Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: «Розробка програмної підсистеми алгоритмічного перетворення даних з використанням різнорідних обчислювальних засобів мовою С»

Виконав(ла): студент(ка) VI курсу, групи СПм-61
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Михайлівський О.В. (прізвище та ініціали)

Керівник (підпис) Цуприк Г.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«_____» _____ 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Михайлівському Остапу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмної підсистеми алгоритмічного перетворення даних з використанням різнорідних обчислювальних засобів мовою С

Керівник роботи Цуприк Галина Богданівн, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «4» грудня 2023 року № 4/7-1141
2. Термін подання студентом завершеної роботи 12 грудня 2023 р.
3. Вихідні дані до роботи Завдання на розробку підсистеми

4. Зміст роботи (перелік питань, які потрібно розробити)
Сформулювати, погодити та затвердити тему кваліфікаційної роботи;
розробити та затвердити завдання;
проаналізувати завдання, зробити огляд та проаналізувати бібліографічні матеріали щодо стану справ та тенденій в обраному напрямку дослідження;
сформулювати завдання, обґрунтувати цілі дослідження;
розробити та спроєктувати підсистему;
описати технології, етапи розробки та розроблювану програмну підсистему (у разі потреби)
розробити план тестування програмного продукту; оформити допоміжну документацію;
проаналізувати роботу щодо питань з дотримання положень про охорону праці та безпеку в надзвичайних ситуаціях; оформити пояснювальну записку;
зробити відповідні висновки за результатами виконаної роботи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративна частина кваліфікаційної роботи оформляється у вигляді слайдів, висвітлює основні розділи та етапи роботи та містить:
тему роботи, мету, предмет та об'єкт дослідження,
сформульовані завдання, перелічено етапи виконання кваліфікаційної роботи, короткий аналіз актуальності теми, варіант архітектури програмної системи,
представлено опис програмного забезпечення; опис програмного забезпечення та тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М.		
Безпека в надзв. ситуаціях			
Нормоконтроль	Стоянов Ю.М.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та погодження теми з керівником. Створення та затвердження технічного завдання	26.06.2023-02.07.2023	виконано
2.	Аналіз технічного завдання, підбір та аналіз		
	бібліографічних матеріалів необхідних для виконання	03.07.2023	виконано
	кваліфікаційної роботи		
3.	Проєктування та розробка програмного забезпечення	01.09.2023	виконано
4.	Тестування та дослідна експлуатація програмного		
	забезпечення	19.11.2023	виконано
5.	Створення допоміжної документації. Написання та		
	перевірка на відповідність вимогам пояснювальної		
	записки	03.12.2023	виконано
6.	Апробація результатів роботи	13.12.2023	виконано
7.	Перевірка програмою антиплагіат	04.12.2023	виконано
8.	Охорона праці	24.12.2023	виконано
9.	Безпека в надзвичайних ситуаціях	24.12.2023	виконано
10.	Формування/зшивання записки кваліфікаційної роботи	25.12.2023	виконано
11.	Нормоконтроль	25.12.2023	виконано
12.	Попередній захист	12.12.2023	виконано
13.	Захист кваліфікаційної роботи	27.12.2023	

Студент _____ Михайлівський О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить (загальний обсяг):

стр., рис., табл., літ. джер., додатків.

ОБ'ЄКТ, ПРЕДМЕТ, МЕТА, РОЗРОБКА, АНАЛІЗ, ПРОЄКТУВАННЯ, КОНСТРУЮВАННЯ, ТЕСТУВАННЯ, OPENCL, C#, VISUAL STUDIO.

Об'єктом дослідження та розробки є програмна підсистема призначенням якої є алгоритмічне (криптографічне) перетворення даних, які не призначені для широкого загалу та містять інформацію конфіденційного толку, чи можуть стосуватись комерційної таємниці підприємства чи організації.

Метою роботи є розробка неоднорідної реалізації алгоритмів симетричного типу алгоритмічного (криптографічного) перетворення даних.

За методи розробки обрано технології C#, фреймворк OpenCL, середовище Visual Studio.

OBJECT, SUBJECT, PURPOSE, DEVELOPMENT, ANALYSIS, SOFTWARE DESIGN, SOFTWARE CONSTRUCTION, SOFTWARE TESTING, OPENCL, C#, VISUAL STUDIO.

The object of research and development is a software subsystem whose purpose is the algorithmic (cryptographic) transformation of data that are not intended for the general public and contain confidential information, or may relate to the commercial secret of an enterprise or organization.

The method of work is the development of a heterogeneous implementation of algorithms of a symmetric type of algorithmic (cryptographic) data transformation.

Development methods were C# technologies, the OpenCL framework, and the Visual Studio environment.

ЗМІСТ

ВСТУП 6
1 РОЗРОБКА АЛГОРИТМІВ НЕОДНОРІДНИХ ОБЧИСЛЕНЬ І МЕТОДІВ АЛГОРИТМІЧНОГО ПЕРЕТВОРЕННЯ ДАНИХ 8
1.1 Аналіз щодо вимог до підсистеми алгоритмічного перетворення даних 8
1.1.1 Аналіз та огляд предметної області 8
1.1.2 Постановка завдання 9
1.1.3 Пошук варіантів використання 10
1.1.4 Опис ключових варіантів використання 13
1.2 Проєктування підсистеми алгоритмічного перетворення даних 14
1.2.1 Аналіз та вибір процесу розробки для роботи програмної підсистеми 14
1.2.2 Побудова схеми для алгоритмічного перетворення даних 16
1.2.3 Вибір алгоритму шифрування 20
1.2.4 Вивчення та аналіз алгоритму AES, AddRoundKey 26
1.2.5 Аналітичний огляд SubBytes, InvSubBytes, ShiftRows, InvShiftRows, MixColumns, InvMixColumns 31
1.2.6 Розробка структурної схеми підсистеми 35
1.2.7 Побудова UML – діаграми діяльності 37
1.3 Конструювання програмної системи 41
1.3.1 Вибір мови програмування та розробки програмної підсистеми 41
1.4 Використання програмної системи 47
1.4.1 Системні вимоги 47
2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 48
2.1 Тестування процесора 48
2.2 Реалізація ядра (kernel) 49
2.3 Тестування бібліотеки 50
2.4 Аналіз швидкості шифрування/дешифрування 52
2.5 Аналіз продуктивності системи 56
2.6 Переваги та недоліки бібліотеки 57
3 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 59
3.1 Охорона праці 59
3.2 Безпека в надзвичайних ситуаціях 62
3.2.1 Інженерний захист персоналу об’єкту та населення. Правила застосування 62
3.2.2 Характеристики аварій на виробництвах із застосуванням хлору. Перша допомога. Профілактика уражень 65

ВИСНОВКИ 67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 69
ДОДАТКИ 74
Додаток А Лістинг програми
Додаток Б Апробація результатів роботи
Додаток В Відгук, рецензія, протокол подібності
Додаток Г Презентація

ВСТУП

Щоби розкрити сутність та об’єктивний стан наукової проблеми її значущість підстави та вихідні дані взяті до уваги при розробці теми варта надати загальну характеристику роботи в цілому.

Актуальність теми та доцільність проведеної роботи можу аргументувати тим, що ніби і банальні слова інформація та сучасність твердо увійшли у сучасне буремне сьогодення, що все що нас оточує несе якусь хоча б мінімальну інформацію зі змістом і не випадково знаходиться в тому чи іншому місці. Однак, не до всього будь-хто та будь-де може мати доступ, і цьому є різні причини. Одну із них, і напевно чи не найголовнішу, на мою думку, можна охарактеризувати словом «не нашкодь». А вже розмір тієї «шкоди», безпосередньо може залежати від того, де отриманні відомості будуть застосовані і яке призначення чи напрям (плановане чи якесь своє) буде обрано носієм.

В загальному я цікавився науковими дослідженнями, які проводяться на кафедрі програмної інженерії і можу сказати про те, що, навіть не вникаючи в подробиці, обрана мною тема тісно пов’язана з ними, оскільки дослідження це інформація, яка також потребує конфіденційності, аж до її оприлюднення у вигляді кінцевих результатів чи звіту.

Таким чином, погодивши із керівником, на затвердження завідувачем кафедри я подав тему пов’язану із розробкою програмної підсистеми кодування із використанням різнорідних обчислювальних засобів.

Метою роботи є розробка неоднорідної реалізації алгоритмів симетричного типу прямого та зворотного алгоритмічного перетворення даних.

Об’єктом дослідження та розробки є програмна підсистема призначенням якої є алгоритмічне (криптографічне) перетворення даних, які не призначені для широкого загалу та містять інформацію конфіденційного толку, чи можуть стосуватись комерційної таємниці підприємства чи організації.

Вже безпосередньо предмет дослідження міститься в межах об’єкту, зокрема програмної підсистеми і остаточно затверджена тема кваліфікаційної роботи вказана на титульному аркуші в якості її назви.

Якщо говорити про задачі, які були означені як пріоритетні, то я б виділив наступні:

- передбачається, що програмний продукт повинен забезпечувати повноцінне алгоритмічне перетворення даних (криптографічне), яке здійснюється в посимвольній послідовності, використовуючи блочний симетричний алгоритм при цьому;
- планується, що швидкість алгоритмічного перетворення даних не повинна залежати (знижуватись) при збільшенні об’єму оброблюваних даних.
- для алгоритмічного перетворення даних можна вибрати будь-який файл або ввести будь-який текст.
- програма повинна розпаралелювати процес алгоритмічного перетворення даних якнайбільше для швидкого результату
- програма повинна бути орієнтована на платформу Microsoft Windows.

Для реалізації заплановано використати технології фреймворку OpenCL метою якого є доповнення Open GL і OpenA L, які можна назвати відкритими галузевими стандартами прямим призначенням яких є тривимірна комп'ютерна графіка і звук, при використанні можливостей GPU. Важливим моментом є те, що OpenCL розроблявся та підтримується некомерційною організацією Khronos Group – консорціум, до якого входять такі відомі IT компанії як Apple,AMD, Intel,SunMicrosystems, Sony CE та деякі інші. Відкрита мова обчислень OpenCL вигідно виділяється також бо «дозволяє» програмам динамічно визначати доступність процесорів, з багатоядерними центральними процесорами та графічними процесорами включно. Саме завдяки цьому розробники мають можливість, проте залежать від можливостей апаратного забезпечення самих клієнтів, достатньо інтенсивно покращувати продуктивність своїх програм.

1 РОЗРОБКА АЛГОРИТМІВ НЕОДНОРІДНИХ ОБЧИСЛЕНЬ І МЕТОДІВ АЛГОРИТМІЧНОГО ПЕРЕТВОРЕННЯ ДАНИХ

1.1 Аналіз щодо вимог до підсистеми алгоритмічного перетворення даних

1.1.1 Аналіз та огляд предметної області

В результаті обговорення я обрав собі тему, яку і погодив мій керівник, що стосується розробки програмної підсистеми алгоритмічного перетворення даних з використанням різнорідних обчислювальних засобів при використанні сучасних інформаційних технологій [1].

Основне призначення розроблюваною мною програмної підсистем, є пряме та зворотнє алгоритмічне перетворення даних, які не призначені для широкого загалу та містять інформацію конфіденційного толку, чи можуть стосуватись комерційної таємниці підприємства чи організації. Передбачається, що програмний продукт може застосовуватися в будь–якій галузі, де виникає подібна потреба із-за невідповідного степеню надійності сховищ, а також для приховування інформації з метою її передачі по незахищеним каналам зв’язку. Зберігання опрацьованих шляхом алгоритмічного перетворення даних в недостатньо надійних сховищах дозволить уникнути необхідності в фізичному захисті сховищ [2-8].

OpenCL (англ. Open Computing Language) – стандарт для універсального паралельного програмування різних типів процесорів, який не потребує ліцензійних відрахувань. Стандарт надає програмістам зручний та ефективний доступ до всієї маси необнорідних обчислювальних платформ. OpenCL підтримується широким колом програмних засобів: від вбудованих і клієнтських додатків до високопродуктивних рішень через досить низькорівневий, переносний програмний інтерфейс [9].

У результаті створення ефективного програмного інтерфейсу OpenCL придатний для формування базового рівеня паралельного обчислювального

коду незалежної платформи програмних інструментів, проміжного ПЗ та інших видів додатків.

OpenCL також добре підходить на важливу роль інтерактивних графічних додатків, що інтенсивно розвиваються, і які включають як універсальні алгоритми паралельних обчислень, так і алгоритми побудови складних графічних сцен.

Стандарт OpenCL описує API для управління процесом паралельних обчислень серед гетерогенних процесорів (процесорів в системі різного типу), а також це так само кросплатформна мова програмування з детально описаним обчислювальним середовищем [10,11].

1.1.2 Постановка завдання

В результаті проведеного аналізу, мною були сформульовані наступні завдання, які повинна виконувати програмна система, зокрема:

- програма повинна виконувати пряме алгоритмічне перетворення даних, використовуючи блочний симетричний алгоритм;
- програма повинна здійснювати і зворотнє алгоритмічне перетворення даних, використовуючи блочний симетричний алгоритм;
- алгоритмічне перетворення даних повинне здійснюватись достатньо швидко, і не залежати від об’єму даних;
- для процесу має бути придатним / обраним будь-який файл чи будь-який «залитий» текст;
- процес алгоритмічного перетворення програмно повинен розпаралелюватись, з метою прискорення отримання результату;
- програмна підсистема повинна бути орієнтована на платформу Microsoft Windows.

1.1.2 Постановка завдання

Для даного програмного продукту – програмної підсистеми, головним актором є користувач. Відповідно, цьому актору необхідно надати можливість виконувати деякі визначені функції. Візуалізувати це можливо через представлення у вигляді такому як діаграма прецедентів.

Діаграму прецедентів (англійською ueuse case diagram) – призначено для опису функціонального призначення системи чи, якщо іншими словами, конкретно те, що робитиме система в процесі свого функціонування.

Дана діаграма подана у вигляді сутностей та акторів, які мають здатність взаємодіяти із системою за допомогою варіанті використання. Актор або дійова особа – це якась сутність, яка може взаємодіяти із системою ззовні. Зокрема, ним може бути як людина, так і технічний пристрій чи будь-яка програма.

Варіант використання (англійською use case), зазвичай, призначено власне для описів сервісів, які надаються акторові певною системою, тобто кожним варіантом використання визначається той чи інший набір дій, який здійснюється системою при діалозі з актором. Проте немає жодної інформації, як саме реалізовуватиметься взаємодія акторів із системою [12, 13] .

Конструкція чи іншими словами стандартний елемент мовиUML варіант прецедентів застосовують для так званої поведінки системи або ж для іншої предметної області без деталізації та огляду на внутрішню структуру цієї системи.

Окремий варіант використання на діаграмі зазвичай позначається у вигляді еліпсому, всередині якого (або під ним) розташовується його лаконічною назвою чи ім’ям із роз’ясненням.

В цілому варіанти використання визначають закінчений аспект чи фрагмент варіанту поведінки деякої із сутностей, але без розкриття її внутрішньої структури. За таку сутність може виступати як вихідна система так

і якийсь якийсь інший з елементів моделі, який наділено власною поведінкою, подібно як із підсистемою або класом у моделі системи [14-17].

Актор представляє собою якусь зовнішню, відносно до модельованої системи сутність, яка може взаємодіяти з системою та використовувати її функціональні можливості з метою досягнення певних цілей чи вирішення поставлених завдань.

Короткий опис актора можна переглянути в таблиці 1.1.

Таблиця 1.1 – Актор системи

Актор	Короткий опис
Користувач	Задає файли або послідовність байт, можливість відправляти для шифрування/дешифрування на пристрої, задає певний кусок тексту для обробки.

Отже, в даній системі є один основний актор: Користувач, який виконує функції вибору шифрування, може задавати одночасно кілька файлів для обробки. Об’єднавши вищеназвані варіанти використання та актора, можнарозробити мінімальну та просту діаграму використання, де відображено зв’язки актора та прецедентів (варіантів використання). Переглянути дану діаграму можна на рисунку 1.2.

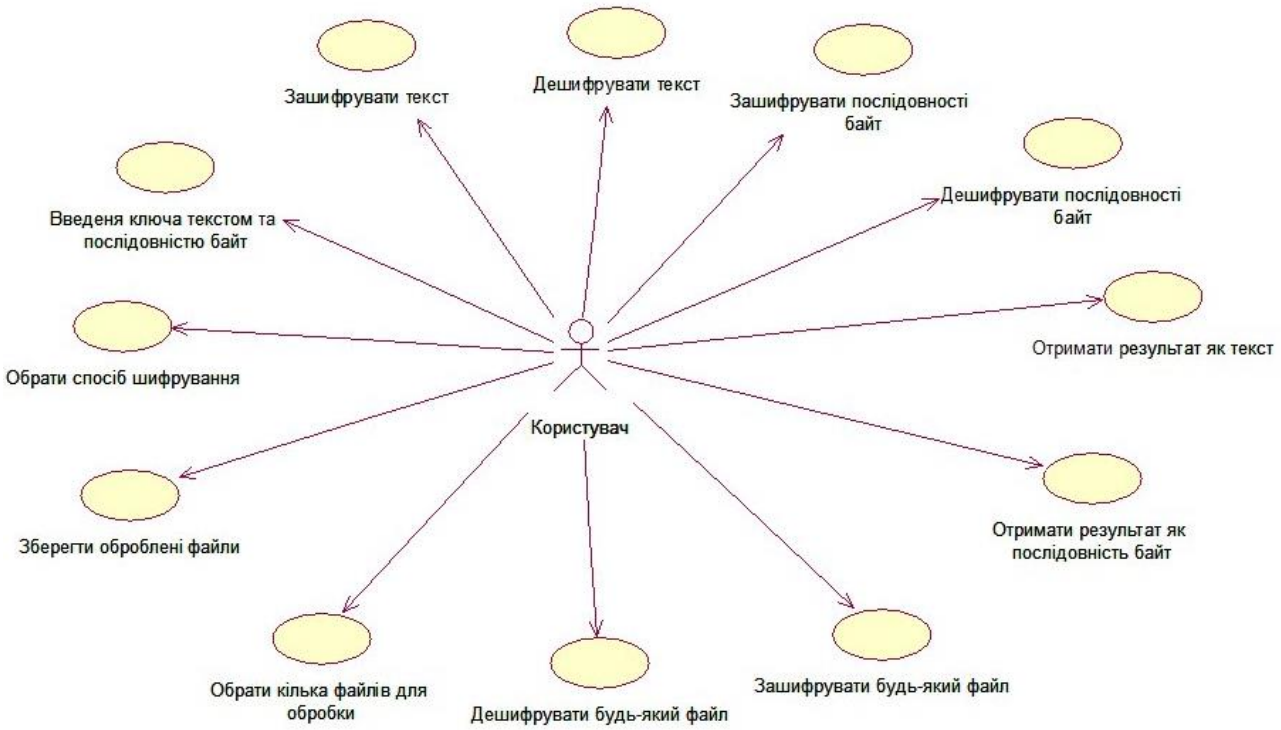


Рисунок 1.2 – Діаграма прецедентів

Дана діаграма прецедентів по своїй суті представлена у вигляді графу, на якому представлено акторів, також прецедентів, асоціацій між ними (акторами) та прецедентів. Діаграми прецедентів представляють елементи моделей варіантів використання.

