матриці на матрицю.

1.15.3 Тестування програми на паралельній обчислювальній системі

Задля тестування використовувалась паралельна обчислювальна система із наступними апаратними характеристиками:

- процесор Intel i5-8250U 1.6 Ггц із технологією Turbo Boost до 3.4 Ггц;
- графічний процесор N-vidia Ge-force GTX1660 із 6 Гб відеопам’яті типу GDDR5, частотою 1785 МГц і 192-бітною шиною даних;
- оперативна сховище даних 16 ГБ.

Використане програмне забезпечення:

- операційна система: Microsoft Windows 10;
- середовище розробки програми: Microsoft Visual Studio 2017 і CUDA Toolkit 10.1 Update 1.

Задля вимірювання часу реалізації програми використовувався базовий метод мови програмування C – clock() задля фіксації початку роботи програми та її завершення. Задля тестування обрано такі розмірності векторів та матриць: N = 896, N = 1792, N = 2688. Програму виконано із різним значенням значення ядр (32, 128, 256), але разом з цим було використано аналогічну послідовну програму в однопроцесорній системі.

Коефіцієнт ускорення K_n показує скорочення часу реалізації паралельної програми в паралельній системі із P процесорами T_p в зрівнянні із часом реалізації послідовної програми в однопроцесорній системі T_1:

$$K_п = T_1/T_P \quad (1.1)$$

Коефіцієнт ефективності K_e впровадження комп'ютерної системи показує ступінь впровадження P обчислювачів системи:

$$K_e = \frac{K_п}{P} * 100\% \quad (1.2)$$

Значення тестування та проведених досліджень ефективності розробленої програми наведено в таблицях 1.6 – 1.8.

Таблиця 1.6. Період реалізації програми

N	*T1, мс ЦП*	*ГПУ*		
		T2, мс, P=32	*T3, мс, P=128*	*T4, мс P=256*
896	6433	2948	1814	1291
1792	84170	41561	24734	18891
2688	224500	100418	71305	47053

На основі даних із таблиці 1.6 виконано розрахунок значень коефіцієнтів ускорення, котрі наведені в таблиці 1.7.

Таблиця 1.7. Значення K_n

N	*Кількість обчислювачів (P)*			
	ЦП	*ГПУ*		
	1	32	128	256
896	1	2,18	3,54	4,98
1792	1	2,02	3,40	4,45
2688	1	2,23	3,14	4,77

Коефіцієнти ефективності (таблиця 1.8) обчислено поза даними таблиці 1.2. і таблиці 1.3.

Таблиця 1.8. Значення K_e

N	*Кількість обчислювачів (P)*			
	ЦП	*ГПУ*		
	1	32	128	256
896	100	6,81	2,76	1,94
1792	100	6,31	2,65	1,74
2688	100	6,98	2,453	1,86

Використовуючи таблиці 1.7-1.8 побудовано графіки зміни коефіцієнтів ускорення та ефективності у залежності з N та P.

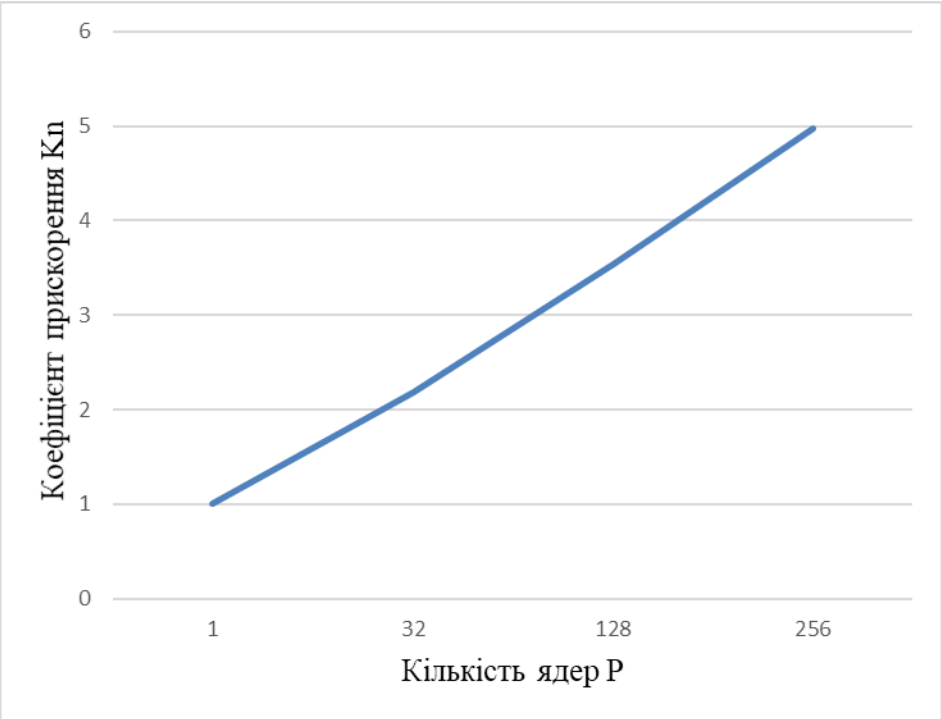


Рисунок 1.29. Графік зміни коефіцієнту ускорення програми у залежності з значення ядр при N=896

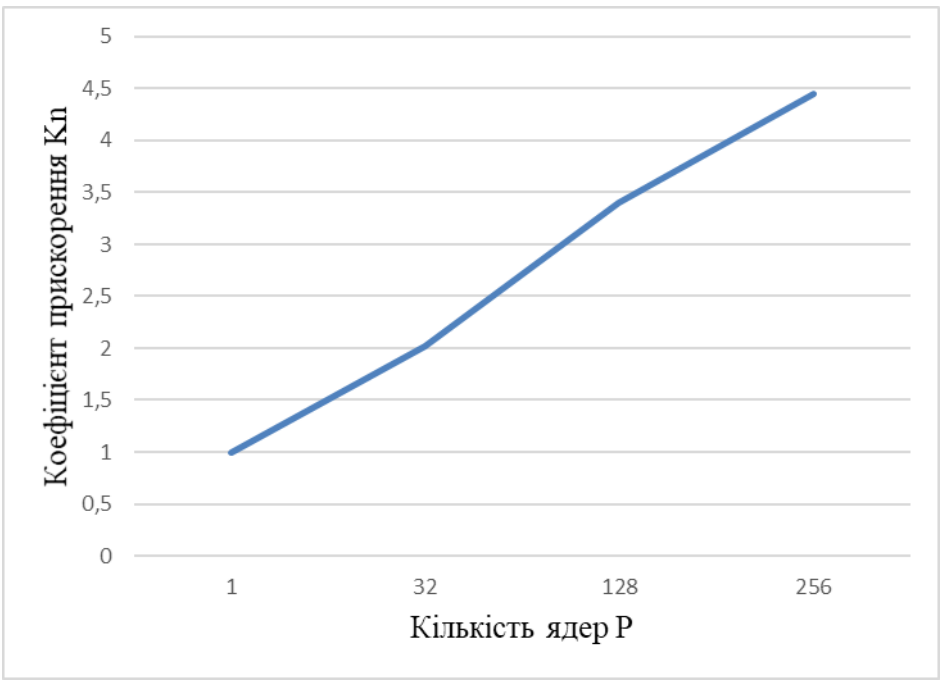


Рисунок 1.30. Графік зміни коефіцієнту ускорення програми у залежності з значення ядр при N=1792