Дана діаграма відображає зв’язки між актором та прецедентами (варіантами використання) у системі.

1.1.3 Опис ключових варіантів використання

Усі варіанти використання актора можна переглянути у таблиці 1.2.

Таблиця 1.2 – Варіанти використання

Код	Актор	Найменування	Опис
К1	Користувач	Зашифровування тексту	Користувач обирає необхідний спосіб зашифровування даних
К2	Користувач	Дешифрування тексту	Користувач має можливість розшифровувати текст
К3	Користувач	Зашифровування послідовності байт	Зашифровує масив байтів і виводить результат
К4	Користувач	Дешифровування послідовності байт	Розшифровує масив байтів і виводить результат
К5	Користувач	Отримання результату як текст	Користувач здатний отримати готовий результат у вигляді тексту
К6	Користувач	Отримання результату як послідовність байт	Можливість отримати послідовність байт в тексті
К7	Користувач	Зашифровування будь-якого файлу	Користувач здатний зашифровувати той чи інший текст
К8	Користувач	Дешифрування будь-якого файлу	Здатність розшифровувати будь-який текст
К9	Користувач	Обрання кількох файлів для обробки	Користувач здатний обрати для оброблення будь-який текст
К10	Користувач	Збереження оброблених файлів	Можливість збереження готового результату

Продовження таблиці 1.2 – Варіанти використання

Код	Актор	Найменування	Опис
К11	Користувач	Обрання способу шифрування	Використання простого процесора або використання OpenCL
К12	Користувач	Введення ключа текстом та послідовністю байт	Задання у вигляді тексту або задання 16-ми байтами

1.2 Проєктування підсистеми алгоритмічного перетворення даних

1.2.1 Аналіз та вибір процесу розробки для роботи програмної підсистеми

Однією з найпоширеніших операційних систем (ОС) на світовому ринку в даний час є Microsoft Windows, тому цю ОС було обрано для реалізації бібліотеки, що забезпечує неоднорідне пряме та зворотнє алгоритмічне перетворення даних з використанням технології OpenCL.

Програма повинна забезпечувати такі функції:

1. пряме алгоритмічне перетворення тексту;
2. зворотнє алгоритмічне перетворення тексту;
3. пряме алгоритмічне перетворення послідовності байт;
4. зворотнє алгоритмічне перетворення послідовності байт;
5. виводу результату як тексту;
6. виводу результату як послідовність байтів;
7. пряме алгоритмічне перетворення будь - якого вибраного файлу;
8. зворотнє алгоритмічне перетворення будь - якого вибраного файлу;
9. можливість одночасного вибору кількох файлів для обробки;
10. зберігання оброблених файлів;

11. вибір способу прямого алгоритмічного перетворення;

12. можливість вводу ключа текстом і послідовністю байтів.

Операційна система також має ряд переваг над іншими та є безпечною, тому вона і є однією із найпоширеніших.

Основними перевагами ОС Microsoft Windows є:

-простота у використанні. Через зручний графічний інтерфейс ця ОС буде зручна будь-якому користувачеві. Також при переході на більш нову версію ОС Windows не виникатиме труднощів, оскільки версії є подібні в плані користування (можливості звісно зростають з кожною новішою версією);

-доступність програмного забезпечення. Існує величезний вибір програмного забезпечення для Windows. Це пов'язано з домінуванням Microsoft на світовому ринку ПК з операційними системами та офісним програмним забезпеченням:

-зворотна сумісність. При переході на новішу ОС Windows не повинні виникнути труднощі з використанням програм, які добре працювали на старіших версіях цієї ОС;

-підтримка нового обладнання. Практично всі виробники апаратного забезпечення надають підтримку їхньому обладнанню останньої версії Windows, коли вони йдуть на ринок з новим продуктом. Домінування Microsoft на ринку програмного забезпечення для Windows просто робить неможливим ігнорування виробниками апаратного забезпечення підтримки цієї ОС. Практично будь-яке обладнання для комп’ютерів є сумісним з ОС Windows [18].

Plug & Play(дослівно з англійської «Увімкни і грай(працюй)»). В якості операційної системи для середнього домашнього користувача, Windows, як і раніше, має перевагу над конкурентами в області Plug & Play підтримки апаратного забезпечення ПК. Якщо встановлені правильні драйвери, Windows, як правило, добре розпізнає нове обладнання. Інші операційні системи також

пропонують Plug & Play функціональність і частіше вимагають ручного втручання [19].

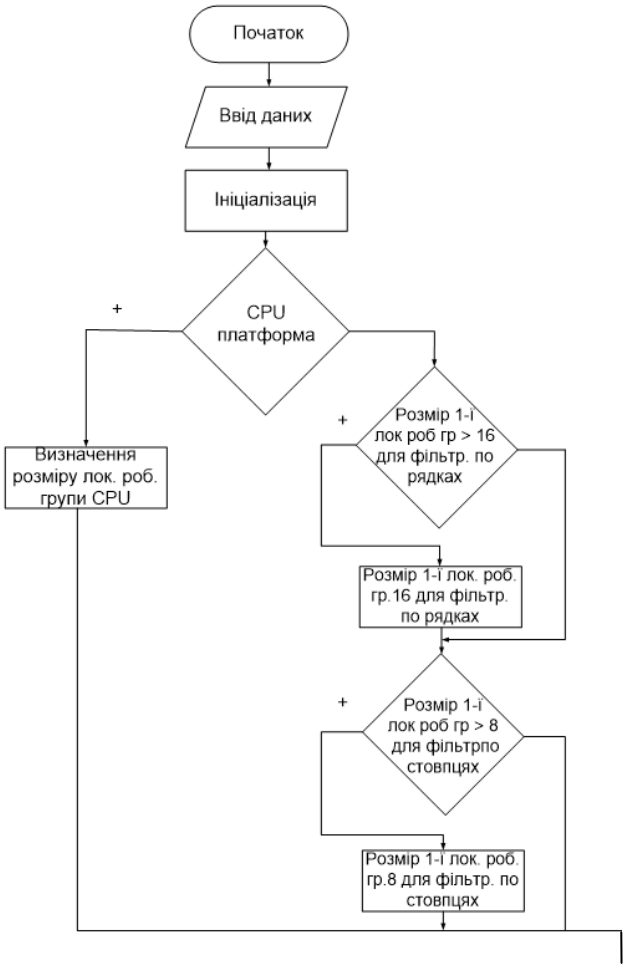
1.2.2 Побудова схеми для алгоритмічного перетворення даних

Під час розробки бібліотеки було поставлене завдання реалізувати симетричний алгоритм перетворення даних тому було проаналізовано різні алгоритми: DES, AES, Serpent, Twofish, SAFER+, RC6. За такими критеріями як надійність, швидкодія, і можливість розпаралелення, для реалізації було обрано алгоритм AES. Цей алгоритм є одним із найнадійніших і найшвидших в теперішній час, також, цей алгоритм добре «піддається» розпаралеленню, що дозволяє досягти високої продуктивності, використовуючи потужні обчислювальні засоби [20-25].

Також було завдання використати гетерогенні технології. На даний час їх є не так багато. CUDA, OpenCL, Microsoft DirectCompute. Для реалізації було обрано технологію OpenCL, оскільки вона є відкритим стандартом, а також є кросплатформовою, тобто незалежна від виробника пристрою, потрібен лише пристрій з підримкою OpenCL.

OpenCL (англійською це Open Computing Language) – є фреймворком, основним призначенням якого є створювати комп’ютерні програми, пов’язані із паралельними обчисленнями на різноманітних графічних (GPU) та центральних процесорах (CPU). До фреймворку OpenCL належить мова програмування, яка регламентується стандартом C99, та інтерфейсом призначеним для програмування комп’ютерних програм (API). OpenCL підтримує паралельність і на рівні інструкцій і на рівні данних, також являється реалізацією для технології GP GPU. Open CL є повністю відкритим стандартом, його застосування доступне на базі відкритих ліцензій. Сама ж сутність OpenCL є у

тому, щоби доповнювати OpenGL та OpenAL, які являють собою відкриті галузеві стандарти призначені для трьох вимірної комп'ютерної ґрафіки та звуку, використовуючи можливості GPU. Open CL розроблявся та в подальшому підтримується не комерційною організацією - консорціумом Khronos Group, до якого включено великі компанії, зокрема Apple, AMD, SunMicrosystems, Sony CE та й деякі інші. Дана схема для алгоритму криптографічного перетворення даних представлено на рисунку 1.3.



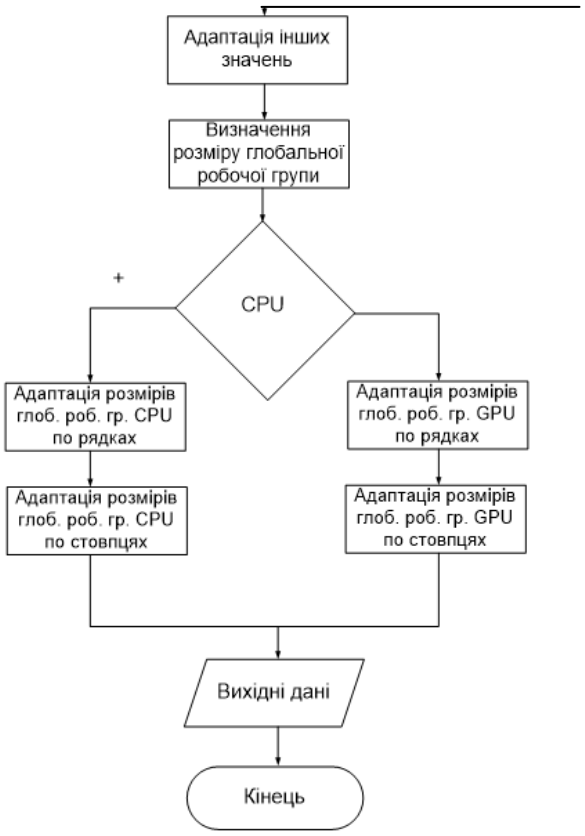


Рисунок 1.3 – Алгоритм роботи програми

AES забезпечує надійне алгоритмічне перетворення даних, а також може бути розпаралелений. Алгоритм працює з блоками розміром по 16 байт, і кожен блок обробляється незалежно один від одного, тому кожен блок можна паралельно обробляти, тим самим досягається більша продуктивність.

При роботі з бібліотекою спочатку відбувається вибір доступних OpenCL платформ. Користувач може вибрати будь-яку з доступних платформ. У випадку, коли користувач не вибрав ніякої, вибирається платформа з найбільшою кількістю обчислювальних пристроїв.

Виконання процесу прямого та зворотнього алгоритмічного перетворення даних виконується на найбільш потужному доступному обчислювальному пристрої або на пристрої, який може бути заданий користувачем. Найбільш поужним пристроєм буде той, в якого добуток кількості обчислювальних елементів на частоту роботи буде найбільшим.

Після цього всього відбуваються службові процедури для виконання прямого та зворотнього алгоритмічного перетворення даних: створення контексту, буферів, ядер, черги команд.

Керування створенням контексту, необхідних буферів даних, вміст яких буде передаватися на пристрій, виклик необхідних процедур і запуск програми на виконання на пристрої здійснюється хостовою програмою.

Всі команди, які повинні виконуватися на пристрої, записані в окремій програмі, яка містить одне чи кілька ядер. Ядро – це вхідна точка, з якої починає роботу програма.

На пристрій передається вміст буферів з даними для обробки. У моїй роботі обробка даних відбувалася на графічному процесорі (GPU). GPU має велику кількість робочих елементів, які можуть обробляти дані незалежно один від одного, тому на кожен робочий елемент передається блок даних розміром 16 байт і цей елемент здійснює обробку цього блоку за алгоритмом AES. Після обробки результат записується у певну позицію вихідного буферу. Коли всі робочі елементи завершують обробку, у вихідному буфері знаходиться результат прямого та зворотного алгоритмічного перетворення даних.

Після цього хостова програма зчитує дані з пристрою в буфер результату і ці дані можуть бути передані на подальшу обробку. Розроблена бібліотека відповідає поставленому завданню і відповідає всім вимогам. Також зроблено економічне обґрунтування для розробки бібліотеки. Нижче, на рисунку 1.4,.наведено структурну схема.

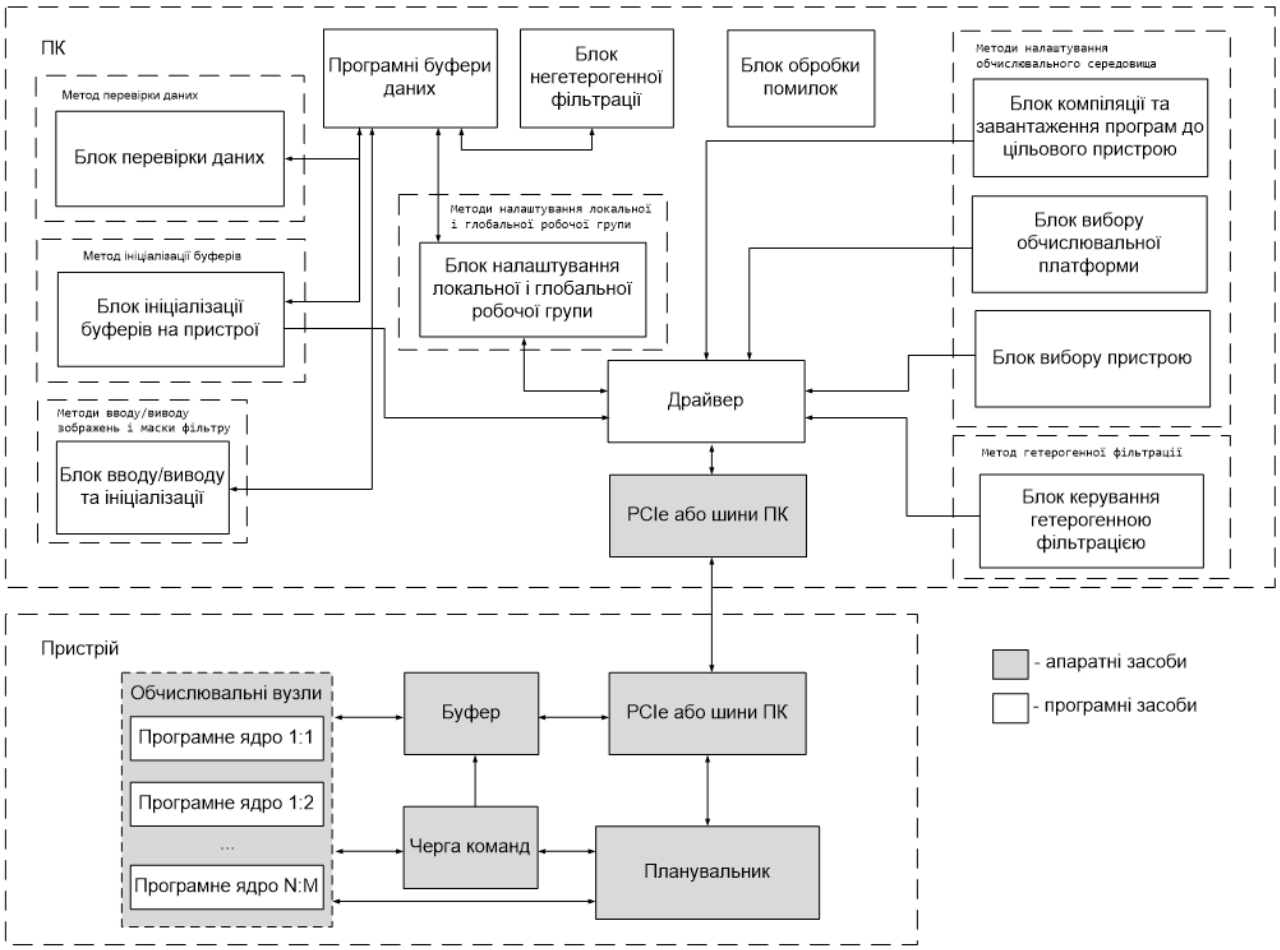


Рисунок 1.4 – Структурна схема

1.2.3 Вибір алгоритму шифрування

Для ефективного та швидкого алгоритмічного перетворення даних слід використати симетричний блочний алгоритм перетворення даних.

В симетричних алгоритмах перетворення даних один і той же ключ (який зберігається у секреті) використовується як з метою прямого алгоритмічного перетворення даних, так і для зворотнього алгоритмічного перетворення даних.

Блочний шифр є різновидом симетричної сукупності алгоритмів криптографічних перетворень, який оперує групами біт фіксованої довжини, які

називаються блоками. Якщо вихідний текст (або його лишок) менший за розмір блоку, перед алгоритмічним перетворенням його доповнюють.

Симетричні алгоритми шифрування зазвичай є швидкими, простими при реалізації, добре вивчені, і саме тому їх використання і є доцільним.

В даний час є багато надійних симетричних блочних алгоритмів, а саме:

DES [26], AES [27], Serpent [28], Twofish [29], SAFER+ [30], RC6 [31].

1) DES (англійською Data Encryption Standard) – є симетричним алгоритмом прямого та зворотного перетворення даних, тобто стандарт по шифруванню прийнятий урядом США проте, який з часом набув і міжнародного значення. Оскільки Data Encryption St. Drk.xfd засекречені елементи у своїй структурі, виникли небезпідставні побоювання з приводу можливості контролю з боку National Security Agency (в перекладі на українську Національне Агентство з Безпеки). Однак, зручність і не велика довжина ключ переважили і алгоритм затвердили як загальноприйнятий стандарт. Завдяки DES ми отримали і сучасні уявлення стосовно блочних алгоритмів шифрування та криптоаналізу.

2) AES (англійською Advanced Encryption St.), інша назва якого Rijndael – є симетричним алгоритмом т.з. блочного шифрування (з 128 бітним розміром блока, 128/192/256 бітними ключами), прийнятим урядом США за американський стандарт шифрування. Такий вибір пов’язано враховуючи широке застосування та активний аналіз алгоритму. Advanced Encryption St.оголошено стандартом шифрування у травні 2002 року.

3) Serpent є симетричним блочним алгоритм перетворення даних з 128 бітним розміром блока та 128, 192 або 256бітними можливостями довжини ключа. Алгоритм є 32-раундовою сукупністю алгоритмів криптографічних перетворень, що базується на SP-мережі, робота якого пов’язана із блоком, який складається з чотирьох 32-бітних слів. Розроблено Serpent було таким чином, щоби всі операції могли виконуватись паралельно, з використанням 32-них 1-бітних «потоків». Під час розробки Serpent-у використили більш

консервативний підхід ніж до безпеки у AES, оскільки проєктувальники шифру вважали, що для того, щоби протистояти вже відомим на той час видам криптоаналізу буде достатньо і 16 раундів, проте вони все таки збільшили число раундів аж до 32, з врахуванням можливості того, що алгоритм зможе краще протистояти ще на той час не відомим методикам криптоаналізу. Шифр Serpent не є запатентованим і вважається громадським надбанням.

4) Twofish вважається симетричним алгоритмом для виконання блочного алгоритмічного перетворення даних, з 128 бітним розміром блока, та 256 бітною довжиною ключа. Число раундів складає 16. Алгоритм було розроблено на основі таких алгоритмів як Blowfish,SAFER таSquare. Відмінностями, які відрізняють цей алгоритм від попередників є застосування попередньо обчислюваних, а також S-боксів, які залежать від ключа, а також складна схема для розгортки з метою підключення алгоритмічного перетворення даних. Тут в якості ключа шифрування використано половину n-бітного ключа, іншу ж використано в метою модифікації алгоритму, а вже від неї залежать S-бокси).

5) SAFER + - симетричний алгоритм шифрування на основі SP- мережі, являє собою поліпшений варіант алгоритмів сімейства SAFER. Алгоритм розроблений в 1998 році для участі у конкурсі на стандарт AES. Над його створенням спільно працювали фахівці з каліфорнійської корпорації Cylinc (Джеймс Мессі) та Академії наук республіки Вірменії (Гурген Хачатрян та Мельсік Курегян). У конкурсі AES алгоритм пройшов перший відбірковий тур поряд з 14 іншими алгоритмами. У фінал конкурсу, до якого допускалися лише 5 алгоритмів, SAFER + не пройшов, оскільки за результатами ретельного аналізу виявилося, що він схильний ряду атак і має низьку швидкість виконання. Алгоритм створювався для роботи на 8-бітних процесорах, і як наслідок, досить повільно працює на 32-бітних CPU.

6) RC 6 вважається симетричним блочним шифрувальним алгоритмом, який є похідним від RC 5-го алгоритму, запатентований RSA Security. Варіант

шифру як в RC 6, підтримує 128 бітні блоки та 128, 192 а також 256 бітні ключі, проте безпосередньо і сам алгоритм, так власне як і його попередник RC 5, може налаштовуватись для підтримки ширшого діапазону довжини як блоків, так і безпосередньо ключів (в межах від 0 і до 2040 біт). За своєю структурою RC 6 схожий до RC 5 і, так само, достатньо простий при реалізації. Однак, і це виявилось неочікувано для авторів, операція множення, яка повільно виконується на певному апаратному забезпеченні та дещо ускладнює саму реалізацію шифра на певних апаратних платформах із системою з архітектурою типу IntelIA-64 також реалізована не достатньо добре. Таким чином, тоді алгоритм втрачає свою перевагу (одну із) – а це висока швидкість при виконанні, це викликало критику і стало однією з перепон при обранні за новий стандарт. Однак, є системи на яких алгоритм RC 6 випереджає Rijndael, зокрема йдеться про системи із процесором PentiumII, PentiumPro, PentiumIII, PowerPC таARM.

Пряме та зворотнє перетворення даних займає значний час – більший розмір даних для шифрування забирає більше часу, тому є доцільним реалізувати шифрування з використанням паралельних технологій. В даний час найпопулярнішими технологіями є: OpenMP, CUDA,OpenCL.

1) Open MP в перекладі з англійської це Open Multi-Proc. є набором і директив компілятора і бібліотечних процедур, а також змінних середовищ, призначених для програмування безпосередньо багатопотокових застосунків на системах в яких пам’ять є спільною типу багатопроцесорні мовами: C,C++ а також Fortran. Open MP здійснює паралельні обчислення при використанні багатопотоковості, коли «головний» master-потік утворює набір залежних slave-потоків і тоді завдання розділяється на них. Враховано, що потоки реалізуються паралельно на обладнанні із кількома процесорами, при чому кількість процесорів не повинна бути більшою чи рівною до кількості потоків. Завдання, по потрібно виконати паралельними потоками, а також дані, потрібні для виконання завдань, описуються при допомозі спеціальних директив

препроцесору відповідної мови програмування. Слід зазначити й те, що кількість створюваних для виконання завдань потоків може корегуватись як і безпосередньо самою програмою при використанні виклика бібліотечних процедур, а також і ззовні, при використанні змінних середовища (оточення) [32-34].

2) CUDA в перекладі з англійської це: Compute UnifiedDevice Archit. є технологією GPGPU в перекладі з англійської це: General-purpose comp. on Graphics Proces. Units, за допомогою якої програміст має можливість реалізовувати при використанні мови С алгоритми, які будуть виконуватись при використанні графічних процесорів типу Geforce 8-го покоління і вищого: Nvidia Quadro та Tesla компанії Nvidia. CUDA є технологією розробленою компанією Nvidia є середовищем розробки на мові С, яка дозволяє створювати програмне забезпечення призначене вирішувати складні обчислювальні завдання протягом меншого часу саме завдячуючи графічним процесорам з багатоядерною обчислювальною потужністю. CUDA-технологія дає розробникові на свій розсуд організувати доступ до графічного прискорювача, а саме до набору інструкцій і керувати пам’яттю, та організувати за його допомогою паралельні обчислення високої складності. Графічний процесор з пожливостями CUDA є достатньо потужною програмованою архітектурою відкритого типу, такою як сьогоднішні центральні процесори [35].

3) Open CL в перекладі з англійської це: Open Computing Lang. є фреймворком призначеним створювати комп’ютерні програми, які пов’язані із безпосередньо паралельними обчисленнями в застосуванням різних центральних процесорів (CPU) та на процесорах графічного (GPU) типу. До фреймворку Open CL відноситься мова програмування, яка відповідає стандарту С99, а також інтерфейс призначений програмувати комп’ютерні програми. Open CL підтримує паралельність, зокрема на рівнях і інструкцій і даних та являється реалізацією для техніки GPGPU. Open CL є повністю відкритим стандартом, а його використання дозволяється з врахуванням

вільних ліцензій. Метою Open CL є доповнення Open GL і Open AL, які вважаються відкритими галузевими стандартами застосовними для трьоохвимірної комп'ютерної ґрафіки та звуку, з використанням можливостей GPU. Open CL розроблявся а також підтримується не комерційною організацією-консорціумом KhronosGroup, яка включає такі великі компанії як, наприклад, Apple, AMD,Intel, Sun Microsyst., Sony Computer Entert. та інші. Мова Open CL вирізняється тим, що дає можливість програмам в динаміці визначати, які процесори є доступними, зокрема включаючи центральні багатоядерні та графічні процесори. Це дає змогу динамічно збільшувати продуктивність програм, проте залежить від наявного апаратного забезпечення на стороні клієнта. Актуальною тут є і розробка бібліотеки оскільки пряме та зворотнє алгоритмічне перетворення даних відіграє значну роль в інформаційних сферах, а бібліотека підтримує швидке та надійне шифрування та дешифрування даних [36].

Таким чином алгоритмічне перетворення даних, підходи та способи, мають як свої переваги так й недоліки, проте, проаналізувавши я зробив висновок про те, що найдоцільнішим буде прийняти алгоритм на базі шифра Rijndael – Advanced EncryptionStand. Він широко використовується і є одним із найпопулярніших симетричних алгоритмів шифрування, також він є безпечним – ще не було вдалих атак на цей алгоритм. AES є досить швидким алгоритмом і піддається розпаралелюванню.

Основні переваги AES:

1) висока надійність;

2) висока швидкість в апаратних та програмних засобах;

3) підтримка великих розмірів ключів шифрування – це робить його менш відкритим до атак на основі парадоксу днів народжень.

Зваживши оптимальність вибору, я дійшов висновку про те, що завдяки вище названим перевагам вибір цього алгоритму для шифрування є обгрунтованим та доцільним.

1.2.4 Вивчення та аналіз алгоритму AES, AddRoundKey

Advanced Encryption Stand.(скорочено AES) ще так само відомий за назвою Rijndael, є специфікацією основним призначенням якої є шифрування електронних даних, затверджена Національним інститутом стандартів та технологій в 2001 році. Робота AES здійснюється на базі шифра Rijndael, розробленого двома бельгійцями-криптографами. Rijndael є сімейством шифрів з ключами різної розмірності та блоків даних.
Для AES, NIST вибрали три види Rijndael, кожен із 128-ми бітною розмірністю блоку та із трьома різної розмірності, а саме: 256-ти, 192 та 128-ми бітними довжинами ключа [37].

AES вважається симетричним алгоритмом блочного алгоритмічного перетворення даних, і це значить, що один ключ використовують для прямого та зворотного алгоритмічного перетворення даних, а дані для шифрування розбиваються на блоки. Він базується на основі принципу проектування, відомому як мережа заміни-перестановки (в перекладі на англ. Substitution-Permutation(SP) Net.) – поєднання заміни і перестановки, що є швидким в програмному і апаратному забезпеченні. На відміну AES – варіант Rijndael, з фіксованим 128 бітним розміром блоку та 128,192 чи 256-ти бітними розмірами ключа. Також, у специфікації Rijndael зазначено алгоритм із змінними розмірами блоку і ключа, які можуть бути з кратністю 32 біта та довжиною в межах від 128 і до 256біт.

Для AES характерне оперування масивом розмірності 4×4 байт, та називається станом (в перекладі на англійськуState) [38].

Розмір ключа, який використовується для шифру AES, визначає кількість раундів, які перетворюють вхідні (відкриті) дані в зашифровані дані. Кількість раундів для різних довжин ключа:

1-10 раундів для ключів розмірністю 128 біт;

2-12 раундів для ключів розмірністю 192 біт;

3-14 раундів для ключів розмірністю 256 біт..;

Кожен раунд складається із кількох етапів опрацювання, кожен з яких містить чотири різні етапи, у тому числі той, який залежить від самого ключа шифрування. Набір зворотних раундів застосовують для зворотного перетворення зашифрованого тексту, тобто до вигляду вихідного відкритого тексту, при використанні того же ключа для шифрування.

Блок-схему алгоритму для прямого та зворотного алгоритмічного перетворення даних при застосуванні алгоритму AES із 128-ми бітною довжиною ключа представлено на рисунках 1.5, 1.6.

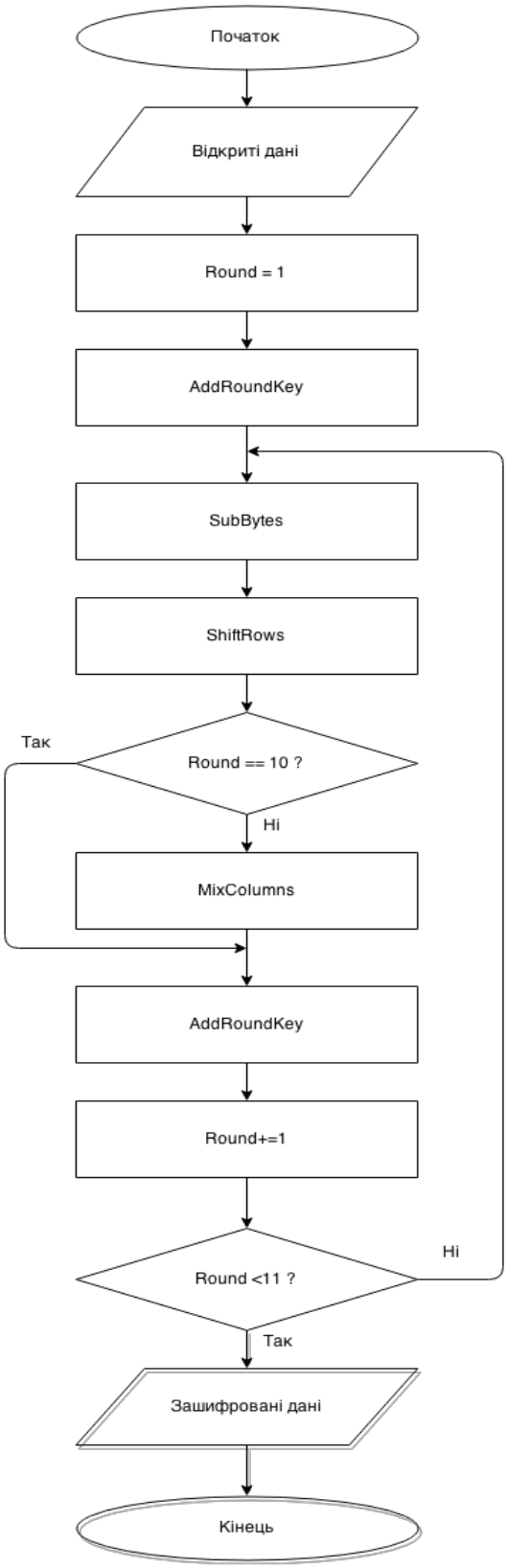


Рисунок 1.5 – Схема прямого та зворотного алгоритмічного перетворення даних при використанні алгоритма AES

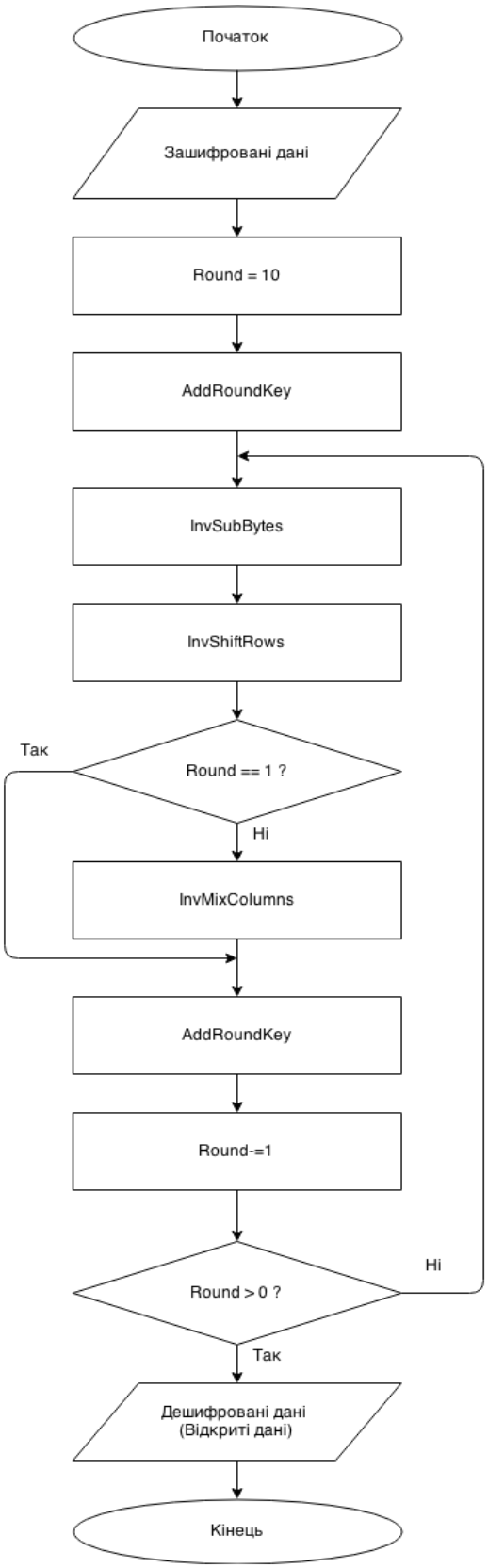


Рисунок 1.6 – Схема дешифрування з використання алгоритму AES

AddRoundKey є процедура, в якій відбувається об’єднання раундового ключа із станом. Для кожного раунду, раундовий ключ є похідним від основного ключа, використовуючи алгоритм основною задачею якого є розширювання ключа Rijndael (Rijndael KeyExpansion). Кожен раундовий ключ такого ж розміру, як стан. Об'єднання кожного байту стану з відповідним байтом ключа відбувається з використанням побітового XOR.

Алгоритм основною задачею якого покладено обробку ключа складається із 2-х основних процедур:

1-алгоритму розширювання ключа.

2-алгоритму вибору раундового ітераційного ключа.

При допомозі алгоритму по розширенню ключа (AES), при використанні процедури KeyExpansion і подаючи до неї Cipher Key (так званий секретний ключ), я зможу отримати ключі для усіх раундів [39].

При реалізації алгоритму призначенням якого є вибір раундового ключа під час кожної ітерації раундовий ключ AddRoundKey обирається із цілого їх масиву.

При здійсненні процедури прямого та зворотного алгоритмічного перетворення даних процедурно обираються різні раундові ключі. Для процедури шифрування раундові ключі обираються із масиву ключів від першого і аж до останнього, а вже при дешифруванні порядок зворотній[40].Дану процедуру представлено нижче на рис. 1.7.

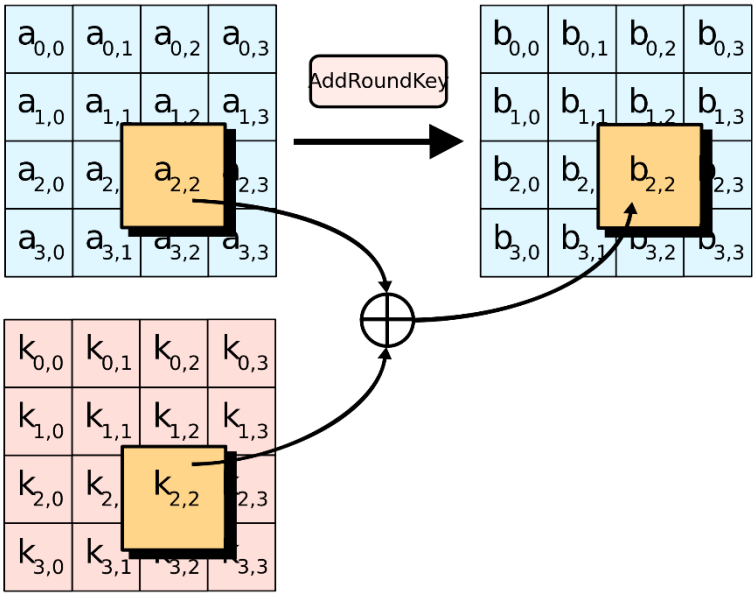


Рисунок 1.7 – Процедура вибору раундового ключа (під час кожної ітерації раундовий ключ AddRoundKey обирається із цілого їх масиву).

1.2.5 Аналітичний огляд SubBytes, InvSubBytes, ShiftRows, InvShiftRows, MixColumns, InvMixColumns

Для процедур таких як SubBytes та InvSubBytes властиво, що для кожного байту стану AES здійснюється заміна відповідним елементом згідно фіксованої 8-бітної таблиці для пошуку відповідно S-box чи InvS-box. SubBytes та InvSubBytes процедури представив нижче, на рисунку 1.8.

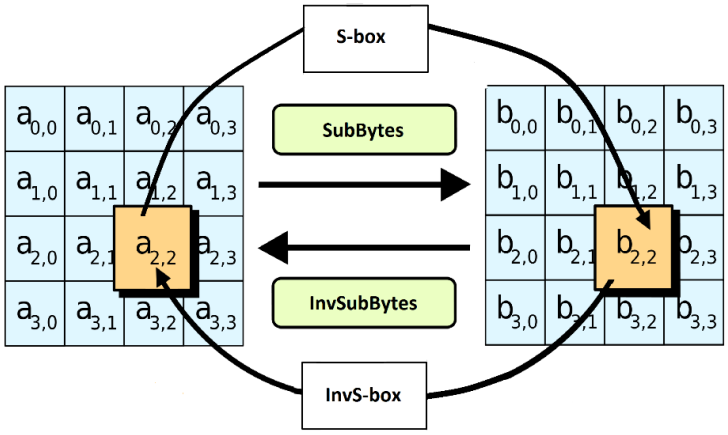


Рисунок 1.8 – Представлення процедур SubBytes і InvSubBytes

Процедурами незалежно опрацьовується кожен байт стану, через нелінійну заміну байтів при використанні таблиць для замін S-box та InvS-box. Такою операцією забезпечується нелінійність алгоритму, що дає змогу мінімізувати ризик або ж уникнути атак з логікою простих алгебраїчних властивостей, що є, фактично, звичайним шифром простої підстановки. Таблиці простої підстановки для S-box та InvS-box представлено на рис. 1.3 нижче.