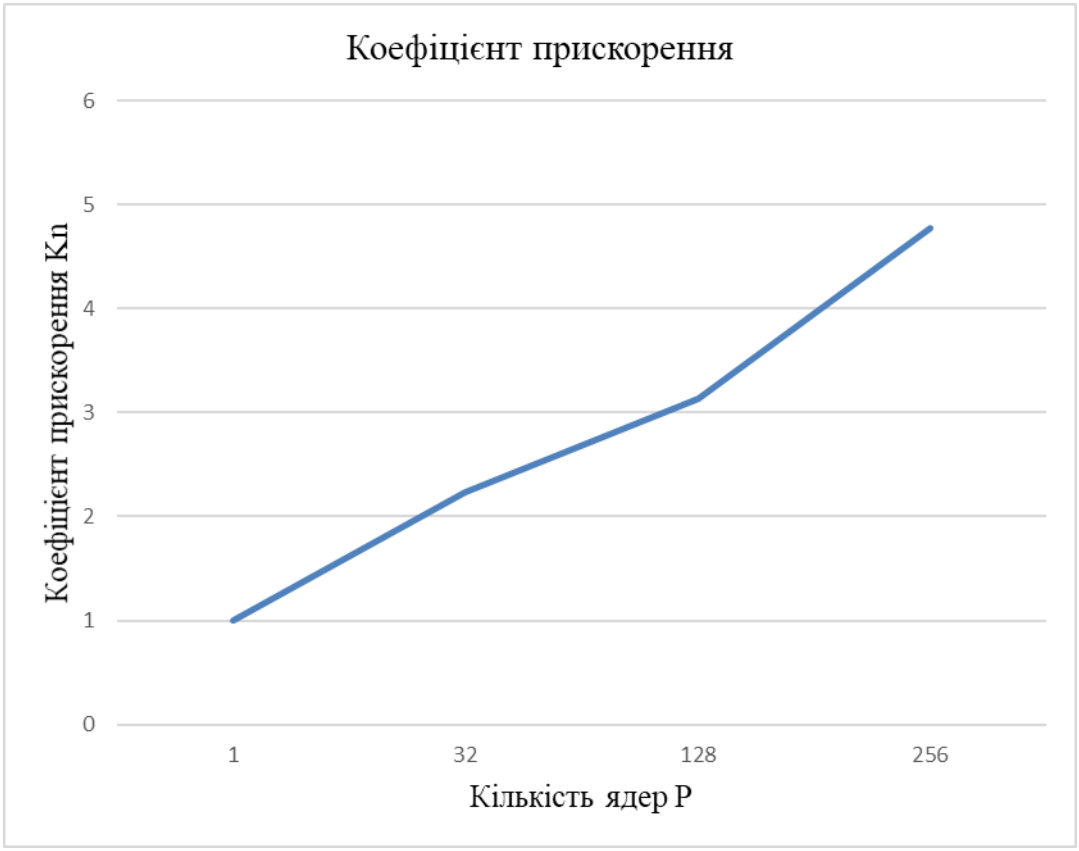


Рисунок 1.31. Графік зміни коефіцієнту ускорення програми у залежності з значення ядр при N=2688

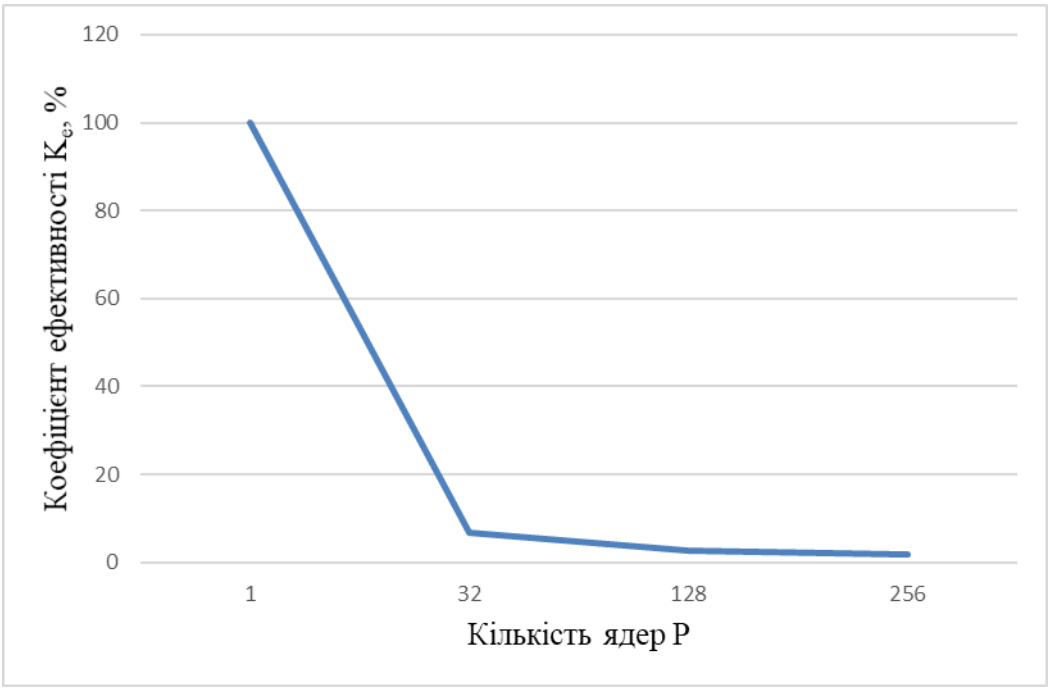


Рисунок 1.32. Графік зміни коефіцієнту ефективності програми у залежності з значення ядр при N=896

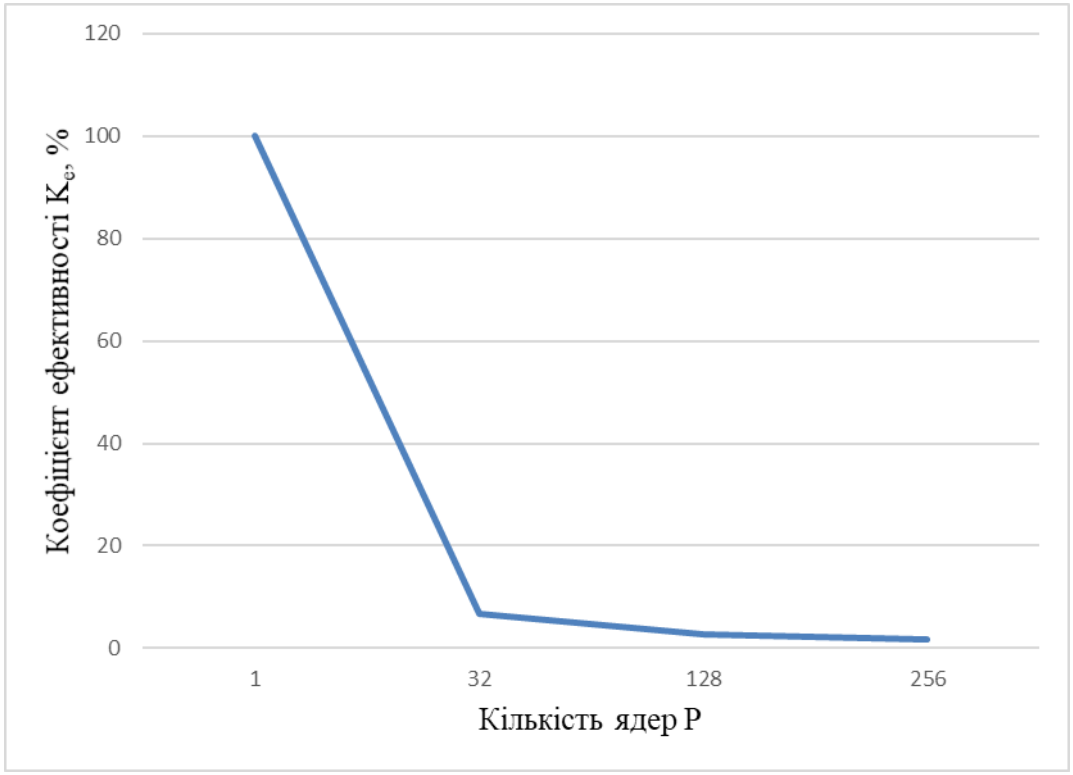


Рисунок 1.33. Графік зміни коефіцієнту ефективності програми у залежності з значення ядр при N=1792

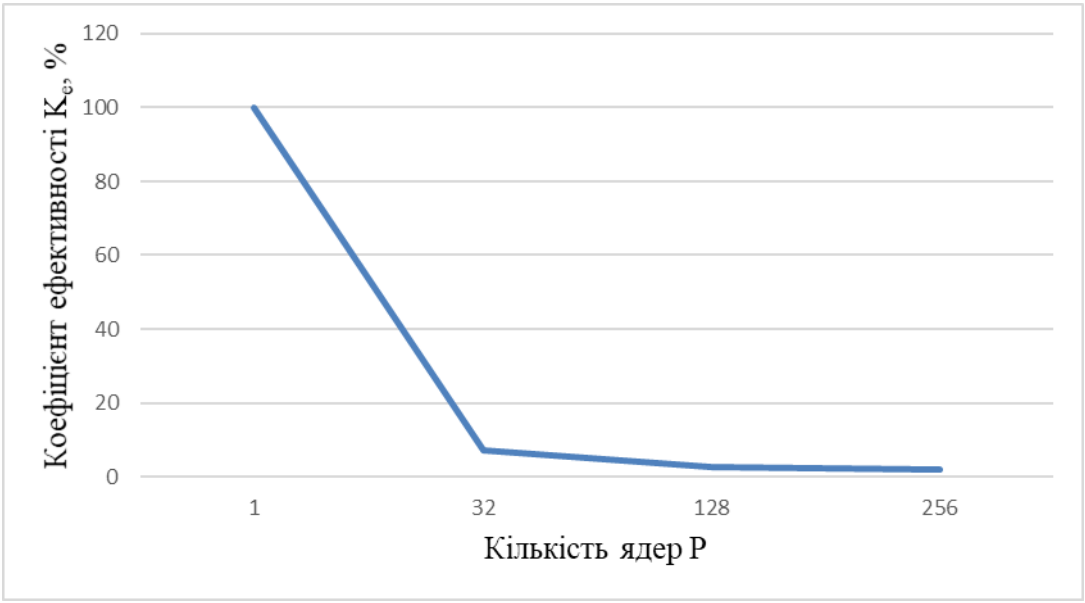


Рисунок 1.34. Графік зміни коефіцієнту ефективності програми у залежності з значення ядр при N=2688