Таблиця 1.3 – Таблиця шифру для простої підстановки S-box

S-box (SubBytes Transformation Table)

\	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Таблиця 1.4 – Таблиця шифру простої підстановки для InvS-box

InvS-box (InvSubBytes Transformation Table)															
\	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Процедури ShiftRows та InvShiftRows працюють в рядках стану - залишається без змін. Кожен байт другого рядка зсувається один вліво. Аналогічним чином, третій і четвертий рядки зсуваються на 2 і 3 байти відповідно. Для блоків розмірами 128 біт і 192 біт зсув відбувається одинаково. Для 256-бітного блоку, перший рядок залишається незмінним, а зсув для другого, третього і четвертого відбувається на 1 байт, 3 байти і 4 байти відповідно. Цей крок призначений для того, щоб уникнути лінійної залежності стовпців. Фактично це проста перестановка байт таблиці стану 4x4. Дані процедури можна переглянути на рисунку 1.9.

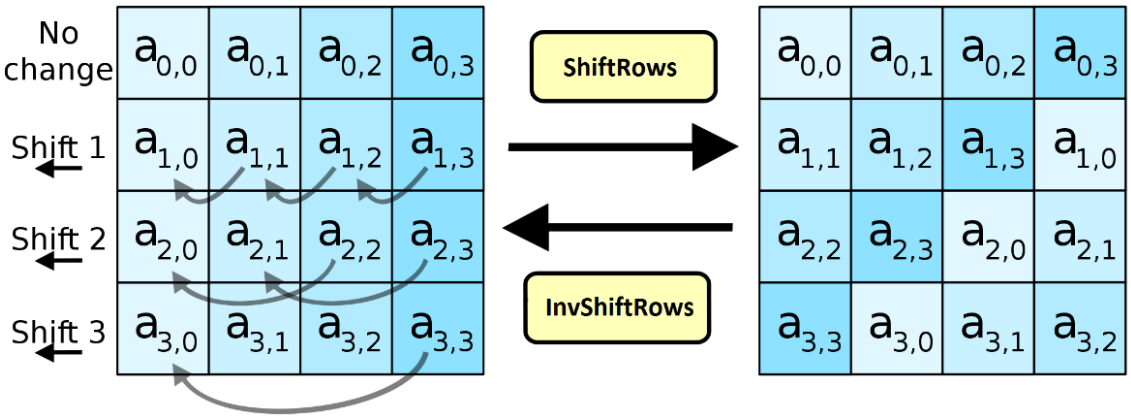


Рисунок 1.9 – Процедури ShiftRows та InvShiftRows

Суть процедур MixColumns та InvMixColumns полягає у тому, що 4 байти кожного стовпця стану об’єднують при допомозі зворотного лінійного перетворення. MixColumns та InvMixColumns «приймають» 4 байти на вході і передають на вихід 4-ри байти, і кожен вхідний байт впливає на всі 4-ри вихідні байти. Разом з ShiftRows та MixColumns забезпечують дифузію в шифрі.

Таким чином, при виконанні цієї операції, кожен стовпець множать на зафіксовану матрицю типу:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}.$$

Множення матриць складає множення та додавання записів. Таким чином операцію множення можна визначити наступним чином: при множенні на 1 – зміни відсутні, при множенні на 2 здійснюється зсув вліво, а при множенні на 3 – зсув вліво та виконання операції XOR з початковим значенням. Після зміщення, повинена бути виконана умовна операція XOR з 0x1B, якщо зміщене значення більше, ніж 0xFF. Роботу процедур MixColumns та InvMixColumns представлено на рисунку. 1.10.

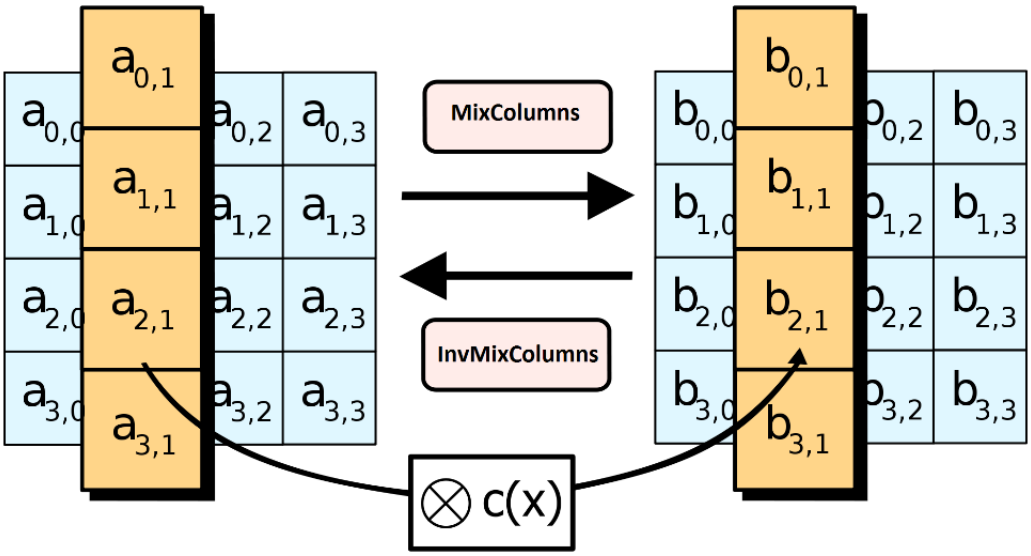


Рисунок 1.10 – Робота процедур MixColumns та InvMixColumns.

1.2.6 Розробка структурної схеми підсистеми

Відповідно до поставленої задачі була розроблена структурна схема гетерогенної реалізації симетричних алгоритмів прямого та зворотного алгоритмічного перетворення даних. Дану структуру можна переглянути на рисунку 1.5.

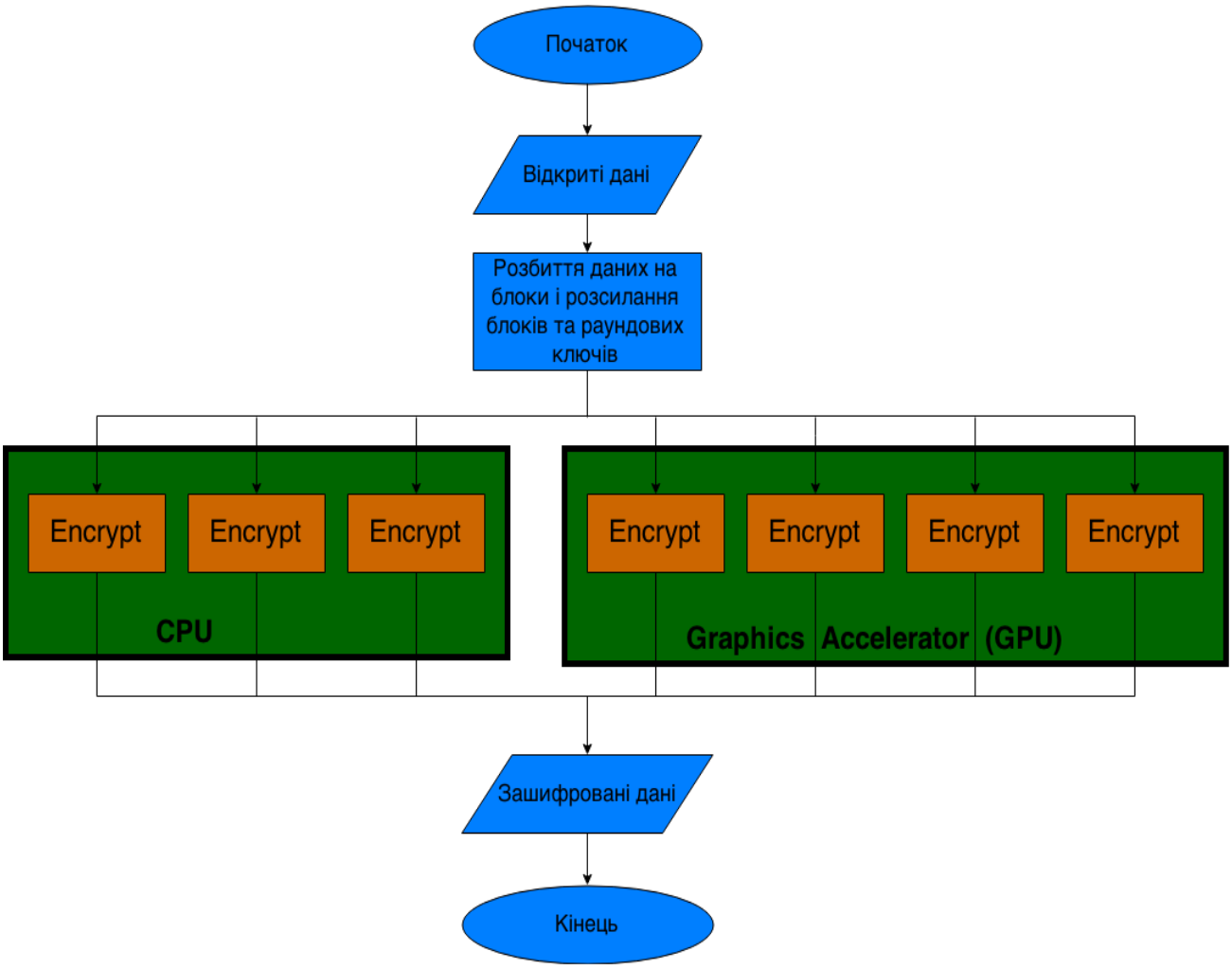


Рисунок. 1.11 – Структурна схема гетерогенної реалізації симетричних алгоритмів прямого та зворотного алгоритмічного перетворення даних

CPU (англійською повністю Central Processing Unit) – центральний процесор, виконує паралельне шифрування даних алгоритмом AES.

Graphics Accelerator є графічним процесором (англійською повністю Graphics Processing Unit), який виконує паралельне шифрування даних алгоритмом AES. Основною перевагою над CPU є те, що графічний прискорювач має набагато більше обчислювальних елементів і за той самий час зашифрує набагато більше даних ніж цетральний процесор.

Encrypt – блок, в якому відбувається шифрування за алгоритмом AES.

На вхід подаються відкриті дані, які розбиваються на блоки з фіксованою довжиною і разом із раундовими ключами подаються на кожен блок Encrypt для паралельного шифрування. Результат – зашифровані дані.

1.2.7 Побудова UML – діаграми діяльності

Під час процесу моделювання поведінки проєктованої чи аналізованої системи у мене виникла необхідність не тільки уявляти процес зміни станів системи, але й потреба в деталізації особливостей алґоритмічної та логічної реалізацій операцій, які та система виконує, тобто йдеться про те, що для отримання бажаного результату виконання послідовності дій чи елементарних операцій. Традиційно з цією метою використовують блок-схеми чи структурні схеми алґоритмів. Традиційним підходом діяв і я при виконанні моєї магістерської роботи.

Для того, щоби змоделювати процес виконання операцій мовою UML використовують так звані діаграми діяльності, які подібні графічно до нотації діаграм станів, оскільки так само містять позначення як станів так і переходів. Проте, є й відмінність, яка полягає саме у семантиці станів, які використовують для подачі не самих діяльностей, а власне дій, та у відсутності, на переходах, сиґнатури подій. Важливо також, що кожен зі станів представлених діаграмою діяльності відповідає за виконання певної з елементарних операцій, при цьому здійснення переходу до наступного стану можливе лише під час завершення даної операції у попередньому стані. Графічне представлення діаграми діяльності представляється у вигляді графу діяльності з вершинами станами дії та дугами – переходами від стану дії і до іншого стану дії.

Отже, підсумовуючи, можу сказати, що діаграму діяльності варта вважати за окремий випадок діаграми станів, і за допомогою них є можливість реалізації

в мові UML особливостей процедурного та синхронного управління, яке обумовлюється завершенням внутрішніх діяльностей та дій. А потрібні для цього терміни і семантику вже беремо із метамоделі UML.

В термінах мови UML діяльність представляє собою певну сукупність деяких окремих обчислень, які виконуються автоматом. Застосування певних елементарних обчислень при цьому може призвести до певної дії (action) чи результату. На діаграмі діяльності представляється логіка або послідовності переходів від одного вида діяльності до другого, але увага фокусується на результат діяльності, який може призводити до змін в стані системи або ж поверненню деякого певного значення.

Стан дії (англійською звучить як action state) прийнято вважати за спеціальний випадок стана з деякими вхідними діями і щонайменше одним переходом. Цим переходом неявно робиться припущення про завершення вхідної дії. Важливо розуміти також те, що action state не може мати ніяких внутрішніх переходів, оскільки вважається за елементарний. Суть звичайного використання action state - моделювання 1-го кроку здійснення алгоритм / процедури чи потока керування.

В графічному представленні стану дії є фігура що нагадує прямокутник, але бічні сторони цього «прямокутника» замінено на опуклі дуги. Всередину цього представлення записується action - expression (в перекладі з англійської звучить як вираз дії), який є унікальним в рамках однієї діаграми діяльності.

При побудові діаграми діяльності використовують лише ті переходи, які можуть спрацювати одразу після її завершення чи ж виконання відповідної дії. Такий перехід переведе діяльність в наступний стан одразу, як тільки скінчиться дія попереднього стану. Якщо звернути увагу на діаграму, то перехід представляється у вигляді суцільної лінії зі стрілкою.

Розгалуження на діаграмі діяльності графічно позначається ромбом до якого може від стану дії може заходити лише єдина стрілка, і після реалізації якого потік керування продовжиться за однією із гілок. Виходів стрілок може

бути від двох і більше, проте обов’язковим є вказання для кожного виходу на відповідну сторожову умову за формою булівського виразу.

Отже, можу сказати, що головним призначенням діаграми діяльності є демонстрація того які дії користувач може виконувати програмно.

Таким чином, основні дії, які користувач може виконати наступні:

1- запуск (програми);

2- вибір (Open CL платформи);

3- створення)контексту та буферів);

4- здатність (зашифровувати дан)і;

5- здатність (розшифровувати дані);

6- перегляд готового результату.

Вигляд діаграми діяльності представлена на рисунку 1.12.

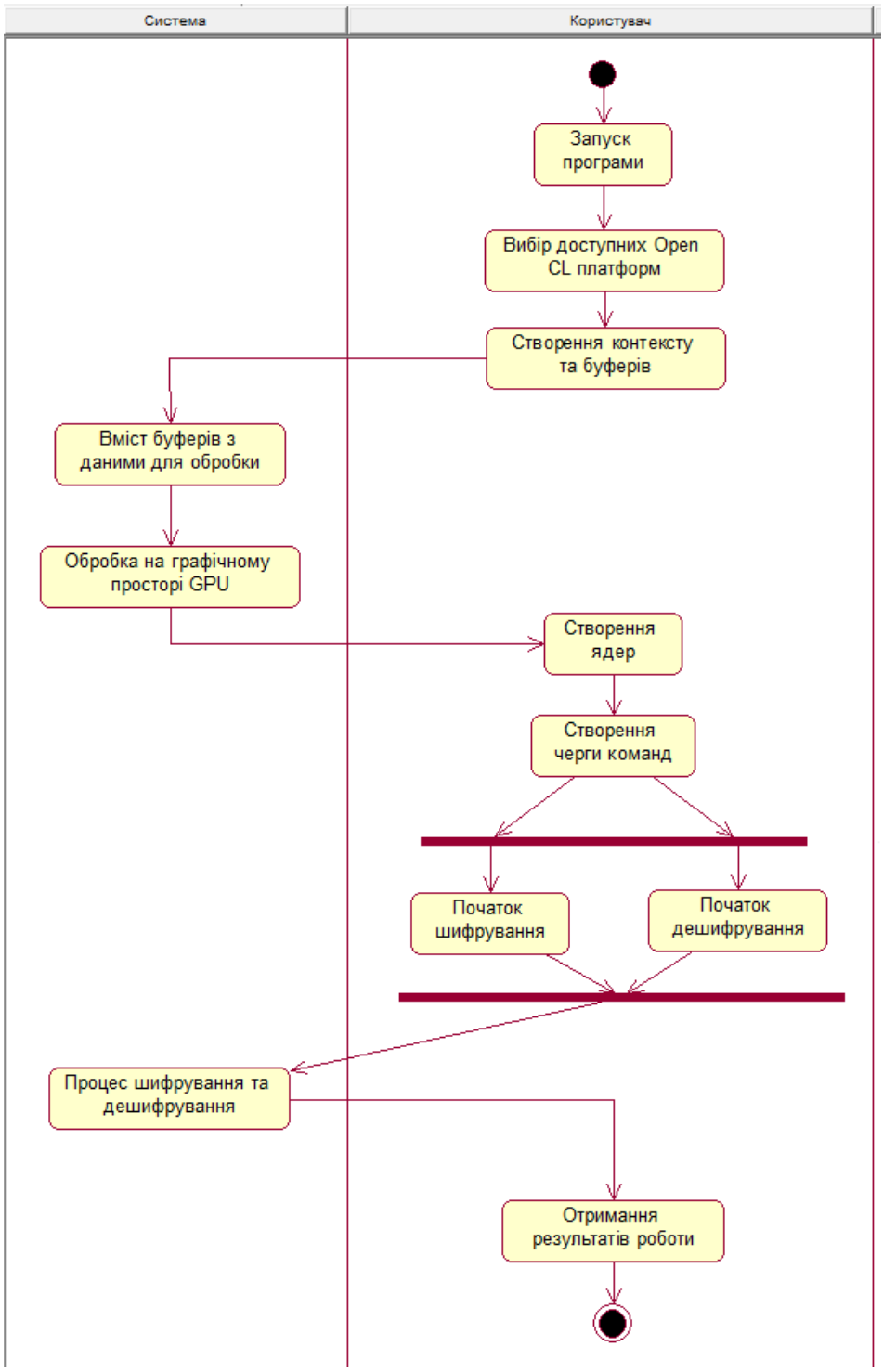


Рисунок 1.12 – Вигляд діаграми діяльності

На рисунку відображено можливі дії для користувача у програмі. Отже після початку роботи вхід у програму, після якого користувач обирає доступні для нього Open CL платформи щоб далі працювати в програмі. Наступний крок

системи є обробкою вмісту даних та їх подача до обробки на графічному процесорі GPU, після чого здійснюється перехід до створення черг команд, тоді користувач обире один із запропонованих йому способів алгоритмічного перетворення даних(шифрування/дешифрування). Після всіх вище описаних послідовностей кроків дані опрацьовуються системою після завершення якого користувач отримає вже готовий кінцевий результат.

1.3 Конструювання програмної системи

1.3.1 Вибір мови програмування та розробки програмної підсистеми

Існує багато хороших і перевірених мов програмування, кожна має свої переваги і недоліки. На сьогодні, зробивши аналіз, я дійшов висновку що найпопулярнішими з ним є C/C++, Java, C#.