Тестування програми показало наступне:

- впровадження моделі CUDA забезпечує скорочення часу визначення заданої математичної задачі при збільшенні P задля усіх N, як показано в

табл. 1.6. Значення K_n лежить у межах з 2.18 до 4.77 (табл. 1.7);

- максимальне значення K_n = 4.98 забезпечує ПКС із P = 256 і N = 896;
- мінімальне значення K_n = 2.02 виявлено в ПКС із P = 32 і N = 1792;
- значення K_e змінюються з 1.86 до 6.81%, як показано в таблиці 3.8;
- найефективніше програма використовує ПКС із P = 32 і N = 2688; при цьому K_n = 6,98%;
- найнижча ефективність впровадження ПКС програмою виявлена при P = 256, N = 1792.

Швидкість реалізації програми не зростала лінійно при збільшенні значення стрімів, адже даний метод не є повністю паралельним. Частина часу реалізації витрачалася на синхронізацію і доступ до спільних ресурсів. Коефіцієнти ефективності стрімів ГПУ виявилися достатньо низькими. При використанні ЦП вони були б значно більшими, але необхідно враховувати те, яке ядра графічного процесору не є повноцінними ядрами. Їх перевага полягає в великій значення та використанні добре розпаралелених задач.

2 РОЗДІЛ ОХОРОНИ ПРАЦІ І ТЕХНІКИ БЕЗПЕКИ

Безпека праці, як галузь практичної діяльності, спрямована на створення небезпечних та нешкідливих умов праці. На сучасному етапі розвитку виробництва вона набуває все більше важливого значення

У умовах сучасного виробництва, яке характеризується масовим характером і широким застосуванням комп'ютерної техніки, попередні пріоритети зазнали суттєвої трасформації. В центрі уваги вітчизняних і зарубіжних фахівців є питання щодо визначення характеру і умов праці користувачів комп'ютерів, функціональних змін в динаміці реалізації трудових завдань, захворюваності і стану здоров'я, розробки засобів захисту.

У розділі охорона праці дипломного проекту виконується аналіз ефективності впровадження моделі CUDA при рішенні математичних завдань.

1.Аналіз небезпечних та шкідливих факторів, яке впливають на програміста при розробці даного програмного комплексу.

Виробниче середовище у умовах сучасних методів господарювання характеризуються посиленим негативним впливом шкідливих і небезпечних чинників на гігієнічні показники й санітарний стан умов праці, але відтак та на організм людини. Подолання цього явища веде до зниження рівня професійних захворювань, до зміцнення здоров’я працюючих, яке забезпечується системою соціально-економічних заходів.

Всесвітня організація охорони здоров'я дійшла висновку, яке робота із використанням персональних комп'ютерів (ПЕОМ) супроводжується зоровим та нервово-емоційним напруженням, негативними зрушення-ми у кістково-м'язовій системі людини. Слабкі рівні неіонізуючих та іонізуючих випромінювань, яке створюються відеодисплейними терміналами на електрон-но-променевих трубках, несуть загрозу збільшення онкопаталогії, негативного впливу на вагітних та плід.

В усіх розвинених країнах, в через це числі у країнах Європейської спільно-ти, існують сотні документів, котрі регламентують вимоги не тільки до комп'ютерів, але та до організації робочих місць із їх використанням. Таким

чином, безконтрольне впровадження комп'ютерної техніки може призвести до негативного впливу на здоров'я користувачів комп'ютерів.

Аналіз умов праці, технологічних процесів, апаратури та обладнання проводиться із точки зіру можливості виникнення появи небезпечних факторів, виділення шкідливих виробничих речовин. На основі такого аналізу визначають-ся небезпечні ділянки виробництва, можливі аварійні ситуації, розробляються заходи щодо їх усунення або обмеження наслідків.

Вибір технічних засобів забезпечення безпеки повинен здійснюватися на основі вивчення особливостей будь-якого виявленого небезпечного й шкідливого виробничого фактора та зони його дії – так званої небезпечної зони.

Можливі небезпечні і шкідливі чинники:

- Психофізіологічні;
- напруження органу зіру;
- нервово-емоційне напруження;
- гіподинамія; вимушена робоча поза;
- напруження окремих групп м'язів передпліччя, кистей і пальців рук виникає внаслідок реалізації дрібних стереотипних рухів;
- монотонність праці.

Фізичні:

- змінні електромагнітні поля - рентгенівське, ультрафіолетове, оптичне, інфрачервоне,
- радіочастотне випромінювання;
- електростатичні поля;
- нераціональне освітлення - недостатня або (і) нерівномірна освітленість, відбита блискучість, нераціональний розподіл яскравості у полі зіру;
- шум від роботи принтерів (переважно матричних і струменевих), сканерів;
- вентиляторів охолодження комп'ютера);
- зміна іонного складу повітря;

– вібрація внаслідок роботи матричних і струменевих принтерів;
– пил паперу і тонеру, яке виникає при друкуванні на лазерних принтерах на низькосортному папері.

Хімічні:

– леткі хімічні речовини, яке виділяються в повітря робочої зони із полімерів і пластмас, із котрих виготовлений комп'ютер і інші периферичні пристрої:
– діоксин, фуран, поліхромовані біфеніли тощо;
– озон, яке виділяється під період роботи декотрих моделей моніторів і лазерних принтерів;
– хлоровмісні сполуки, яке виділяються при використанні задля друку на лазерних принтерах паперу, який відбілений хлором;
– леткі компоненти чорнил і тонерів, яке виділяються при друкуванні на струменевих і лазерних принтерах;

Біологічні: деякі види патогенних макроорганізмів, яке передаються контактним шляхом через клавіатуру і мишу різним користувачам одного комп'ютера.

Гігієнічна оцінка умов і характеру праці на робочих місцях проводиться відповідної до Державних санітарних норм і правил «Гігієнічна класифікація праці поза показниками шкідливості і небезпечності факторів виробничого середовища, важкості і напруженості трудового процесу», затверджених

Наказом МОЗ України з 08.04.2014р. №248 і залежить з фактично визначених рівнів впливу факторів виробничого середовища та трудового процесу і із урахуванням їх можливої шкідливої дії на здоров’я працівника.

2.1 Виробниче приміщення

Площа приміщення на одну людину по вимогам ДСанПіН 3.3.2-007-98 становить 6м квадратних, але об’єм 20м кубічних, у приміщенні є вікна, котрі орієнтовані на південь. Через це основні кольори інтер’єру – стіни світло-блакитного кольору; підлога-зеленого.

2.1.1 Вимоги до робочого середовища при роботі із ВДТ

Основними нормативними документами, яке регламентують параметри мікроклімату виробничих приміщень, є ДСН 3.3.6.042-99 і ГОСТ 12.1.005-88. Ці параметри нормуються задля робочої зони - визначеного простору, у якому знаходяться робочі місця. В нормативному документі ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» параметри мікроклімату повинні відповідати даним таблиці 2.1.

Таблиця 2.1. Санітарні норми мікроклімату виробничих приміщень

Період року	Категорія робіт	Температура, С		Відносить. Вологість, %
		оптимальна	допустима	
Холодний	Легка – Іа	22-24	21-25	40-60
Теплий	Легка – Іа	23-25	22-26	40-60

Іа – категорія легких фізичних робіт при котрих витрата енергії дорівнює 105-140Вт (90-120 ккал/год)

Задля створенная у приміщенні нормальних умов мікроклімату задля працівника та видалення шкідливих забруднень, була спроектована та належним чином встановлена вентиляційна система – загально обмінна, припливно-витяжна по нормам ДСТУ Б АЛЕ. 3.2-12:2009 ССБП.

Вентиляція створює на робочому місці, метеорологічні умови та чистоту повітряного середовища, яке відповідають чинним санітарним нормам. Разом із тим вентиляція забезпечує умови, яке відповідають вимогам технологічного процесу. Висока чи низька температура повітря у приміщенні із ПК негативно впливає на функціональний стан користувача. Недостатня вологість у приміщенні призводить до надмірного висихання слизових оболонок очей, носа, горла і до нагромадження зарядів статичної електрики, яке утворюються у процесі роботи ПК.

Разом із тим недопустима вологість повітря більше 75%. На робочих місцях користувачів ПК параметри мікроклімату мають відповідати вимогам температури не більше 22-24 градусів в холодну пору року, 23-25 – в теплу.

Задля подачі у приміщення повітря використовуються системи механічної

вентиляції та кондиціонування, але разом з цим природна вентиляція

В приміщенні де працює програміст параметри мікроклімату відповідають нормам із таблиці 2.1.

2.1.2 Освітлення виробничого приміщення

Задля створення сприятливих умов задля здорової роботи, котрі б запобігали швидкій втомлюваності очей, виникненню професійних захворювань, нещасних випадків та сприяли підвищенню ефективності праці і якості продукції, виробниче освітлення повинно відповідати наступним вимогам:

- створювати на робочій поверхні освітленість, яке відповідає характеру зорової роботи та не є нижчою поза встановлені норми;

- забезпечити достатню рівномірність і постійність рівня освітленості в виробничих приміщеннях, щоб уникнути частої переадаптації органів зіру;

- не створювати засліплювальної дії як з самих джерел освітлення, так та з інших предметів, яке знаходяться у полі зіру;

- не створювати на робочій поверхні різноманітних і глибоких тіней (особливо рухомих);- повинен існувати достатній задля розрізнення деталей контраст поверхонь, яке освітлюються;

- не створювати небезпечних і шкідливих виробничих чинників (шум, теплові променя, небезпека уражений струмом, пожежо- і вибухонебезпека світильників):

- повинно існувати надійним та простим и експлуатації, економічним і естетичним.

Приміщення, у котрих встановлені персональні комп'ютери, повинні мати природне і штучне освітлення. Природне освітлення здійснюється через світові прорізи (вікна), орієнтовані переважно на північ чи північний схід. Штучне освітлення у приміщені здійснюється системою загального рівномірно-го освітлення. На поверхні столу у зоні розміщення документів штучне освітлення має становити 300-500лк.

Штучне освітлення у приміщеннях із робочими місцями, обладнаними ВДТ має здійснюватись системою загального рівномірного освітлення.

ВИСНОВКИ

В даній роботі проаналізована ефективність впровадження моделі CUDA при рішенні математичних завдань, розглянута архітектура, можливості і впровадження цієї моделі в науково-технічних розрахунках. Технологія CUDA є однією із реалізацій моделі GPGPU задля перенесення одночасних розрахунків із центрального процесору на відеоадаптер N-vidia. Розглянуто вживання даної обчислювальної моделі на кількох реальних прикладах наукових і інженерних завдань, таких як здійснення методу Viola-Jones, програмування задачі розповсюдження світла в багатошаровому середовищі, здійснення методу Монте-Карло задля завдань оптичної біомедицинської діагностики, рішення завдань комп'ютерного зіру.

В результаті виконаного дослідження можливо переконатися у через це, яке програмно-апаратна архітектура задля розрахунків на відеочіпах CUDA підходить задля рішення багатьох завдань, яке вимагають великої значення одночасних розрахунків. CUDA працює на усіх сучасних рішеннях N-VIDIA, зручна в вживанні (у зрівнянні із іншими моделями програмування GPGPU) та вже достатньо довгий період знаходиться у розробці, яке мінімізує кількість недоробок та помилок.

Отримано порівняльну характеристику ефективності реалізації програм задля різної розмірності елементів і різної значення CUDA-ядр. Швидкість реалізації програм найчастіше не зростає лінійно при збільшенні значення стрімів, адже звичайно методи не є повністю паралельним, частина часу реалізації витрачається на синхронізацію і доступ до спільних ресурсів.

Технологія CUDA є доступною будь-якому розробникові програмного забезпечення незалежно з сфери діяльності та масштабу завдань. Коли вживані у роботі методи добре розпаралелюються (точність розрахунків, об'єм сховища даних), то розробникові необхідно опанувати іншу парадигму програмування, властиву паралельним обчисленням, та тоді витрати часу на програмування повернуться в вигляді істотного приросту ефективності (в 50-100 разів задля примітивних процедур, в 5-10 разів задля складних високорівневих методів).

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Семеренко В.П. Технології паралельних обчислень: навчальний посібник – Вінниця: ВНТУ, 2018. – 104 с.
2. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник. – К.: Корнійчук, 2014. – 284 с. – ISBN 996-7599-01-9.
3. Gregory J. Game Engine Architecture, Third Edition / Jason Gregory., 2018. – 1240 с.
4. Robey R., Zamora Y. Parallel and High Performance Computing. New York, NY: Manning Publications, 2021. 704 p.
5. Foster I. Designing and Building Parallel Programs. London: Pearson Plc, 2019. 408 p.
6. Grama A., Gupta A., Karypis G., Vipin K. Introduction to Parallel Computing. Boston, MA: Addison-Wesley, 2003. 664 p.
7. Mattson T. G., He Y., Koniges A. E. The OpenMP Common Core: Making OpenMP Simple Again. Cambridge, MA: The MIT Press, 2019. 320 p.
8. Van Der Pas R., Stotzer E., Terboven C. Using OpenMP-The Next Step: Affinity, Accelerators, Tasking, and SIMD. Cambridge, MA: The MIT Press, 2017. 392 p.
9. Prickett T. M. Diving deep into the NVIDIA Ampere GPU architecture [Електронний ресурс] / Timothy Morgan Prickett. – 2020. https://www.nextplatform.com/2020/05/28/diving-deep-into-the-nvidia-ampere-gpu-architecture/.
10. About CUDA [Електронний ресурс] / NVIDIA – Режим доступу до ресурсу: https://developer.nvidia.com/about-cuda.
11. Le D. Towards Microarchitectural Design of Nvidia GPUs [Електронний ресурс] / Dung Le. – 2020. https://medium.com/distributed-knowledge/towards-microarchitectural-design-of-nvidia-gpus-part-1-abe2fd5d9e52.
12. Evanson N. Nvidia Ampere vs. AMD RDNA 2: Battle of the Architectures [Електронний ресурс] / Nick Evanson. – 2020. https://www.techspot.com/article/2151-nvidia-ampere-vs-amd-rdna2/.