Отже, наведу частину свого аналізу, який дозволив прийняти оптимальне рішення.

C# вважається об’єктно-орієнтованою мовою програмування в якій передбачено безпечну систему типізації під платформу .NET.

Якщо говорити стосовно синтаксису, то він є близьким до синтаксису таких мов як C++ та Java. C# є мовою яка володіє строгою статичною типізацією, з підтримкою поліморфізму, можливістю перевантажувати операторів, вказівники на функції, які є членами класів, атрибутів, подій, властивостей, винятків, коментарів у форматі XML. В C#, ґрунтуючись на практику використання мов попередників, виключено моделі, які, при розробці програмних систем, зарекомендували себе як проблемні. Зокрема йдеться про множинне наслідування класів.

C# розроблялась в якості мови для програмування на прикладному рівні для CLR і саме тому зрозумілою є її залежність від можливостей CLR, зокрема

стосовно, перш за все, системи типів мови C#. А наявність чи ж відсутність якихось виразних особливостей в мові пояснюється можливістю конкретної мовної особливрсті транслюватись до відповідних конструкцій CLR. Природно також, що з пргресуванням CLR з версії до версії значно покращилася сама мова C#. CLR надається C# велика кількість можливостей, які відсутні у так званих «класичних» мовах програмування. [44].

Таким чином, проаналізувавши, я виділив наступні переваги:

1- C # розроблено на базі існуючих Java і C ++, з врахуванням надмірної заплутаності та помилковості попередників..

2- написання програмина мові C # передбачено в простому текстовому редакторі та командному рядку. Вони є загальноприйнятими для будь-якої з існуючих операційних систем при умові, що розробник встановив CLR та.NET Framework.

3- для розробників передбачено можливість для розширення кода, а отже це дає їм можливість на створення розширення та обгортки.

4- C # об'єктно-орієнтована мова,і це дає можливість розробляти безпечні додатки.

5- передбачено автоматичне звільнення пам'яті, тому розробник не повинен це робити в ручному режимі.

6- використано концепцію збірок, що дає можливість вирішити проблему керування версіями.

7- мова є достатньо легкою для розробки, оскільки має багату бібліотеку класів, за допомогою якої реалізується багато зручних функцій.

8- передбачено підтримку розподілених систем.

Microsoft Visual Studio є інтегрованим середовищем призначення якого є розробка(IDE) від Microsoft. Зазвичай використовується при розробці комп'ютерних програм під Microsoft Windows, так само як і web-сайтів, web-додатків та web-сервісів. В Visual Studio використовуються платформи для розробки програмного забезпеченняMicrosoft, зокрема: WindowsAPI,

WindowsForms, Windows Presentation Found., Windows Store і MS Silverlight. Може виконувати як машинний, так і керований код [45].

Visual Studio підтримує різні мови програмування. Вбудованими є мови C, C ++ і C ++ / CLI (Visual C ++), VB.NET(Visual Basic.NET), C # (Visual C #), і F # . Підтримка інших мов, таких як Python, Rubi доступні через мовні служби, які встановлюються окремо.

Це середовище розробки є зручним і має дуже широкі можливості. На даний час немає кращого середовища для розробки мовами C, C++ і C#.

Для розробки інтерфейсу програмної системи я використав такі технології:

1- Технологія WPF – є графічною (презентаційноюа) підсистемою, яка входить до складу.NET Framework версії 3.0, та має безпосереднє відношення до XAML. WPF заразом із .NET Framework версії 3.0 вбудована до Wind.Vista, ну і є також доступною для установки у WindowsXP ServicePack2 і WindowsServer2003[46].

Від часу випуску Windows95 є реальним першим оновленням для технологічного середовища, призначенням якого є робота для користувача інтерфейсу. Оновлення включає в себе нове ядро, яке повинно замінювати GDI та GDI +, які використовують сьогодні на Windows - платформі. WPF вважається за високорівневий функціональним шаром з об’єктно-орієнтованим підходом - фреймворком, за допомогою якого можна створити 2-вимірні та 3-вимірні інтерфейси.

По суті своїй WPF є з векторною системою візуалізації, що є незалежною від дозволів пристрою виводу, створеною із врахуванням можливості сучасного ґрафічного устаткування. WPF дає засоби та можливість створювати візуальний інтерфейс долучаючи мову XAML (повністю англійською Extensible Application Markup Lang.), так само як і елементи керування, можливість прив’язки даних, макети, 2-вимірну та 3-вимірну графіку, також передбачена анімація, стилі, шаблони, документи, тексти, мультимедіа і оформлення.

Графічна технологія на якій базується робота WPF це DirectX, відмінність якої від Windows Forms, в якій використовуються GDI/GDI+. Продуктивність WPF вища, ніж у GDI + за рахунок використання апаратного прискорення графіки через DirectX [47].

2- На сьогодні на ринку є зовсім не велика кількість технологій з принципом гетерогенного програмування, саме тому мій вибір буде не надто складним. Отже, одною із найбільше поширених гетерогенних технологій вважається Open CL (англійською повністю Open Computing Lang.), яка є фреймворком призначеним безпосередньо для створення комп'ютерних програм, які пов'язані з паралельними обчисленнями на різних графічних (GPU) і центральних процесорах (CPU). До фреймворку Open CL входить мова програмування, що підтримується стандартом C 99, і інтерфейс для програмування комп'ютерних програм(API) [48].

Метою Open CL полягає в доповненні Open GL та Open AL, які являються відкритими ґалузевими стандартами для 3-вимірної комп'ютерної графіки та звуку, при використанні можливостей GPU.

Перевагою Open CL є те, що при його використанні значно підвищується швидкість для великої кількості застосувань з різних ринкових категорій – від ігор та розваг аж до наукового та медичного програмного забезпечення.

Основними перевагами цієї технології є:

1) OpenCL в рази прискорює виконання коду. Багато вбудованих або настільних обчислювальні платформи мають GPU на борту. Він перебуває ненавантаженим більшу частину часу – інтенсивно працює лише при відображенні певної графіки. При простої GPU можна змусити код працювати швидше.

2) Open CL є відкритим стандартом. Це означає, що кожен, хто хоче використовувати OpenCL має вільний і відкритий доступ до стандарту. OpenCL має повний набір високоякісної документації, в тому числі онлайн підручники,

керівництва з програмування та довідкових посібників. Є багато інструментів із вихідним кодом відкритого типу.

3) Open CL може бути використаний з різними мовами програмування. OpenCL ядра можуть бути викликані з мов, таких як C, C ++, Java, Python, JavaScript, Perl, Haskell, Ruby, ... і список зростає. Це дозволяє забезпечити переносимість і повторне використання існуючих ядер OpenCL, що забезпечує гнучку підтримку GPGPU обчислень в цілому ряді середовищ розробки.

4) Open CL є незалежним від платформи. OpenCL представляє платформу незалежної модель програмування, яка дозволяє багатоядерну гетерогенну розробку програмного забезпечення. Різні простору пам'яті визначити модель узгодженості пам'яті. Низькорівневі завдання, такі як черги і виконавчі ядра, захищені від користувачів OpenCL під час виконання. Примітиви, такі як події, бар'єри забезпечують можливості синхронізації. Це означає, що OpenCL код буде працювати без змін на OpenCL сумісних платформах. Як наслідок, навіть якщо подання на різних платформах можуть відрізнятися, перенесення реалізації з однієї платформи на іншу спрощується. OpenCL забезпечує функціональну точку відліку на основі загальної моделі системи.

Тому OpenCL виходить з найбільш загальних передумов, що дають уявлення про пристрій з підтримкою OpenCL: так як цей пристрій передбачається використовувати для обчислень – в ньому є якийсь «процесор» в загальному розумінні цього слова, назвемо його «клієнт». Щось, що може виконувати команди. Так як OpenCL створений для паралельних обчислень, то такий процесор може, мати кошти паралелізму всередині себе (наприклад, кілька ядер одного CPU, кілька SPE процесорів в Cell). Також елементарним способом нарощування продуктивності паралельних обчислень є установка декількох таких процесорів на пристрої (наприклад, багатопроцесорні материнські плати PC і т.д.). І природно в гетерогенній системі може бути декілька таких OpenCL-пристроїв (взагалі кажучи, з різною архітектурою).

Крім обчислювальних ресурсів, пристрій має якийсь обсяг пам'яті. Причому ніяких вимог до цієї пам'яті не пред'являється, вона може бути як на пристрої, так і взагалі бути розмічена на ОЗУ хоста (як наприклад, це зроблено у вбудованих відеокарт). Таке широке поняття про пристрій дозволяє не накладати жодних обмежень на програми, розроблені для OpenCL. Ця технологія дозволить розробляти як додатки, сильно оптимізовані під конкретну архітектуру специфічного пристрою, що підтримує OpenCL, так і ті, які будуть демонструвати стабільну продуктивність на всіх типах пристроїв (за умови еквівалентної продуктивності цих пристроїв).

OpenCL надає програмісту низькорівневий API, через який він взаємодіє з ресурсами пристрою. OpenCL API може або безпосередньо підтримуватися пристроєм, або працювати через проміжний API (як у випадку NVidia: OpenCL працює поверх CUDA Driver API, підтримуваний пристроями), це залежить від конкретної реалізації не описується стандартом.

Головним конкурентом OpenCL для проведення обчислень на графічних ядрах є технологія CUDA від NVIDIA.

CUDA (англійською повністю Compute Unified Device Architecture) – технологія GPGPU (англійською повністю General-purpose computing on Graph. Proces. Un.), що дозволяє програмістам реалізовувати мовою програмування С алгоритми, що виконуватимуться на графічних процесорах Geforce восьмого покоління і вище (англ. Geforce 8 Series, Geforce 9 Series, Geforce 200 Series), Nvidia Quadro і Tesla компанії Nvidia [49].

Технологія CUDA – це середовище розробки на С, яке дозволяє програмістам і розробникам писати програмне забезпечення для вирішення складних обчислювальних завдань за менший час завдяки багатоядерній обчислювальній потужності графічних процесорів. Простіше кажучи, графічна підсистема комп'ютера з підтримкою CUDA може бути використана, як обчислювальна.

Дві технології є досить потужними для реалізації складних обчислень, але CUDA має два основні недоліки, у порівнянні з OpenCL.

1) CUDA призначена для виконання обчислень лише на графічних ядрах, в той час як OpenCL є гетерогенною і може виконувати обчислення як на графічних ядрах, так і на ядрах центрального процесора.

2) CUDA підтримується лише відеоадаптерами фірми NVIDIA, тому нею не можуть скористатися користувачі з адаптером іншої фірми. OpenCL є кросплатформовою технологією [49].

Проаналізувавши всі недоліки і переваги кожної з технологій, для розробки було обрано технологію OpenCL.

1.4 Використання програмної системи

1.4.1 Системні вимоги

Отже в результаті проведеного дослідження я визначився, що для коректної роботи програми потрібне наступне програмне забезпечення забезпечення:

1-Операційна система типу Microsoft Windows версій 7/8/8.1.

2-MS.NET Framework версії 4.5 та вище.

Однак, Windows має один суттєвий, на мою думку, недолік – його вартість. Ціна цієї операційної системи є досить висока, але враховуючи ряд переваг, доцільно буде вибрати цю ОС для реалізації бібліотеки гетерогенного шифрування даних.

2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Тестування процесора

Дана бібліотека дозволяє задати платформу та пристрій, на якому будуть проводитися обчислення. Але якщо користувач не задав цих параметрів, то бібліотека визначає платформу і пристрій, на яких обчислення будуть проводитися найбільш швидко [50,51].

За допомогою класу ComputePlatform можна динамічно отримувати та задавати платформи для виконання обчислень. За допомогою властивості ComputePlatform.Platforms отримується список всіх доступних платформ з підтримкою OpenCL. Кожна платформа має певну кількість пристроїв. Найпотужнішою платформою буде та, яка має найбільшу кількість пристроїв, бо всі пристрої можуть бути задіяні в обчисленнях, тому обчислення будуть проводитися швидше.

Після вибору платформи потрібно вибрати найбільш потужний пристрій. Можна обрати всі пристрої, які доступні в обраній платформі, але тоді їх треба рівномірно навантажити, а для цього немає якогось універсального рішення. До платформи входять пристрої різного типу: графічні і центральні процесори, прискорювачі. Всі вони відрізняються між собою за призначеннями та можливостям, рівномірно навантажити всі ці пристрої є досить трудомістким процесом, тому доцільніше вибрати один з цих пристроїв, який є найпотужнішим і найшвидшим.

Вибрати найпотужніший пристрій також нелегко. Одним з найпопулярніших способів вибору є вибір такого пристрою, в якого добуток частоти на кількість обчислювальних елементів буде найбільшим (добуток продуктивності).

Але цей спосіб має значний недолік – може бути так, що в системі, в якій присутні центральний і відеопроцесор, потужнішим вважатиметься

центральний процесор, оскільки добуток продуктивності в нього більший. Незважаючи на те, що цей добуток і є більший, це не означає, шо обчислення будуть проходити швидше. Архітектури центрального і відеопроцесора дуже відрізняються.

Графічний процесор призначений для значного розпаралелення завдань і містить багато процесорних елементів, які проводять обчислення – набагато більше, як центральний процесор. В такому разі, буде обрано не найпотужніший пристрій.

При виборі пристрою, бібліотека спочатку перевіряє чи є в системі графічні процесори. Якщо GPU присутні, то з них вибирається вже той, в якого добуток продуктивності найвищий. Якщо GPU немає, то тоді відбувається перевірка, чи є прискорювачі, і при їх наявності, також обирається найбільш потужний. Якщо немає і прискорювачів, то тоді пристроєм буде процесор. Якщо процесорів кілька, також вибирається найпотужніший.

Цей підхід вибору є досить ефективним і в плані розвантаження центрального процесора. Якщо є можливість вибрати не CPU, то буде обрано інший пристрій, а центральний процесор зможе виконати більш загальні завдання, чи ті, які не підлягають розпаралеленню.

2.2 Реалізація ядра (kernel)

Алгоримт шифрування AES працює з блоками довжиною 16 байт. OpenCL має підходящий тип даних – uchar16, тому буде доцільно використати цей тип даних.

Ядро (kernel) – це вхідна точка для пристрою – пристрій починає виконання з цієї функції. Після створення фунції-ядра, вона може бути відправлена в чергу команд для будь якого пристрою. Кожне ядро має певні

аргументи, з якими буде працювати певний пристрій. В ядро передаються 4 аргументи: буфер з вхідними даними (inputData), буфер з раундовими ключами (keys), кількість раундів алгоритму (gRounds) і буфер, в який будуть записуватися оброблені дані (result).

При виконанні функції – ядра, кожен робочий елемент (work item) пристрою отримує певний набір даних, які займають 16 байт і обробляє їх. Після обробки, ці дані записуються в певну позицію буферу – результату. Після того, як всі робочі елементи завершили свою обробку, на виході буде буфер з зашифрованими/дешифрованими даними.

За допомогою функції get_global_id()кожен робочий елемент отримує свій індекс. За допомогою цього індексу він дістає певний набір даних з вхідного буферу.

Також в ядрі присутні звертання до змінної gRounds. Ця змінна знаходиться в глобальній памяті, тому доступ до неї буде повільним. Оскільки ця змінна використовується в циклі, і доступатися до неї потрібно неодноразово, то буде доцільним створити приватну змінну для робочого елемента. Доступ до приватних змінних відбувається значно скоріше, ніж до глобальних, тому це призведе, до виграшу в продуктивності (не дуже значного, але виграш буде).

2.3 Тестування бібліотеки

Для того, щоб протестувати розроблену бібліотеку, був розроблений графічний інтерфейс, який надає всі необхідні можливості для тестування: ввід тексту, вибір файлів для шифрування/дешифрування, можливість вибору способу шифрування – використовувати OpenCL чи ні. Також поле з

результатом обробки певних даних. Графічний інтерфейс зображено на рисунку 2.1.

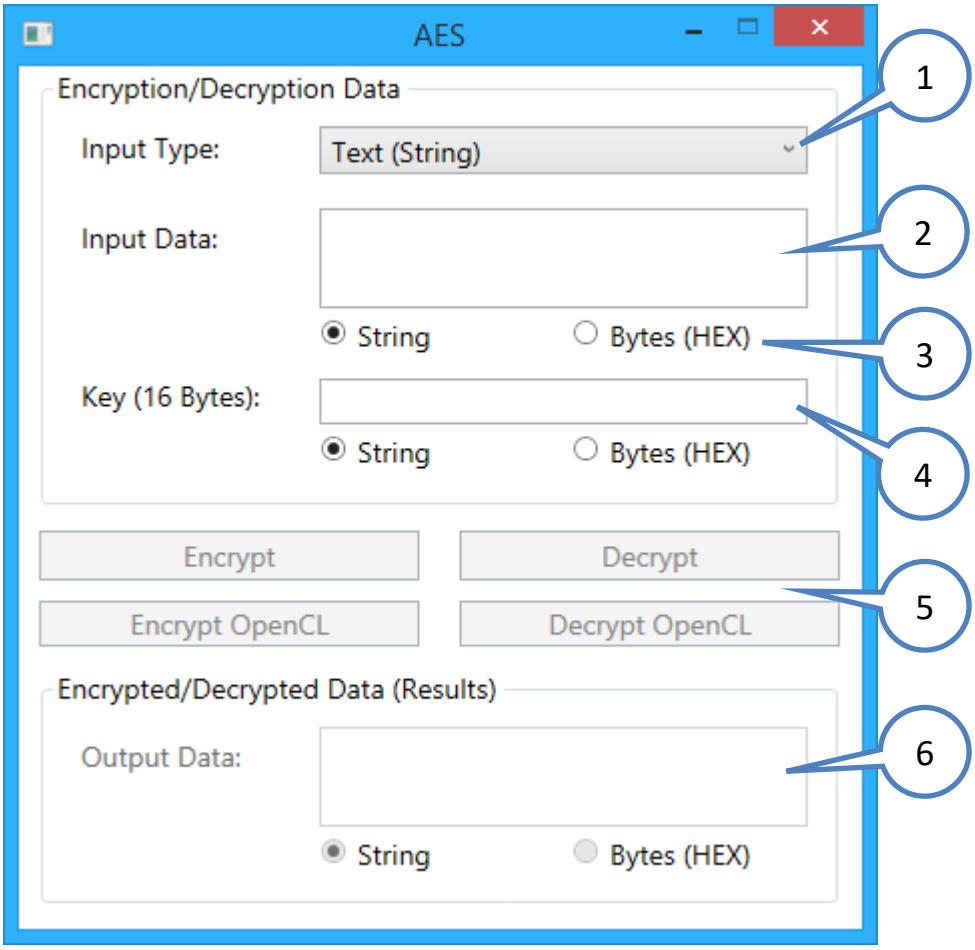


Рисунок 2.1 – Графічний інтерфейс для тестування системи.

Елементи графічної оболонки:

1. Вибір типу введення – випадаючий список, який дозволяє вибрати один із способів введення даних для обробки. Дозволяє вибрати ввід тексту або обробку файлів.

2. Поле вводу – дозволяє ввести дані для обробки. Якщо було вибрано режим текстового вводу, то в поле вводиться будь-який текст, якщо вибрано файловий ввід – вводиться шлях до одного або кількох файлів.