ДОДАТОК А

Фрагменти тексту програми мовою CUDA/C для реалізації множення матриць

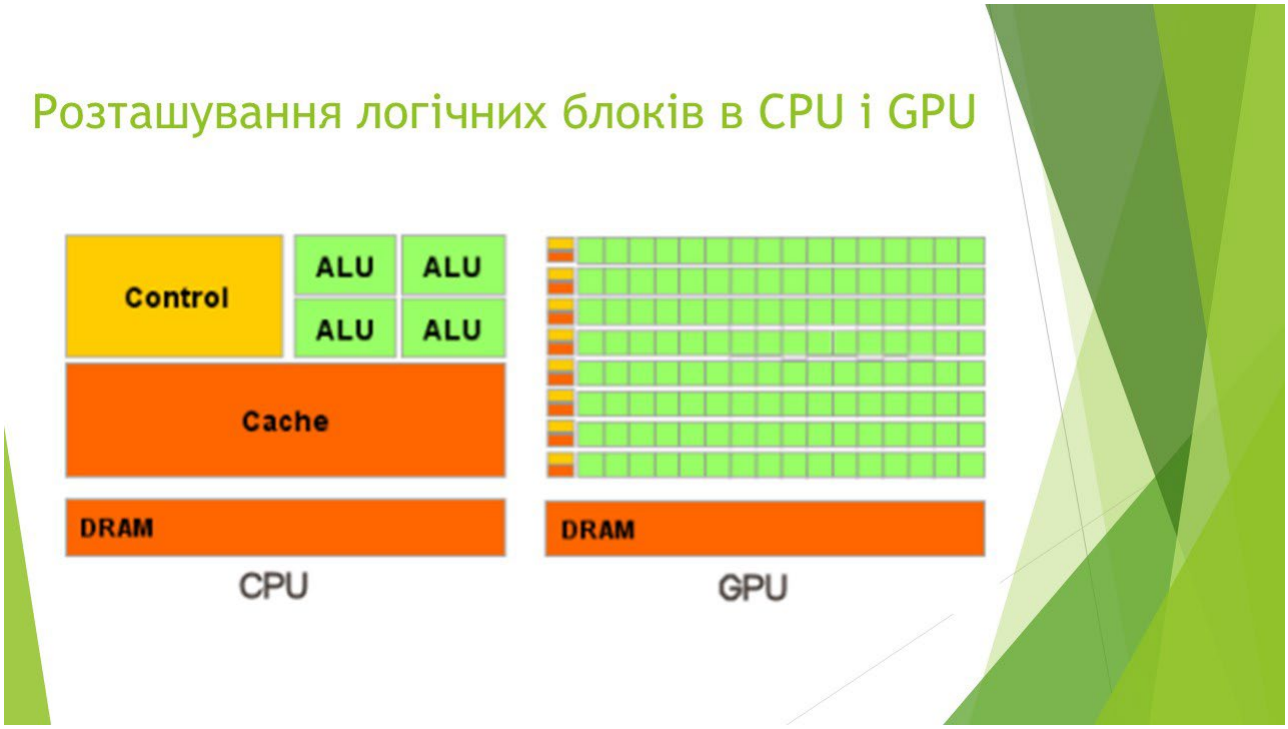
```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>

#define BLOCK 8
#define SIZE (BLOCK * 64)
#define TILE_SIZE (8)

int n;


float *
create_matrix_h(unsigned int w, unsigned int h) {
  float *m;
  m = (float *) malloc(w * h * sizeof(float));
  if (m == NULL) {
    fprintf(stderr, "Failed to malloc.\n");
    exit(1);
  }
  return m;
}

void
print_matrix(const float *m, const int w, const int h) {
  int x, y;
  for (y = 0; y != h; ++y) {
    for (x = 0; x != w; ++x)
      printf("%.03f ", m[y*w + x]);
    printf("\n");
  }
}


void
cpu_mult(const float *m1, const float *m2, float *m3, unsigned int width) {
  unsigned int i, j, k;
  float result;

  for (i = 0; i != width; ++i) {
    for (j = 0; j != width; ++j) {
      result = 0;
      for (k = 0; k != width; ++k)
        result += m1[i*width + k] * m2[k*width + j];
      m3[i*width + j] = result;
    }
  }
}


__global__ void
kernel3(const float *m1, const float *m2, float *m3, unsigned int width) {
  const unsigned int row = blockIdx.y*blockDim.y + threadIdx.y;
  const unsigned int col = blockIdx.x*blockDim.x + threadIdx.x;
  unsigned int t, i;
  float result = 0, a, b;

  for (t = 0; t < width / TILE_SIZE; ++t) {
    for (i = 0; i != TILE_SIZE; ++i) {
```

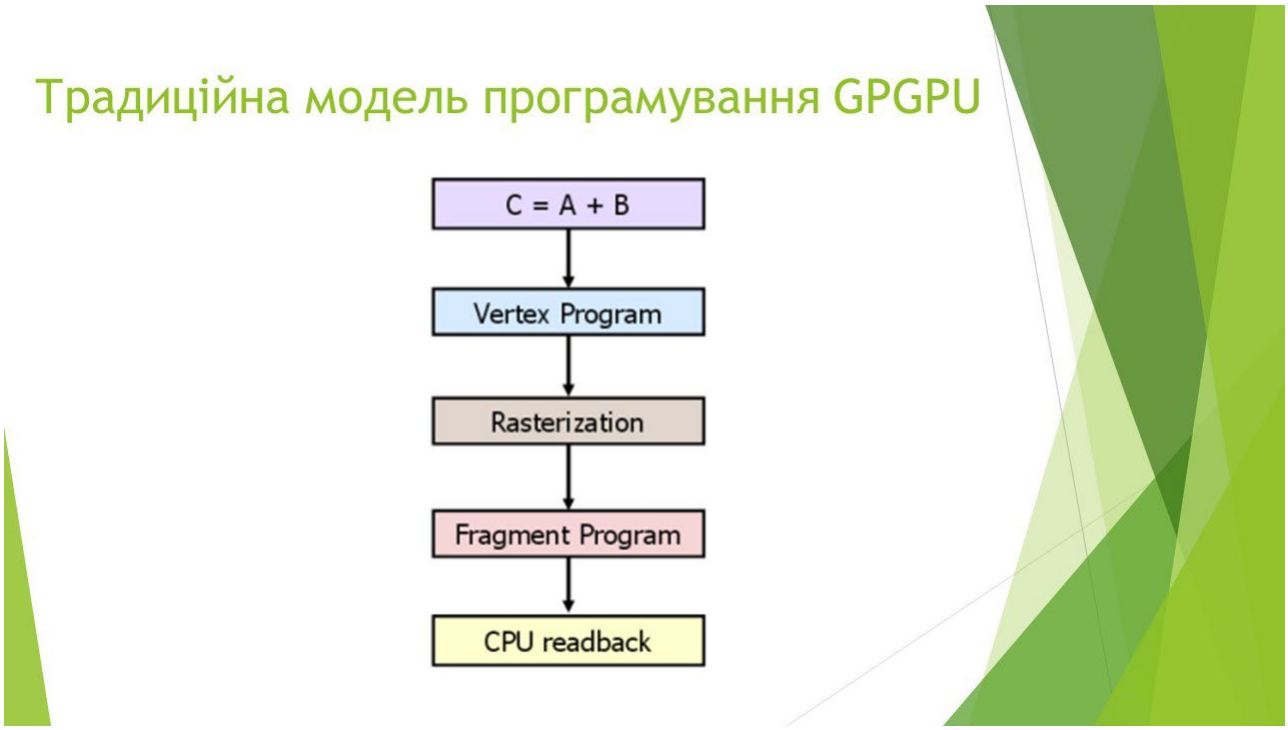
```
      a = m1[row*width + t*TILE_SIZE + i];
      b = m2[(t*TILE_SIZE + i)*width + col];
      result += a * b;
    }
    __syncthreads();
  }
  m3[row*width + col] = result;
}

float *
create_matrix_d(int w, int h) {
  float *m;
  if (cudaMalloc(&m, w * h * sizeof(float)) == cudaErrorMemoryAllocation) {
    fprintf(stderr, "Failed to cudaMalloc.\n");
    return NULL;
    //exit(1);
  }
  return m;
}

void
fill_matrix_h(float *const m, int w, int h, float *const values, int nvalues) {
  int i, j = 0;
  for (i = 0; i != w * h; ++i) {
    m[i] = values[j];
    j = (j + 1) % nvalues;
  }
}

int
main(void) {
    int k;
    if (scanf("%d", &n) !=1 || n<1){
        return 0;
    }
    if (scanf(" %d", &k) !=1 || k<0){
        return 0;
    }
    float *hm[3], *dm[3];
    dim3 bdim(TILE_SIZE, TILE_SIZE);
    dim3 gdim(SIZE/TILE_SIZE, SIZE/TILE_SIZE);
    int i;
    for(i=0; i<3; ++i) {
        hm[i] = create_matrix_h(SIZE, SIZE);
        dm[i] = create_matrix_d(SIZE, SIZE);
    }
    float tem[n*n];
    for(i=0; i<n*n; ++i) {
        if (scanf(" %f", &tem[i]) !=1){
            return 0;
        }
    }
    float temid[n*n];
    int j = 0;
    for (i = 0; i != n*n; ++i) {
        if (i==0 || j == n) { // not j + (n+1)
            temid[i] = 1;
            j=0;
        }
        else {
            temid[i] = 0;
            j++;
        }
    }
    fill_matrix_h(hm[0], SIZE, SIZE, tem, sizeof(tem)/sizeof(float));
    fill_matrix_h(hm[1], SIZE, SIZE, temid, sizeof(temid)/sizeof(float));
```

```
    cudaMemcpy(dm[0], hm[0], SIZE*SIZE*sizeof(float), cudaMemcpyHostToDevice);
    dm[1] = dm[0]; // For the first iteration Result = A * A;
    int w;
    if (k==0) {
        hm[2] = hm[1];
    }
    else if (k==1) {
        hm[2] = hm[0];
    }
    else {
        for (w=1; w<k; ++w) {
            kernel3<<<gdim, bdim>>>(dm[0], dm[1], dm[2], SIZE);
            cudaThreadSynchronize();
            // No need to copy back to host
            // cudaMemcpy(hm[2], dm[2], SIZE*SIZE*sizeof(float), cudaMemcpyDeviceToHost);
            // Copy between device pointers
            cudaMemcpy(dm[1], dm[2], SIZE*SIZE*sizeof(float), cudaMemcpyDeviceToDevice);
        }
        cudaMemcpy(hm[2], dm[1], SIZE*SIZE*sizeof(float), cudaMemcpyDeviceToHost);
    }


    print_matrix(hm[2], n, n);

    return 0;

}
```

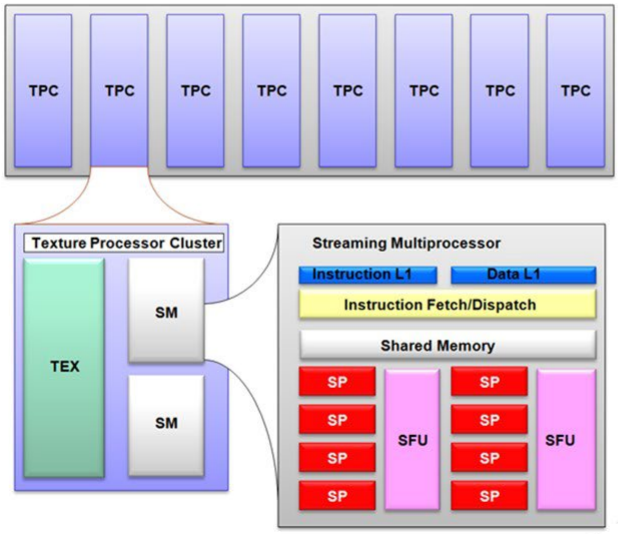
ДОДАТОК Б

Слайди презентації випускної роботи

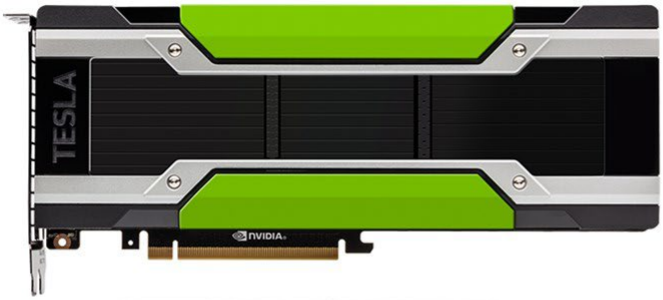


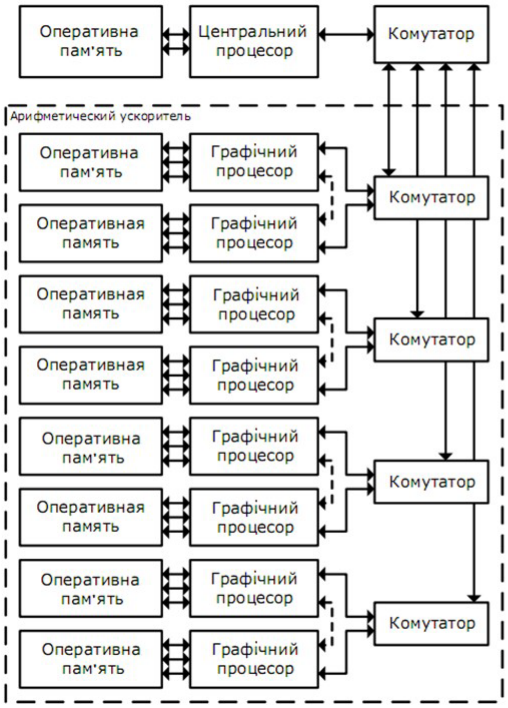


Узагальнена структура відеоадаптера NVidia

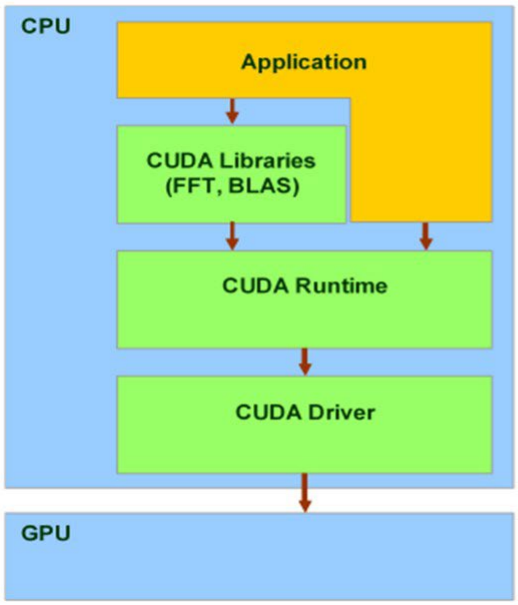


Відеоадаптери Nvidia Tesla



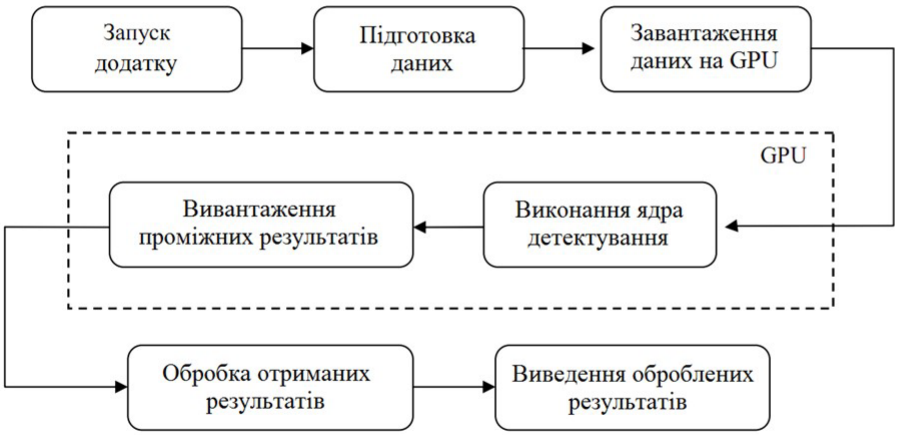


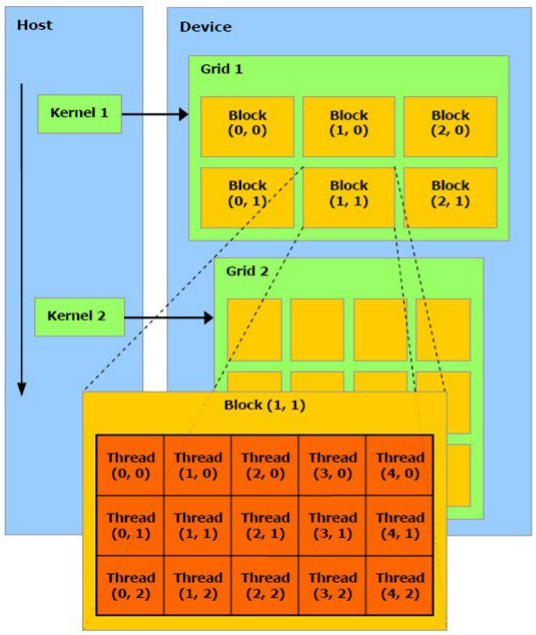
Структурна схема гібридної ОС



Рівні обробки даних за технологією NVIDIA CUDA

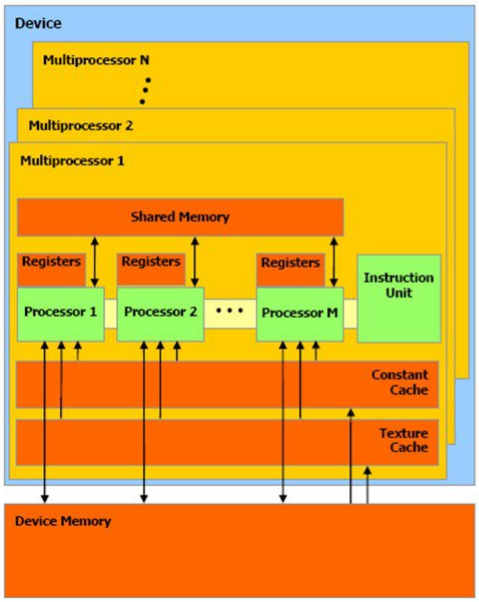
Схема роботи алгоритму рішення задачі з використанням GPU