3. Тип текстового вводу – дозволяє вказати, як сприймати введений текст, дійсно як текст чи, сприймати його як набір байтів, записаних в шістнадцятковій системі числення.

4. Поле вводу ключа – призначене для введення ключа шифрування. Також є можливість сприймати ключ як текст, або як набір байтів.

5. Панель керування – дозволяє обрати режим роботи. В наслідок використання центрального процесору без технології OpenCL (Кнопки «Encrypt» і «Decrypt») є можливість шифрування і дешифрування, а також можливість шифрування, використовуючи пристрої з підтримкою OpenCL (кнопки «Encrypt OpenCL» і «Decrypt OpenCL»).

6. Поле результату – виводить результат після обробки. Якщо оброблявся текст, то в полі буде виведено результат обробки, який може бути представлений також як набір байтів. Якщо оброблялися файли, то поле буде містити шляхи до шифрованих/дешифрованих файлів.

2.4 Аналіз швидкості шифрування/дешифрування

У процесі тестування було проведено перевірку на швидкість дії обробки даних. Для цього було вибрано текстову інформацію і файли з різними розмірами, шифрування проводилося як на центральному процесорі без використання додаткових технологій, так і на графічному процесорі з використанням технології OpenCL. Нижче наведена таблиця з вхідними даними для обробки:

Таблиця 2.1 – Вхідні дані обробки

Тип даних	Розмір даних
Текст	15 Б
Текст	75 Б
Файл	3.25 МБ
Файл	73.24 МБ
Файл	405 МБ

Далі наведені результати тестування – графіки, на яких порівнюється час обробки однакових даних для CPU і GPU.

Вертикальна шкала містить час виконання, горизонтальна – розмір даних.

На кожному графіку зображено дві прямих – синю і зелену. Синя пряма зображає роботу CPU, зелена – GPU. На рисунку 2.2 зображено графік обробки даних розміром 15 байтів.

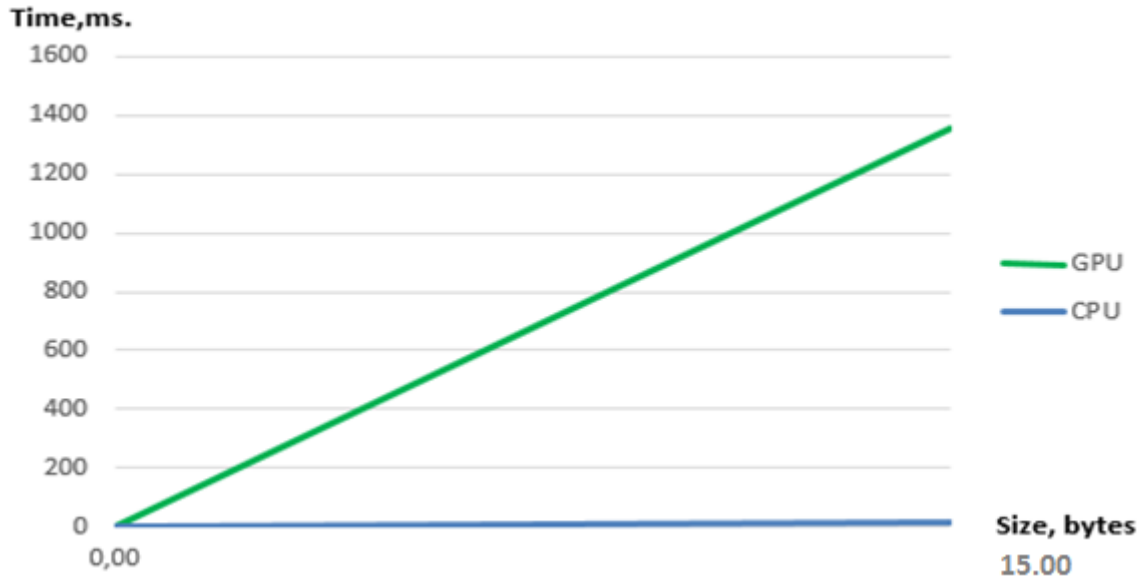


Рисунок 2.2 – Результати обробки даних розміром 15 байтів

На графіку видно, що шифрування даних, які мають малий розмір є неефективним з використанням GPU і OpenCL. На центральному процесорі воно виконується значно швидше. Причина полягає в тому, що при використанні графічного процесора, йому треба передати дані для обробки в його пам'ять, а це займає час, тому в такому випадку центральний процесор є більш продуктивним. Наступий графік наведено у рисунку 2.3 обробки даних розміром 75 байтів.

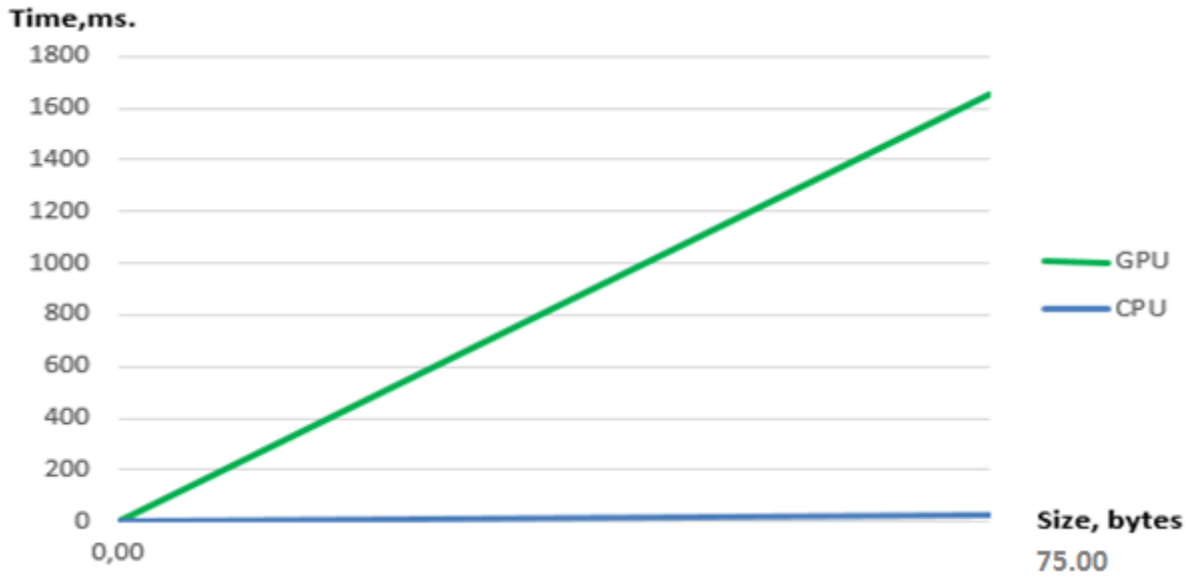


Рисунок 2.3 – Результат обробки даних розміром 75 байтів

З цього графіку також видно, шо центральний процесор швидше справляється з шифруванням, ніж графічний. Це також зумовлено малим обсягом вхідних даних і часом, необхідним на пересилання даних на графічний процесор. На рисунку 2.4 зображено графік обробки даних розміром 3.25 МБ.

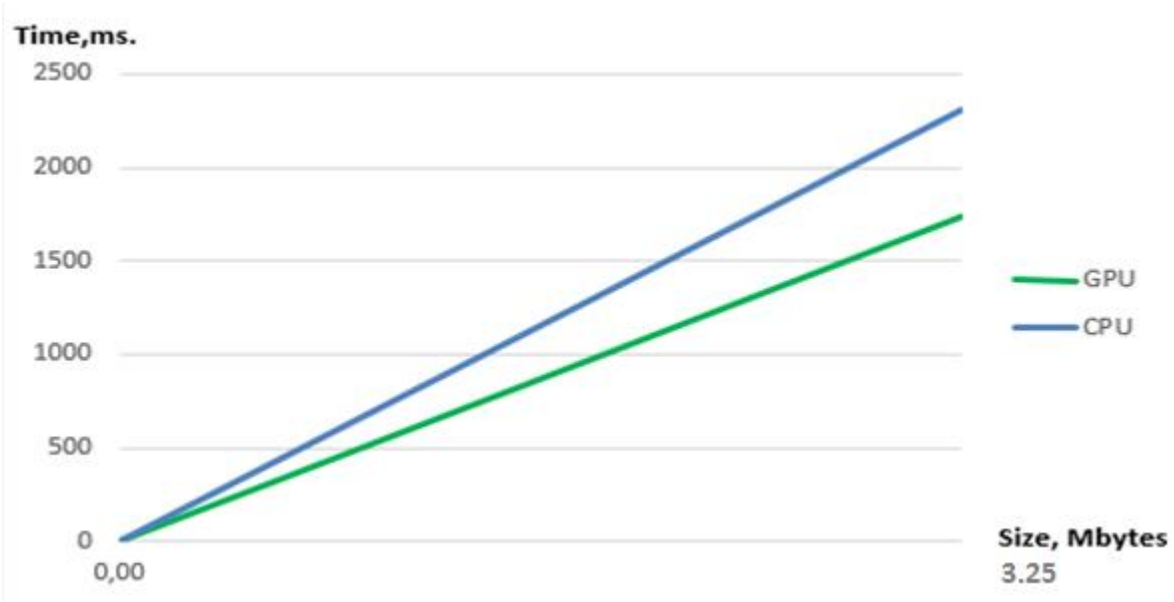


Рисунок 2.4 – Результат обробки даних розміром 3.25 МБ

На графіку видно, що ефективнішим є графічний процесор. Розмір вхідних даних значно збільшився. Тепер час пересилання даних на OpenCL – пристрій не є критичним для виграшу в продуктивності. Центральний процесор виконує обробку повільніше. На рисунку 2. 5 подано графік обробки даних розміром 73.24 МБ.

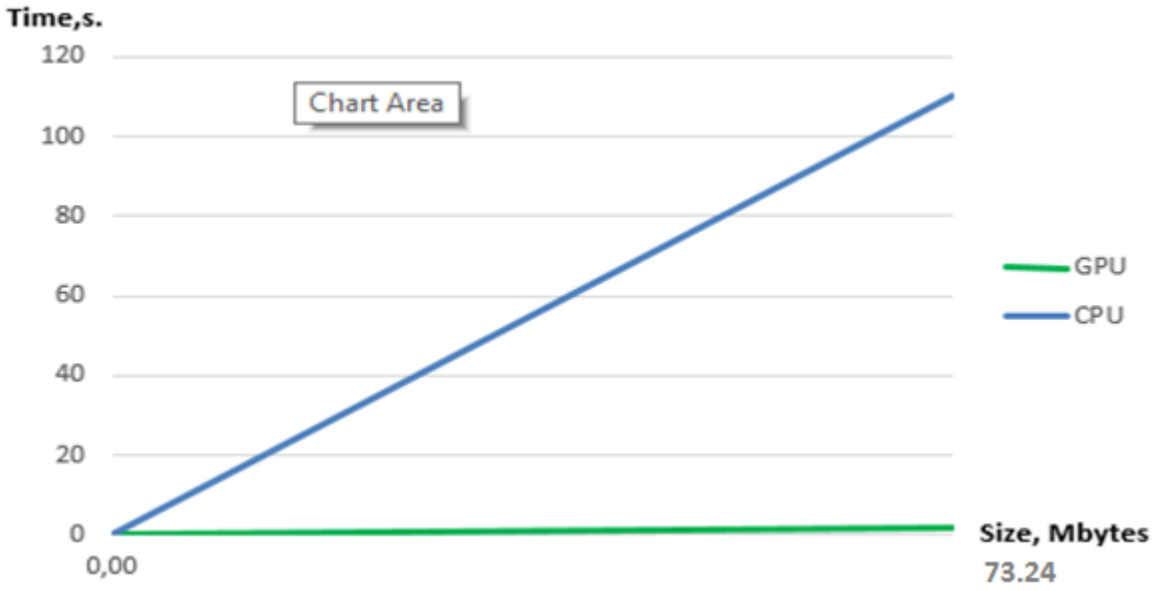


Рисунок 2.5 – Результат обробки даних розміром 73.24 МБ

На цьому графіку видно значний виграш в продуктивності при використання графічного процесора і технології OpenCL. Час обробки даних центральним процесором є значно більшим, ніж час обробки графічним процесором. Починають проявлятися переваги використання GPU і OpenCL. На рисунку 2.6 зображено графік обробки даних розміром 405 МБ.

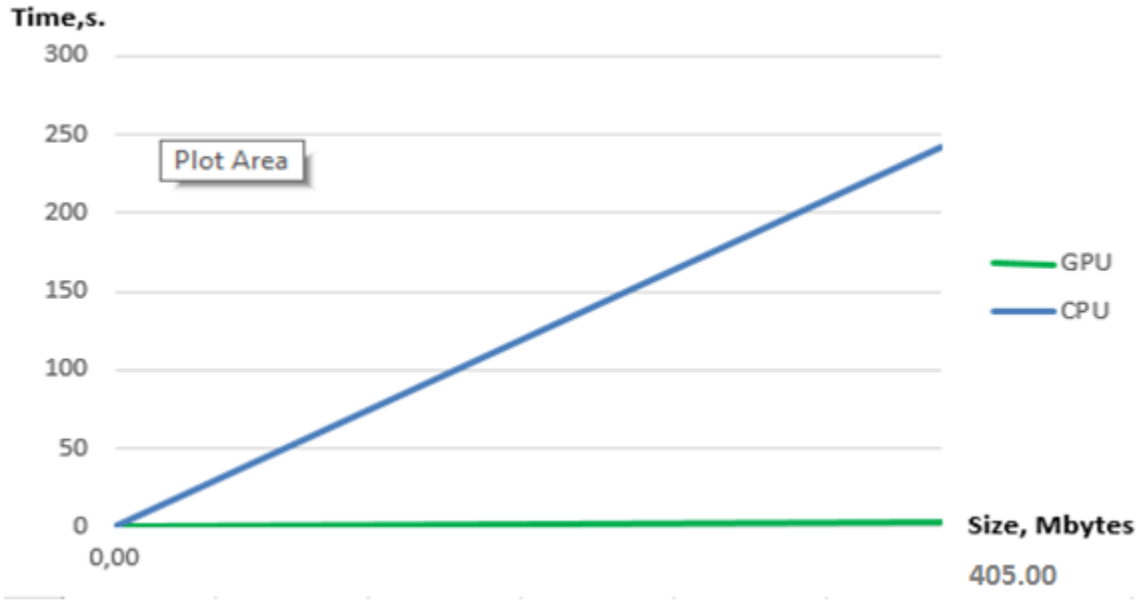


Рисунок 2.6 – Результат обробки даних розміром 405 МБ

Цей графік демонструє ще більший виграш в продуктивності з використанням GPU і OpenCL в порівнянні з центральним процесором. При збільшені обсягу вхідних даних з 73.64 МБ до 405 МБ час обробки з використанням GPU збільшився приблизно в 1.5 разу, а з використанням CPU – більше, як в 2 рази. Обробка за допомогою центрально процесу займає кілька хвилин, а графічний процесор здійснює обробку в межах 5 секунд.

2.5 Аналіз продуктивності системи

При тестуванні розробленої бібліотеки було проведено 5 тестів, які демонструють виграш продуктивності при використанні GPU і OpenCL в

порівнянні з використанням CPU. Нижче наведена таблиця з готовими результатами тестів.

Таблиця 2.2 – Готові результати тестів

Тип даних	Розмір даних	Час обробки за допомогою CPU	Час обробки за допомогою GPU і OpenCL
Текст	15 Б	15мс.	1с. 241мс.
Текст	75 Б	26мс.	1с. 581мс.
Файл	3.25 МБ	3с. 207мс.	1с. 842мс.
Файл	73.24 МБ	1хв. 40с. 226мс.	1с. 987мс.
Файл	405 МБ	4хв. 25с. 154мс.	3с. 754мс.

Із результатів тестування видно, що при збільшенні розміру вхідних даних, виграш GPU над CPU збільшується в рази. Дана бібліотека може застосовуватися в галузях, де потрібне швидке шифрування/дешифрування великих обсягів даних.

2.6 Переваги та недоліки бібліотеки

Після тестування бібліотеки, можна виділити її переваги та недоліки.

Основною перевагою даної бібліотеки є її продуктивність – вона швидко проводить обробку великих за обсягом даних. Бібліотека дозволяє вибрати платформу і пристрій для обробки даних або обирає такі платформу і пристрій, які будуть найбільш потужними для виконання обчислень.

Основним недоліком є те, що при малому обсязі вхідних даних з використанням OpenCL, вона не буде настільки ефективною, як центральний процесор. У цьому розділі було детально описано блок – схеми алгоритмів, що реалізовані у системі. Представлено та детально описано структуру класів системи. Проведено тестування швидкості роботи при різних розмірах вхідних даних. Перераховано переваги та недоліки розробленої бібліотеки.

3 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Охорона праці [52,53]

Розробка дипломного проекту «Розробка програмної підсистеми алгоритмічного перетворення даних з використанням різнорідних обчислювальних засобів мовою С», вимагає роботи за комп'ютером, відповідно необхідно знати та дотримуватись вимог техніки безпеки, охорони праці та протипожежної безпеки.

На сьогоднішній день, існують чітко визначені правила та норми експлуатації електронно-обчислювальних машин. Діючі норми описані в нормативно-правовому акті НПАОП.

Відповідно до нормативно-правового акту з охорони праці розміщення робочих місць з ЕОМ у підвалах і цокольних приміщеннях не допускається. Площа приміщення з ПЕОМ визначається з розрахунку на одне робоче місце - не менш 6 м², а обсяг - не менш 20,0 м³ з урахуванням кількості осіб, що одночасно працюють у зміну.

Віконні прорізи повинні бути орієнтовані на північний схід і мати жалюзі або штори.

Забороняється в приміщеннях з ПЕОМ застосовувати полімерні матеріали (дерев'яно-стружкові плити, шпалери, що миються, рулонні синтетичні матеріали), які виділяють у повітря шкідливі хімічні речовини. У приміщеннях необхідно проводити щоденне вологе прибирання.

Приміщення для роботи з ПЕОМ повинні бути обладнані системами опалення, кондиціювання повітря або припливно-витяжною вентиляцією.

Робочі місця в офісі працівників варто розташовувати так, щоб природне світло падало збоку, переважно ліворуч. При розміщенні робочих місць із ПЕОМ варто дотримуватися наступних вимог:

- робочі місця розташовуються на відстані не менше 1 метра від стін зі світловими прорізами;

- відстань між бічними поверхнями відеотерміналів повинна бути не менш 1,2 метри;

- відстань між тильною поверхнею одного відеотермінала й екраном іншого не повинен бути менш ніж 2,5 метри;

- відстань між рядами робочих місць повинен бути не менше 1 метра.

Висота робочої поверхні для відеотерміналу повинна бути в межах 680-800 мм. Рекомендовані розміри стола: висота 725 мм, ширина 600-1400 мм, глибина 800-1000 мм. Робочий стіл повинен мати простір для ніг висотою не менш 600 мм, шириною не менш 500 мм, глибиною на рівні колін не менше 450 мм, на рівні витягнутої ноги не менш 650 мм, а так само повинна бути підставка для ніг.