Групування потоків в моделі програмування CUDA

Модель пам'яті CUDA



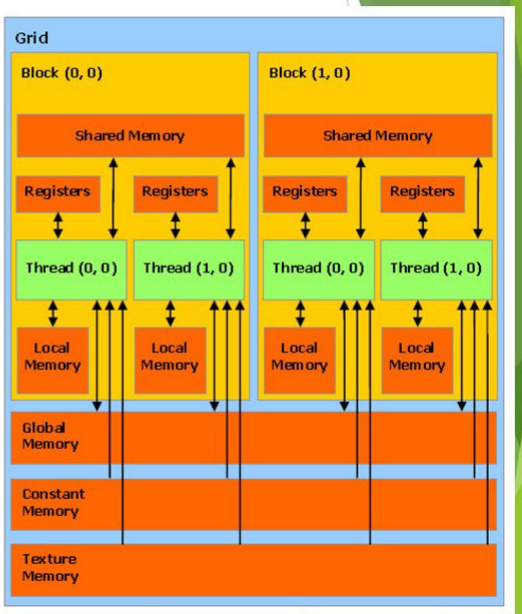
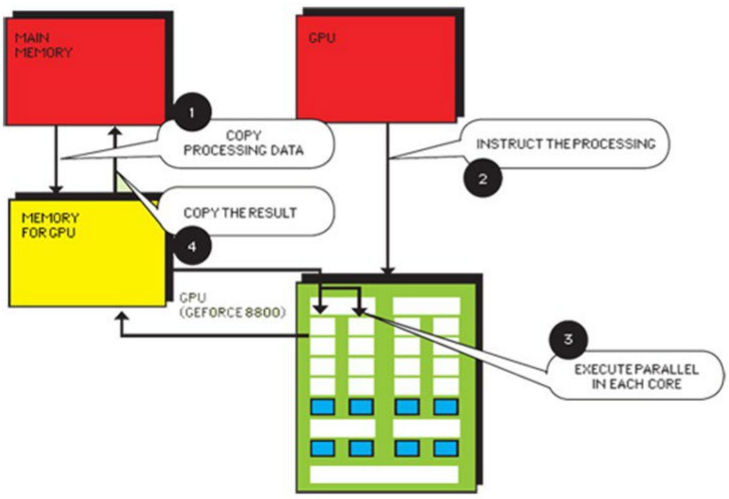
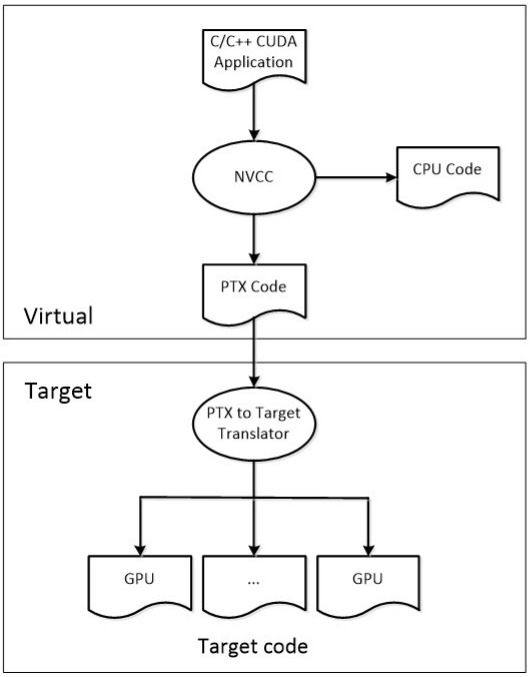
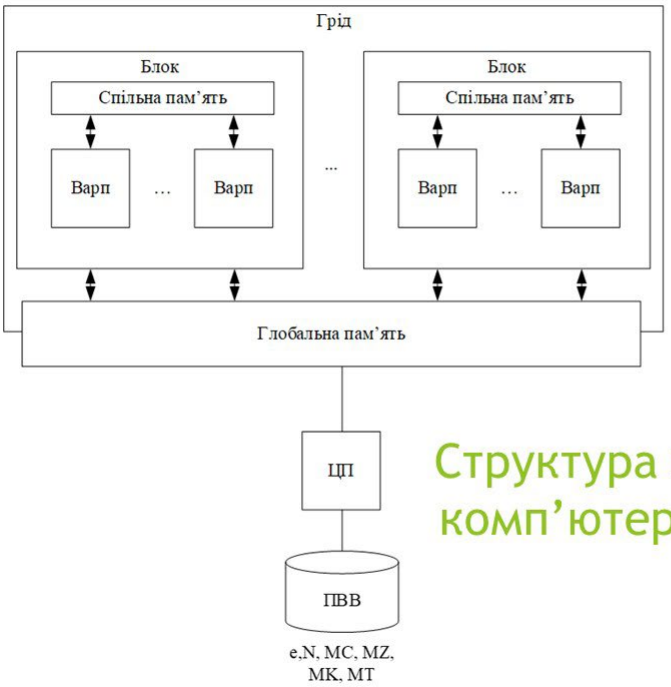


Схема взаємодії між CPU і GPU





Стадії компіляції CUDA-додатку



Структура паралельної комп'ютерної системи

Паралельний математичний алгоритм

1. $m_i = \max(R_H), i = \overline{1,P}$
2. $m = \max(m, m_i), i = \overline{1,P}$
3. $MX_H = e * (MC_H * MZ * MK) + m * MT_H$

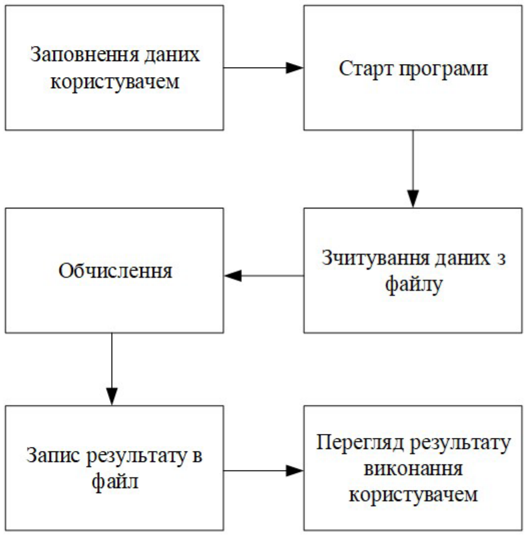
де

- e, m, MZ, MK – спільні ресурси;
- P – кількість потоків;
- N – розмірність векторів та матриць;
- H = N/P;
- R_H – H елементів вектору R;
- MC_H – рядків матриці MC;
- MT_H – рядків матриці MT;
- MX_H – рядків матриці MX;

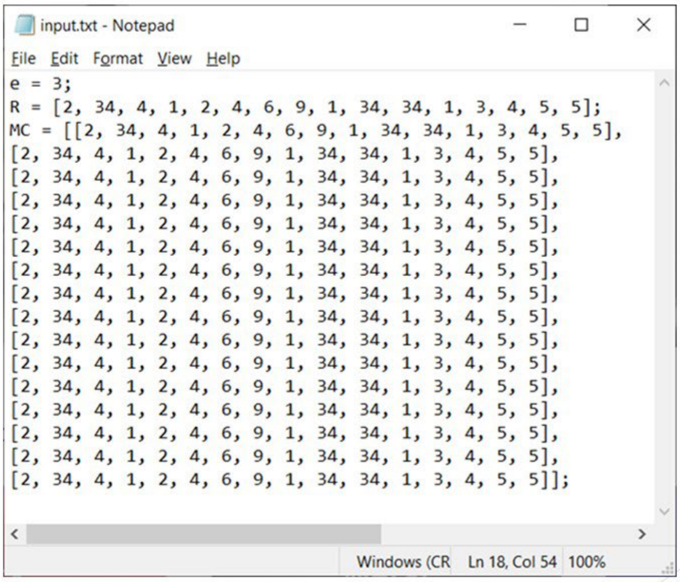
Алгоритм взаємодії потоків

Назва	*Бар'єри, КД*
1. Введення e, R, MC, MZ, MK, MT	
2. Передача e, R, MC, MZ, MK, MT у глобальну пам'ять	
3. Якщо tid = 1, надіслати сигнал про введення e, R, MC, MZ, MK, MT	$S_{2\text{-}P\text{ - }1}$
4. Якщо tid != 1, очікувати сигнал про введення e, R, MC, MZ, MK, MT	$W_{2\text{-}P\text{ - }1}$
5. Обчислення $m_i = \max(R_H)$	
6. Обчислення $m=\max(m, m_i)$	
7. Надіслати сигнал про обрахування m	$S_{1\text{-}(tid\text{-}1),(tid+1)\text{-}P\text{ - }tid}$
8. Очікувати сигнал про обрахування m	$W_{1\text{-}(tid\text{-}1),(tid+1)\text{-}P\text{ - }tid}$
9. Копіювати $e_i = e$, $m_i = m$, $MZ_i = MZ$, $MK_i = MK$	КД
10. Обчислення $MX_H=e_i*(MC_H*MZ_i*MK_i) + m_i*MT_H$	
11. Якщо tid != 1, надіслати сигнал про закінчення обрахування	$S_{1\text{ – }2\text{-}P}$
12. Якщо tid = 1, очікувати сигнали про закінчення обрахування MX_H	$W_{2\text{-}P\text{ - }1}$
13. Якщо tid = 1, передати результат у ОЗП	
14. Виведення результату MX	