Робоче крісло користувача ЕОМ повинен мати сидіння, спинку й стаціонарні або знімні підлокітники, положення яких можна регулювати. Екран відеотермінала й клавіатуру варто розташовувати на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків і символів.

Відстань від екрана до очей працівника повинна становити: при розмірі екрана по діагоналі 35-38 см (14"(15")) - 600-700 мм; 43 см (17") - 700-800 мм; 48 см (19") - 800-900 мм; 53 см (21") - 900-1000 мм.

Розміщення принтера повинне забезпечувати гарну видимість екрана відеотермінала й ручного керування ним. Під матричний принтер варто підкладати вібраційний килимок. При виконанні робіт, що вимагають високої концентрації уваги, необхідно відокремити одне робоче місце від іншого перегородками висотою 1,5-2,0 метра.

Приміщення з ЕОМ повинні бути обладнані системою загального рівномірного освітлення. У виробничих і адміністративно-суспільних

приміщеннях, де переважно ведеться робота з документами, допускається комбінована система штучного освітлення.

Загальне освітлення повинно бути виконане у вигляді суцільних або переривчастих ліній світильників, які розміщаються збоку від робочих місць (переважно ліворуч) паралельно лінії зору працівників.

Для загального освітлення необхідно використовувати світильники із сітками, що розсіюють світло та дзеркальними екранними сітками, укомплектовані високочастотними пускорегулюючими апаратами (ВЧ ПРА).

Як джерела світла, варто використовувати люмінесцентні лампи типу ЛБ. У світильниках місцевого освітлення можна використовувати лампи накалювання або енергозберігаючі лампи. При відсутності світильників із ВЧ ПРА світильники загального освітлення необхідно підключати до різних фаз трифазної мережі.

Коефіцієнт запасу (Кз) для освітлювальної установки варто приймати рівним 1,4. Освітленість на робочих столах у зоні розміщення документів повинна бути в межах 300-500 лк, при цьому світильники місцевого освітлення варто розташовувати таким чином, щоб не було відблисків на поверхні екрана, а освітленість екрана не перевищувала 300 лк.

В офісах, конторських і інших суспільних приміщеннях можуть бути використані також світильники для ламп розжарювання, але з енергозберігаючими лампами.

Світильники з трубчатими люмінесцентними лампами переважно розміщаються безперервними рядами, бажано паралельно стіні з вікнами або найдовшій стороні вузького приміщення. Ряди з розривами допускаються, якщо це необхідно відповідно розрахунку. Світильники на чотири і більше трубчатих люмінесцентних ламп, а також світильники з енергозберігаючими лампами в адміністративно - побутових і громадських приміщеннях установлюються по вершинам квадратних або прямокутних полів.

Отже, при проектуванні інтернет-магазину комп’ютерного обладнаня на базі раціонального уніфікованого процесу було дотримано всіх вимог техніки безпеки, охорони праці та протипожежної безпеки.

3.2 Безпека в надзвичайних ситуаціях [54,55]

3.2.1 Інженерний захист персоналу об’єкту та населення. Правила застосування

Інженерний захист населення та територій - це комплекс організаційних і інженерно-технічних заходів, що проводяться завчасно, а також у оперативному порядку та спрямованих на запобігання або максимальне зниження втрат населення при виникненні надзвичайних ситуацій шляхом забезпечення укриття і життєдіяльності населення в захисних спорудах, запобігання, усунення або зниження до допустимого рівня негативного впливу уражаючих факторів, стихійних лих, аварій, природних і техногенних катастроф.

Інженерний захист планується і здійснюється на основі:

- оцінки характеристик можливу небезпеку;
- урахування категорій захищеного населення;
- результатів інженерно-геодезичних, геологічних, гідрометеорологічних вишукувань;
- схем інженерного захисту території (генеральних, детальних, спеціальних);
- урахування особливостей використання території.

В мирний і воєнний час використовуються наступні види захисних споруд (ЗС): спеціальні фортифікаційні споруди (СФС), польові фортифікаційні споруди (ВФС) та захисні споруди цивільної оборони (ЗС ЦО).

Захисні споруди цивільної оборони класифікуються за такими ознаками:

- часу зведення;
- місця розташування;
- місткості;
- захисним властивостям;
- матеріалу конструкцій;
- забезпечення електроенергією;
- забезпечення фільтровентиляційним обладнанням;
- характером використання в мирний час.

Сховища можуть бути вбудованими в інші будівлі і споруди, а також окремо стоять. Вони можуть бути заглибленими і підлозі заглибленими. Притулку можуть розміщуватися в районах міської забудови, на об'єктах економіки, в метрополітені, підземному просторі міст, у гірських виробках та природних підземних порожнинах.

По місткості сховища поділяються на малі (до 150 осіб), середні (150-500 чоловік) і великі (понад 500 чоловік).

Високу ефективність у справі захисту населення і територій має проведення інженерно-технічних заходів щодо захисту від несприятливих і небезпечних природних явищ і процесів, що передбачають будівництво та експлуатацію відповідних захисних інженерних споруд. До них належать заходи по захисту від землетрусів, протизсувні та противообвальні інженерні заходи, заходи захисту від селів, протилавинні інженерні заходи, протикарстові заходи, заходи інженерного захисту берегів морів, водойм і водотоків, інженерно-технічні заходи по захисту від затоплень та інші.

Інженерні заходи щодо захисту від землетрусів полягають у сейсмічному мікрорайонуванні та дотриманні норм проектування та будівництва будівель і споруд у сейсмічних районах.

Протизсувні та протиобвальні інженерні заходи включають:

- зміну рельєфу схилів в цілях планування укосів, зменшення крутизни схилів, підвищення їх стійкості, а також регулювання стоку поверхневих вод, штучне пониження рівня підземних вод, їх перехоплення за допомогою дренажних систем;
- будівництво утримуючих споруд (банкет, терас, підпірних і підтримують стін, голівок, анкерних кріплень, тунелів, критих огорож, пальових рядів) особливо в тих місцях, де схили підрізають дорогами;
- пристрій направляючих стінок для зміни руху обвальних порід;

Протикарстові інженерні заходи проводяться шляхом:

- заповнення карстових порожнин;
- водозниження і регулювання режиму підземних вод;
- організації поверхневого стоку.

До основних інженерно-технічних заходів по захисту від затоплень і підтоплень відносяться:

- штучне підвищення поверхні території;
- випрямити і поглиблення русел річок та їх розчищення;
- влаштування дамб;
- відведення поверхневих і підземних вод;
- будівництво дренажних систем.

Для вирішення всього комплексу завдань з інженерного захисту населення і територій від надзвичайних ситуацій природного і техногенного характеру перспективна розробка програм інженерного захисту населених пунктів України від впливу небезпечних природних і техногенних явищ та процесів. Реалізація таких програм дозволить суттєво вплинути на стан природної та техногенної безпеки в країні.

3.2.2 Характеристики аварій на виробництвах із застосуванням хлору. Перша допомога. Профілактика уражень

Усередині зони забруднення виникають осередки хімічного ураження (ОХУ) тобто територія, в межах якої відбувається масове забруднення і ураження людей, тваринного і рослинного світу.

У медико-тактичному відношенні ОХУ характеризуються:

- раптовістю ураження;
- масовістю поразок;
- наявністю комбінованих уражень;
- забрудненням навколишнього середовища.

В осередку ураження стійкими речовинами, тривалий час (більше 1 години), зберігається небезпека ураження. Вона зберігається деякий час і після виходу з вогнища за рахунок десорбції СДОР з одягу або в результаті контакту із зараженим транспортом, різним майном.

Перша медична допомога (ПМД) ураженим виявляється безпосередньо на місці ураження. Це досягається двома шляхами:

- уражені надають само-та взаємодопомога;
- негайним залученням медичних формувань.

Само-і взаємодопомога :

- промити очі водою;
- надіти протигаз або ВМД, змочену 2% содою;
- обробити уражені ділянки шкіри мильним розчином;
- негайно покинути вогнище (краще на транспорті).

Перша медична та перша лікарська допомога в місцях збору уражених:

- зняти з ураженої протигаз і звільнити від стискує одягу;
- зігріти ураженого;
- евакуювати до лікувального закладу;

- проводити інгаляції кисню;
- при зупинці дихання ШВЛ, в / в цитітон;
- при спазмі голосової щілини: тепло на область шиї, атропін, при необхідності трахеотомія.

У результаті проведеного дослідження теоретичної частини даного питання можна зробити наступні висновки, що для зменшення аварійних ситуацій на підприємствах, які використовують у технологічному процесі хлор, необхідно:

- суворе спостереження за точністю протікання технологічних процесів, умовами зберігання та транспортування хлору;
- суворий нагляд за станом технологічних конструкцій і надійністю обладнання;
- підвищення ефективності засобів протиаварійного захисту;
- поліпшення стану контрольно-вимірювальних приладів та засобів автоматизації;
- зниження запасів хлору до мінімального, необхідного за технологією, кількості;
- своєчасність і якість планово-попереджувальних ремонтних робіт;
- здійснення організаційних, технічних та інших заходів, спрямованих на обмеження поширення хлору за межі санітарно-захисної зони при аваріях і руйнування;
- підвищення професіоналізму і практичних навичок персоналу;
- створення локальних систем оповіщення;
- організація штатних служб з попередження і ліквідації аварій;
- підготовка персоналу підприємства до дій при виникненні нештатних ситуацій на виробництві;
- навчання населення, що проживає поблизу підприємств виробництва, зберігання і транспортування хлору, дій у надзвичайній ситуації, пов'язаної з викидом хлору.

ВИСНОВКИ

Метою роботи була розробка програмної підсистеми прямого та зворотного алгоритмічного перетворення даних з використанням різнорідних обчислювальних засобів мовою C#.

Об’єктом дослідження та розробки була програмна підсистема призначенням якої є алгоритмічне пряме та зворотне перетворення даних, які не призначені для широкого загалу та містять інформацію конфіденційного толку, чи можуть стосуватись комерційної таємниці підприємства чи організації, а вже безпосередньо предмет дослідження міститься в межах об’єкту, зокрема програмної підсистеми.

В результаті виконання роботи вирішено наступні, пріоритетні, задачі:

- програмний продукт забезпечує повноцінне алгоритмічне перетворення даних (криптографічне), яке здійснюється в посимвольній послідовності, використовуючи блочний симетричний алгоритм при цьому;
- швидкість алгоритмічного перетворення даних не залежить від збільшення об’єму оброблюваних даних;
- для алгоритмічного перетворення даних можна вибрати будь-який файл або ввести будь-який текст;
- програма працює через розпаралелення процесу алгоритмічного перетворення даних що пришвидшує отримання результату;
- програма орієнтована на платформу Microsoft Windows.

Для реалізації заплановано використати технології фреймворку OpenCL метою якого є доповнення Open GL і OpenA L, які можна назвати відкритими галузевими стандартами прямим призначенням яких є тривимірна комп'ютерна графіка і звук, при використанні можливостей GPU. Відкрита мова обчислень OpenCL вигідно виділяється також бо програмно можна в динаміці визначати доступність процесорів, з багатоядерними центральними процесорами та графічними процесорами включно. Саме завдяки цьому є можливість, проте і

залежність від можливостей апаратного забезпечення самих клієнтів, достатньо інтенсивно покращувати продуктивність своїх програм.

Розроблена програмна підсистема забезпечує такі функції:

1-пряме алгоритмічне перетворення тексту;

2-зворотнє алгоритмічне перетворення тексту;

3-пряме алгоритмічне перетворення послідовності байт;

4-зворотнє алгоритмічне перетворення послідовності байт;

5-виводу результату як тексту;

6-виводу результату як послідовність байтів;

7-пряме алгоритмічне перетворення будь - якого вибраного файлу;

8-зворотнє алгоритмічне перетворення будь - якого вибраного файлу;

9-можливість одночасного вибору кількох файлів для обробки;

10-зберігання оброблених файлів;

11-вибір способу прямого алгоритмічного перетворення;

12-можливість вводу ключа текстом і послідовністю байтів.

Під час розробки бібліотеки поставлене завдання реалізувати симетричний алгоритм перетворення даних, враховуючи критерії надійність, швидкодія, і можливість розпаралелення. Для реалізації було обрано алгоритм AES, який забезпечує надійне алгоритмічне перетворення даних, а також може бути розпаралелений. Алгоритм працює з блоками розміром по 16 байт, кожен з яких обробляється незалежно, тому можна паралельно обробляти, і тим самим досягається більша продуктивність.

Обрав найоптимальніший, на мою думку, алгоритм на базі шифра Rijndael – Advanced EncryptionStand, який широко використовується і є одним із найпопулярніших та одним із найбезпечніших симетричних алгоритмів шифрування, який добре піддається розпаралелюванню.

Також було завдання використати гетерогенні технології. Для реалізації було обрано технологію OpenCL, оскільки вона є відкритим стандартом, а також є кросплатформовою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.

3 Нікольський Ю.В. Дискретна математика / Ю.В.Нікольський, В.В.Пасічник, Ю.М.Щербина – Львів: Магнолія Плюс, 2007. – 608 с.

4. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник ІВ Бойко, МР Петрик, Г Цуприк - 2020

5. Дискретні структури (Алгебраїчні та числові системи, комбінаторний аналіз) : навчально-методичний посібник для студентів спеціальності 121 «Інженерія програмного забезпечення», аспірантів та викладачів вищих навчальних закладів / Укладач : Бойко І.В., Петрик М.Р., Цуприк Г.Б. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2017 – 64 с.

6. Моделювання та видобуток даних (висопродуктивні обчислення у великих алгебраїчних та числових системах, комбінаторному аналізі): навчальний посібник. Тернопіль: : ТНТУ 2019 – 62 с.

7. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

8. ВВ Нога, ГБ Цуприк Інформаційні технології при створенні автоматизованих систем для задач збору та збереження інформації - Збірник тез доповідей VII Міжнародної науково-технічної конференції молодих учених та студентів. Актуальні задачі сучасних технологій – Тернопіль 28-29 листопада 2018

9. Aaftab Munshi; Benedict R. Gaster; Timothy G. Mattson; James Fung; Dan Ginsburg. OpenCL Programming Guide. — Addison-Wesley Professional, 2011. — 648 p. — ISBN 978-0-321-74964-2.

10. Embedded Computer Vision (Advances in Computer Vision and Pattern Recognition) / Branislav Kisacanin, Shuvra S. Bhattacharyya, Sek Chai. — Springer, 2010. — P. 17—18. — 284 p. — ISBN 978-1849967761.

11. Hyesoon Kim, Richard Vuduc, Sara Baghsorkhi. Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU). — Morgan & Claypool Publishers, 2012. — 96 p. — ISBN 978-1-60845-954-4.

12. Laplante, Phil (2009). Requirements Engineering for Software and Systems (вид. 1st). Redmond, WA: CRC Press. ISBN 1-42006-467-3.

13. McConnell, Steve (1996). Rapid Development: Taming Wild Software Schedules (вид. 1st). Redmond, WA: Microsoft Press. ISBN 1-55615-900-5.

14. Wiegers, Karl E. (2003). Software Requirements (вид. 2nd). Redmond, WA: Microsoft Press. ISBN 0-7356-1879-8.

15. Andrew Stellman and Jennifer Greene (2005). Applied Software Project Management. Cambridge, MA: O'Reilly Media. ISBN 0-596-00948-8.

16. Brian Berenbach, Daniel Paulish, Juergen Katzmeier, Arnold Rudorfer (2009). Software & Systems Requirements Engineering: In Practice. New York: McGraw-Hill Professional. ISBN 0-07-1605479.

17. Walter Sobkiw (2008). Sustainable Development Possible with Creative System Engineering. New Jersey: CassBeth. ISBN 0615216307.

18. Windows Exploitation in 2014 [Електронний ресурс]. Режим доступу: URL: / http: // www.welivesecurity.com/wp-content/uploads/2015/01/Windows-Exploitation- in-2014.pdf, вільний (10.02.2017).

19. [Електронний ресурс] – Режим доступу: URL: https://www.quora.com/topic/Plug-and-Play

20. [Електронний ресурс] – Режим доступу: URL: https://hostpro.ua/wiki/ua/security/encryption-types-algorithms

21. Mouhamed Abdulla. On the Fundamentals of Stochastic Spatial Modeling and Analysis of Wireless Networks and its Impact to Channel Losses. Ph.D.

Dissertation, Dept. of Electrical and Computer Engineering, Concordia Univ., Montréal, Québec, Canada, Sep. 2012.: (footnote of Page 126).

22. СУЧАСНІ МЕТОДИ ШИФРУВАННЯ ІНФОРМАЦІЇ // Лук'янихін О. В., «Перший крок у науку», 7 грудня 2014 р., Суми, Україна 26 Секція: «Оптика. Електроніка. Інформаційні технології»

23. ГОСТ Р 50922-2006. Захист інформації. Основні терміни та визначення.

24. ГОСТ Р 53114-2008. Захист інформації. Забезпечення інформаційної безпеки в організації. Основні терміни та визначення.

25. ГОСТ 27.002-89. Надійність в техніці. Основні поняття. Терміни та визначення.

26. Алгоритм шифрування DES [Електронний ресурс] – Режим доступу: URL: http://uk.wikipedia.org/wiki/Data_Encryption_Standard. Загол. з екрану.

27. Алгоритм шифрування AES [Електронний ресурс] – Режим доступу: URL: http://uk.wikipedia.org/wiki/Advanced_Encryption_Standard Загол. з екрану.

28. Алгоритм шифрування Serpent [Електронний ресурс] – Режим доступу: URL: http://uk.wikipedia.org/wiki/OpenCL Загол. з екрану.

29. Технологія GPGPU [Електронний ресурс] – Режим доступу: URL: http://uk.wikipedia.org/wiki/GPGPU Загол. з екрану.

30. Технологія Microsoft Direct Compute [Електронний ресурс] – Режим доступу: URL: http://en.wikipedia.org/wiki/DirectCompute Загол. з екрану.

31. Технологія RC6 [Електронний ресурс] – Режим доступу: URL: https://ru.wikipedia.org/wiki/RC6 Загол. з екрану.

32. Жуков І.А. Корочкін О.В. Паралельні та розподілені обчислення. Навч. посібн. 2-ге видання, К.: Корнійчук, 2014. – 284 с.

33. Lutzky G., Korochkin O. Parallel Computing, Korneychuk, Kyiv, 2007, - 240 pp.

34. McCormick J.Singhoff F., Hugues J. Building Parallel, Embedded, and Real-Time Applications with Ada , Cambridge University Press, 2011, - 366 p.

35. CUDA [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/CUDA

36. Khronos [Електронний ресурс] – Режим доступу: URL: https://www.khronos.org/opencl/

37. AES [Електронний ресурс] – Режим доступу: URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

38. [Електронний ресурс] – Режим доступу: URL: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

39. AddRoundKey [Електронний ресурс] – Режим доступу: URL: https://commons.wikimedia.org/wiki/File:AESAddRoundKey.svg?uselang=ua: Загол. з екрану.

40. Standards Ukraine 2014, DSTU 7624:2014 Information Technologies. Cryptographic Data Security. Symmetric block transformation algorithm. [online] Available: http://ukrndnc.org.ua/downloads/new_view/?i=dstu-7624-2014&pz=%C4%D1%D2%D3+7624%3A2014.