Схема процесу вводу-виведення даних



Приклад заповнення вхідного файлу



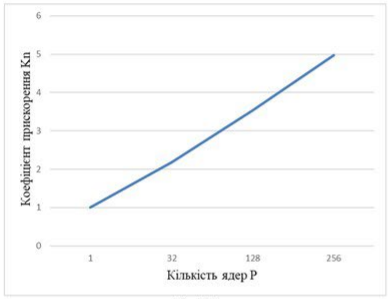
input.txt - Notepad

File Edit Format View Help

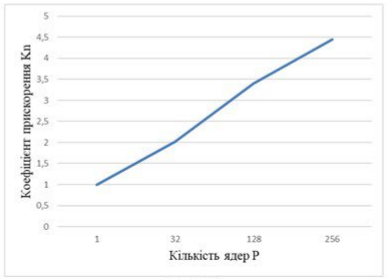
```
e = 3;
R = [2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5];
MC = [[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5],
[2, 34, 4, 1, 2, 4, 6, 9, 1, 34, 34, 1, 3, 4, 5, 5]];
```

Windows (CR Ln 18, Col 54 100%

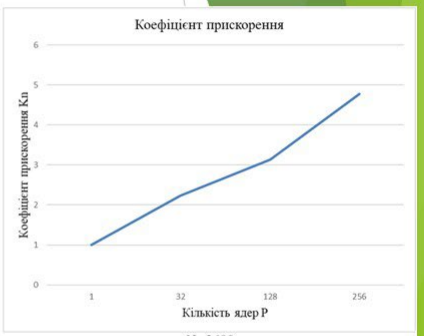
Графіки зміни коефіцієнту прискорення програми в залежності від кількості ядер



N=896

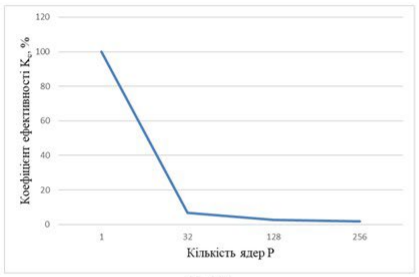


N=1792

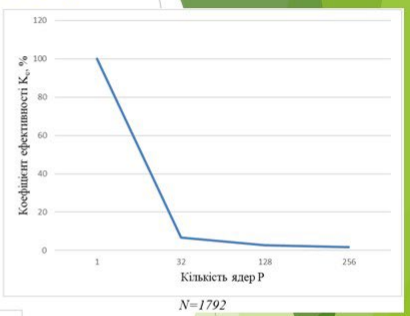


N=2688

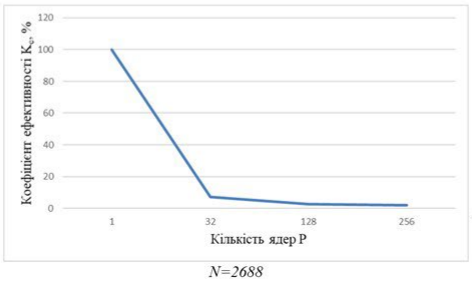
Графіки зміни коефіцієнту ефективності програми в залежності від кількості ядер



N=896



N=1792



N=2688

ВСП «Одеський технічний фаховий коледж ОНТУ»

ВІДГУК

керівника про кваліфікаційну роботу бакалавра

Семка Максима Олеговича

(прізвище, ім`я та по батькові)

Спеціальність *123 "Комп'ютерна інженерія"*

Тема кваліфікаційної роботи *Аналіз ефективності застосування технології CUDA при рішенні математичних задач*

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) Обсяг і якість виконання роботи (графічного матеріалу і розрахунково-пояснювальної записки) *Випускна робота виконана відповідно технічному завданню. Пояснювальна записка до випускної роботи містить 80 сторінок. У пояснювальній записці викладено аналіз сучасного стану розвитку технології CUDA, яка дозволяє використовувати обчислювальні ресурси відеоадаптера для рішення обчислювальних задач широкого кола, зокрема наукового та інженерного напрямку, реалізовано математичну задачу множення матриць. Графічна частина складається з 19 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра, розробку виконано у повному обсязі.*

б) Самостійність роботи *Протягом виконання випускної бакалаврської роботи Семко Максим поступово та послідовно виконував всі етапи, проявив ініціативу у створенні загальної концепції та реалізації випускної роботи. Всі роботи він виконував самостійно, з оглядом на рекомендації керівника.*

в) Теоретична підготовка здобувача освіти

Семко Максим під час роботи над випускною бакалаврською роботою вивчив і опрацював достатню кількість літературних джерел за даною тематикою.

Вважаю, що теоретична підготовка здобувача освіти достатня і він готовий до захисту роботи.

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

Під час виконання роботи Семко Максим мав змогу самостійно приймати окремі рішення з виконання програмної частини роботи та показав вміння організовано працювати над поставленою задачею, складати та оформлювати презентацію проекту, користуючись сучасними комп'ютерними програмними засобами, такими як Microsoft Visual Studio, Microsoft PowerPoint, Microsoft Visio.

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Добре*

Прізвище, ім`я, по батькові *Кірєєв Ігор Анатолійович*

Місце роботи і посада керівника роботи

Державний університет інтелектуальних технологій і зв'язку,

доцент каф. інформаційної безпеки та передачі даних

Підпис

«12» червня 2024р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра здобувача освіти

відділення комп'ютерних систем

Семка Максима Олеговича

(прізвище, ім`я та по батькові)

Спеціальність 123 "Комп'ютерна інженерія"

Освітня програма «Комп'ютерна інженерія»

Керівник дипломного проекту (роботи) Кірєєв Ігор Анатолійович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) Аналіз ефективності застосування технології CUDA при рішенні математичних задач

Обсяг розрахунково-пояснювальної записки 80 сторінок

Обсяг графічної (презентаційної) частини 19 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлена на рецензію кваліфікаційна робота бакалавра повністю відповідає меті проектування та технічному завданню. Тематика кваліфікаційної роботи є актуальною для своєї галузі та присвячена аналізу ефективності застосування технології CUDA при рішенні математичних задач.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті питання застосування технології CUDA при рішенні математичних задач, особливості архітектури CUDA.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана охайно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано мету та цілі аналізу;
2. Проведено ретельний аналіз апаратно-програмних засобів технології CUDA;
3. Виконано дослідження ефективності застосування технології CUDA при рішенні математичних задач на конкретних прикладах.

д) основні недоліки дипломного проекту (роботи)

1. При тестуванні ефективності CUDA доцільно було б використовувати більш потужні відеоадаптери;
2. Варто було б передбачити візуальний інтерфейс для застосунку.

Оцінка розрахункової частини Добре

Оцінка графічної частини Добре

Загальна оцінка Добре

Прізвище, ім'я, по батькові рецензента к.т.н. Селіванова Алла Віталіївна

Місце роботи і посада рецензента Одеський національний технологічний університет, декан факультету комп'ютерної інженерії, програмування та кіберзахисту

МІНІСТЕРСТВО ОСВІТИ І НАУКИ ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ ФАКУЛЬТЕТ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ, ПРОГРАМУВАННЯ ТА КІБЕРЗАХИСТУ

Підпис:

« 17 » червня 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григорівна Краснокутська

ID перевірки:
1016243028

Дата перевірки:
10.05.2024 21:28:59 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
10.05.2024 21:35:20 EEST

ID користувача:
100011688

Назва документа: **2БКС-28_Максим Семко**

Кількість сторінок: **61** Кількість слів: **12415** Кількість символів: **91157** Розмір файлу: **1.78 MB** ID файлу: **1016026459**

8.65%
Схожість

Найбільша схожість: **1.81%** з Інтернет-джерелом (https://card-file.ontu.edu.ua/server/api/core/bitstreams/9fc96430-656..

8.65% Джерела з Інтернету | 519 ..... Сторінка 63

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 108

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Семко Максим Олегович,
здобувач освіти гр. 2БКС-28, та

Кірєєв Ігор Анатолійович,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Аналіз ефективності застосування технології CUDA при рішенні математичних задач» (автор роботи – Семко М.О., керівник роботи – Кірєєв І.А.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Семко М.О. /

Керівник _____ / Кірєєв І.А. /

«13» червня 2024 р.