41. E. Käsper, P. Schwabe, "Faster and Timing-Attack Resistant AES-GCM", in Proc. 11th International Workshop Cryptographic Hardware and Embedded Systems, Lausanne, Switzerland, 2009, pp. 1-17.

42. R. Oliynykov, O. Kazymyrov, O. Kachko, R. Mordvinov, et al. Source code for performance estimation of 64-bit optimized implementation of the block ciphers Kalyna, AES, GOST, BelT, Kuznyechik. [online] Available: https://github.com/Roman-Oliynykov /ciphers-speed/.

43. A. Reyhani-Masoleh, M. Taha, and D. Ashmawy, "Smashing the implementation records of AES S-Box", IACR Transactions on Cryptographic Hardware and Embedded Systems, no. 2, pp. 298–336, 2018.

44. C# [Електронний ресурс] – Режим доступу: URL: https://uk.wikibooks.org/wiki/C%2B%2B

45. Айвор Хортон Microsoft Visual C++ 2005: базовий курс = Beginning Visual C++ 2005. — «Діалектика», 2007. — С. 1152. — ISBN 0-7645-7197-4

46. [Електронний ресурс] – Режим доступу: URL: https://learn.microsoft.com/dotnet/desktop/wpf/overview/?view=netdesktop-7.0

47. WPF [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/Windows_Presentation_Foundation

48. Open CL [Електронний ресурс] – Режим доступу: URL: https://www.macworld.com/article/191827/snowleopard_opencl.html

49. CUDA [Електронний ресурс] – Режим доступу: URL: https://web.archive.org/web/20200122163721/https://developer.nvidia.com/cuda-zone

50. Kaner, Cem; Falk, Jack; Nguyen, Hung Quoc (1999). Testing Computer Software, 2nd Ed. New York, et al: John Wiley and Sons, Inc. с. 480.

51. Тестування ПЗ [Електронний ресурс] – Режим доступу: URL: https://qalight.ua/baza-znan/

52. Дистанційний курс «Основи охорони праці» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: http://dl.tntu.edu.ua/index.php

53. Про затвердження Правил охорони праці під час експлуатації ЕОМ [Електронний ресурс]. – Режим доступу: URL: http://zakon4.rada.gov.ua/laws/show/z0382-99

54. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / В.С. Стручок –Тернопіль: ФОП Паляниця В. А., –156 с. Отримано з https://elartu.tntu.edu.ua/handle/lib/39196.

55. Інформацію при написанні зазначеного підрозділу можна отримати з навчального посібника: Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / автор-укладач В.С. Стручок– Тернопіль: ФОП Паляниця В. А., – 156 с. Отримано з http://elartu.tntu.edu.ua/handle/lib/39424

56. О.В.Михайлівський, Г.Б Цуприк Розробка програмної підсистеми алгоритмічного перетворення даних. Матеріали XI науково-технічної конфції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 13-14 грудня 2023 р.). – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2023. – 257 с.

ДОДАТКИ

Додаток А

Лістинг програми

```
#include "paes_constants_and_datatypes.h"


__constant const uchar sbox_encrypt[AES_SBOX_SIZE] =
AES_SBOX_ENCRYPT;
__constant const uchar sbox_decrypt[AES_SBOX_SIZE] =
AES_SBOX_DECRYPT;
__constant const uchar logtable[AES_SBOX_SIZE] = AES_LOGTABLE;
__constant const uchar alogtable[AES_SBOX_SIZE] = AES_ALOGTABLE;

void sub_bytes(size_t block, __global uchar * buffer, __constant
const uchar * sbox)
{
     for (size_t i = 0; i < 16; ++i)
         buffer[block * 16 + i] = sbox[buffer[block * 16 + i]];
}

void shift_rows(size_t block, __global uchar * buffer)
{
     uchar temp;

     // Rotate first row 1 columns to left
     temp = buffer[block * 16 + 4];
     buffer[block * 16 + 4] = buffer[block * 16 + 5];
     buffer[block * 16 + 5] = buffer[block * 16 + 6];
     buffer[block * 16 + 6] = buffer[block * 16 + 7];
     buffer[block * 16 + 7] = temp;

     // Rotate second row 2 columns to left
     temp = buffer[block * 16 + 8];
     buffer[block * 16 + 8] = buffer[block * 16 + 10];
     buffer[block * 16 + 10] = temp;

     temp = buffer[block * 16 + 9];
     buffer[block * 16 + 9] = buffer[block * 16 + 11];
     buffer[block * 16 + 11] = temp;

     // Rotate third row 3 columns to left
     temp = buffer[block * 16 + 12];
     buffer[block * 16 + 12] = buffer[block * 16 + 15];
     buffer[block * 16 + 15] = buffer[block * 16 + 14];
     buffer[block * 16 + 14] = buffer[block * 16 + 13];
     buffer[block * 16 + 13] = temp;
}

void inv_shift_rows(size_t block, __global uchar * buffer)
{
     uchar temp;
```

```
    // Rotate first row 1 columns to right
    temp = buffer[block * 16 + 7];
    buffer[block * 16 + 7] = buffer[block * 16 + 6];
    buffer[block * 16 + 6] = buffer[block * 16 + 5];
    buffer[block * 16 + 5] = buffer[block * 16 + 4];
    buffer[block * 16 + 4] = temp;

    // Rotate second row 2 columns to right
    temp = buffer[block * 16 + 8];
    buffer[block * 16 + 8] = buffer[block * 16 + 10];
    buffer[block * 16 + 10] = temp;

    temp = buffer[block * 16 + 9];
    buffer[block * 16 + 9] = buffer[block * 16 + 11];
    buffer[block * 16 + 11] = temp;

    // Rotate third row 3 columns to right
    temp = buffer[block * 16 + 12];
    buffer[block * 16 + 12] = buffer[block * 16 + 13];
    buffer[block * 16 + 13] = buffer[block * 16 + 14];
    buffer[block * 16 + 14] = buffer[block * 16 + 15];
    buffer[block * 16 + 15] = temp;
}

#define GF2M(k, b) ((b) == 0 ? 0 : (alogtable[(logtable[(k)] +
logtable[(b)]) % 255]))

void mix_columns(size_t block, __global uchar * m)
{
    uchar new_block[16];
     new_block[0] = GF2M(2, m[block * 16 + 0]) ^ GF2M(3, m[block *
16 + 4]) ^ m[block * 16 + 8] ^ m[block * 16 + 12];
     new_block[1] = GF2M(2, m[block * 16 + 1]) ^ GF2M(3, m[block *
16 + 5]) ^ m[block * 16 + 9] ^ m[block * 16 + 13];
    new_block[2] = GF2M(2, m[block * 16 + 2]) ^ GF2M(3, m[block * 16
+ 6]) ^ m[block * 16 + 10] ^ m[block * 16 + 14];
     new_block[3] = GF2M(2, m[block * 16 + 3]) ^ GF2M(3, m[block *
16 + 7]) ^ m[block * 16 + 11] ^ m[block * 16 + 15];
     new_block[4] = GF2M(2, m[block * 16 + 4]) ^ GF2M(3, m[block *
16 + 8]) ^ m[block * 16 + 12] ^ m[block * 16 + 0];
     new_block[5] = GF2M(2, m[block * 16 + 5]) ^ GF2M(3, m[block *
16 + 9]) ^ m[block * 16 + 13] ^ m[block * 16 + 1];
     new_block[6] = GF2M(2, m[block * 16 + 6]) ^ GF2M(3, m[block *
16 + 10]) ^ m[block * 16 + 14] ^ m[block * 16 + 2];
     new_block[7] = GF2M(2, m[block * 16 + 7]) ^ GF2M(3, m[block *
16 + 11]) ^ m[block * 16 + 15] ^ m[block * 16 + 3];
     new_block[8] = GF2M(2, m[block * 16 + 8]) ^ GF2M(3, m[block *
16 + 12]) ^ m[block * 16 + 0] ^ m[block * 16 + 4];
     new_block[9] = GF2M(2, m[block * 16 + 9]) ^ GF2M(3, m[block *
16 + 13]) ^ m[block * 16 + 1] ^ m[block * 16 + 5];
     new_block[10] =    GF2M(2, m[block * 16 + 10]) ^ GF2M(3,
m[block * 16 + 14]) ^ m[block * 16 + 2] ^ m[block * 16 + 6];
```

```
     new_block[11] =     GF2M(2, m[block * 16 + 11]) ^ GF2M(3,
m[block * 16 + 15]) ^ m[block * 16 + 3] ^ m[block * 16 + 7];
     new_block[12] =     GF2M(2, m[block * 16 + 12]) ^ GF2M(3,
m[block * 16 + 0]) ^ m[block * 16 + 4] ^ m[block * 16 + 8];
     new_block[13] =     GF2M(2, m[block * 16 + 13]) ^ GF2M(3,
m[block * 16 + 1]) ^ m[block * 16 + 5] ^ m[block * 16 + 9];
     new_block[14] =     GF2M(2, m[block * 16 + 14]) ^ GF2M(3,
m[block * 16 + 2]) ^ m[block * 16 + 6] ^ m[block * 16 + 10];
     new_block[15] =     GF2M(2, m[block * 16 + 15]) ^ GF2M(3,
m[block * 16 + 3]) ^ m[block * 16 + 7] ^ m[block * 16 + 11];
     for (size_t i = 0; i < 16; ++i)
         m[block * 16 + i] = new_block[i];
}

void inv_mix_columns(size_t block, __global uchar * m)
{
    uchar new_block[16];
     new_block[0] = GF2M(0xe, m[block * 16 + 0]) ^ GF2M(0xb, m[block
* 16 + 4]) ^ GF2M(0xd, m[block * 16 + 8]) ^ GF2M(0x9, m[block * 16 +
12]);
     new_block[1] = GF2M(0xe, m[block * 16 + 1]) ^ GF2M(0xb, m[block
* 16 + 5]) ^ GF2M(0xd, m[block * 16 + 9]) ^ GF2M(0x9, m[block * 16 +
13]);
     new_block[2] = GF2M(0xe, m[block * 16 + 2]) ^ GF2M(0xb, m[block
* 16 + 6]) ^ GF2M(0xd, m[block * 16 + 10]) ^ GF2M(0x9, m[block * 16
+ 14]);
     new_block[3] = GF2M(0xe, m[block * 16 + 3]) ^ GF2M(0xb, m[block
* 16 + 7]) ^ GF2M(0xd, m[block * 16 + 11]) ^ GF2M(0x9, m[block * 16
+ 15]);
     new_block[4] = GF2M(0xe, m[block * 16 + 4]) ^ GF2M(0xb, m[block
* 16 + 8]) ^ GF2M(0xd, m[block * 16 + 12]) ^ GF2M(0x9, m[block * 16
+ 0]);
     new_block[5] = GF2M(0xe, m[block * 16 + 5]) ^ GF2M(0xb, m[block
* 16 + 9]) ^ GF2M(0xd, m[block * 16 + 13]) ^ GF2M(0x9, m[block * 16
+ 1]);
     new_block[6] = GF2M(0xe, m[block * 16 + 6]) ^ GF2M(0xb, m[block
* 16 + 10]) ^ GF2M(0xd, m[block * 16 + 14]) ^ GF2M(0x9, m[block * 16
+ 2]);
     new_block[7] = GF2M(0xe, m[block * 16 + 7]) ^ GF2M(0xb, m[block
* 16 + 11]) ^ GF2M(0xd, m[block * 16 + 15]) ^ GF2M(0x9, m[block * 16
+ 3]);
     new_block[8] = GF2M(0xe, m[block * 16 + 8]) ^ GF2M(0xb, m[block
* 16 + 12]) ^ GF2M(0xd, m[block * 16 + 0]) ^ GF2M(0x9, m[block * 16
+ 4]);
     new_block[9] = GF2M(0xe, m[block * 16 + 9]) ^ GF2M(0xb, m[block
* 16 + 13]) ^ GF2M(0xd, m[block * 16 + 1]) ^ GF2M(0x9, m[block * 16
+ 5]);
     new_block[10] = GF2M(0xe, m[block * 16 + 10]) ^ GF2M(0xb,
m[block * 16 + 14]) ^ GF2M(0xd, m[block * 16 + 2]) ^ GF2M(0x9,
m[block * 16 + 6]);
     new_block[11] = GF2M(0xe, m[block * 16 + 11]) ^ GF2M(0xb,
m[block * 16 + 15]) ^ GF2M(0xd, m[block * 16 + 3]) ^ GF2M(0x9,
m[block * 16 + 7]);
```

```
    new_block[12] = GF2M(0xe, m[block * 16 + 12]) ^ GF2M(0xb,
m[block * 16 + 0]) ^ GF2M(0xd, m[block * 16 + 4]) ^ GF2M(0x9,
m[block * 16 + 8]);
    new_block[13] = GF2M(0xe, m[block * 16 + 13]) ^ GF2M(0xb,
m[block * 16 + 1]) ^ GF2M(0xd, m[block * 16 + 5]) ^ GF2M(0x9,
m[block * 16 + 9]);
    new_block[14] = GF2M(0xe, m[block * 16 + 14]) ^ GF2M(0xb,
m[block * 16 + 2]) ^ GF2M(0xd, m[block * 16 + 6]) ^ GF2M(0x9,
m[block * 16 + 10]);
    new_block[15] = GF2M(0xe, m[block * 16 + 15]) ^ GF2M(0xb,
m[block * 16 + 3]) ^ GF2M(0xd, m[block * 16 + 7]) ^ GF2M(0x9,
m[block * 16 + 11]);
    for (size_t i = 0; i < 16; ++i)
        m[block * 16 + i] = new_block[i];
}

void add_round_key(size_t block, __global uchar * buffer, __constant
const uchar * round_key, size_t round_key_index)
{
    for (size_t i = 0; i < 16; ++i)
        buffer[block * 16 + i] ^= round_key[round_key_index * 16 +
i];
}

/**
 * OpenCL kernel that does a single AES round.
 * \param buffer the input/output buffer
 * \param blocks the number of blocks contained in the buffer
 * \param mode one between AES_MODE_ENCRYPT and AES_MODE_DECRYPT
 * \param round_key the AES round keys
 * \param rounds the number of rounds
 * \param round the AES round to do
 */
__kernel __attribute__ ((vec_type_hint(uchar)))
void kernel_aes(__global uchar * buffer, const ulong blocks, const
uint mode, __constant const uchar * round_key, const uint rounds,
const uint round)
{
    size_t global_work_size = get_global_size(0);
    size_t global_id = get_global_id(0);
    /* If there are more work items than blocks, each work items
takes AT MOST
       1 block to process, otherwise it could be several blocks per
work item. */
    size_t blocks_per_work_item = global_work_size < blocks ?
blocks / global_work_size : 1;
    size_t reminder = global_work_size < blocks ? blocks %
global_work_size : 0;
    size_t from_block = global_id * blocks_per_work_item;
    if (global_id < reminder)
        from_block += global_id;
    else
        from_block += reminder;
```

```
     /* If the work item should start working from a block outside
the input data
        boundaries, it shall do nothing. */
     if (from_block < blocks) {
          /* The last work item will process the input data from its
start block to
             the very last block. The other work items will just do
blocks_per_work_item
             blocks.
             Each work item will process any block b such as
from_block <= b < to_block. */
          //~ size_t to_block = global_id == global_work_size - 1 ?
blocks : from_block + blocks_per_work_item;
          size_t to_block = from_block + blocks_per_work_item;
          if (global_id < reminder)
               to_block += 1;
          switch (mode) {
          case AES_MODE_ENCRYPT:
               {
                    for (size_t b = from_block; b < to_block; ++b) {
#ifdef SHIFT_ROWS
                         shift_rows(b, buffer);
#elif defined(MIX_COLUMNS)
                         mix_columns(b, buffer);
#elif defined(ADD_ROUND_KEY)
                         add_round_key(b, buffer, round_key, rounds);
#elif defined(SUB_BYTES)
                         sub_bytes(b, buffer, sbox_encrypt);
#else
                         if (round == 0) {
                              add_round_key(b, buffer, round_key, 0);
                         } else if (round == rounds) {
                              sub_bytes(b, buffer, sbox_encrypt);
                              shift_rows(b, buffer);
                              add_round_key(b, buffer, round_key,
rounds);
                         } else {
                              sub_bytes(b, buffer, sbox_encrypt);
                              shift_rows(b, buffer);
                              mix_columns(b, buffer);
                              add_round_key(b, buffer, round_key,
round);
                         }
#endif
                    }
                    break;
               }
          case AES_MODE_DECRYPT:
               {
                    for (size_t b = from_block; b < to_block; ++b) {
#ifdef SHIFT_ROWS
                         inv_shift_rows(b, buffer);
```

```
#elif defined(MIX_COLUMNS)
                        inv_mix_columns(b, buffer);
#elif defined(ADD_ROUND_KEY)
                        add_round_key(b, buffer, round_key, rounds);
#elif defined(SUB_BYTES)
                        sub_bytes(b, buffer, sbox_decrypt);
#else
                        if (round == 0) {
                             add_round_key(b, buffer, round_key,
rounds);
                        } else if (round == rounds) {
                             inv_shift_rows(b, buffer);
                             sub_bytes(b, buffer, sbox_decrypt);
                             add_round_key(b, buffer, round_key, 0);
                        } else {
                             inv_shift_rows(b, buffer);
                             sub_bytes(b, buffer, sbox_decrypt);
                             add_round_key(b, buffer, round_key,
rounds - round);
                             inv_mix_columns(b, buffer);
                        }
#endif
                    }
                    break;
               }
          }
     }
}
```

Додаток Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

О.В.Михайлівський, Г.Б Цуприк, Н.Є. Андрійчук
РОЗРОБКА ПРОГРАМНОЇ ПІДСИСТЕМИ АЛГОРИТМІЧНОГО ПЕРЕТВОРЕННЯ ДАНИХ
O.V. Mykhailivskyi, H.B. Tsupryk, N. Ye. Andriychuk
DEVELOPMENT OF THE SOFTWARE SUBSYSTEM FOR ALGORITHMIC DATA CONVERSION 216

С.В. Осельський, Г. Б. Цуприк
РОЗРОБКА РЕАКТИВНОГО ФРОНТЕНД ФРЕЙМВОРКУ ДЛЯ ОДНОСТОРІНКОВИХ ДОДАТКІВ З ВЛАСНОЮ СИСТЕМОЮ РЕАКТИВНОСТІ
S.V. Oselskyi, H.B. Tsupryk
DEVELOPMENT OF A REACTIVE FRONTEND FRAMEWORK FOR SINGLE-PAGE APPLICATIONS WITH ITS OWN REACTIVITY SYSTEM 217

Д.Р. Пасіка, Г. Б. Цуприк
РОЗРОБКА ОДНОСТОРІНКОВОГО ВЕБ-ЗАСТОСУНКУ З ВИКОРИСТАННЯМ VUE.JS І REACT: ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОДУКТИВНОСТІ КОРИСТУВАЦЬКОГО ДОСВІДУ ТА ДОЦІЛЬНОСТІ
D.R. Pasika, H. B. Tsupryk
DEVELOPING A SINGLE-PAGE WEB APPLICATION USING VUE.JS AND REACT: A COMPARATIVE ANALYSIS OF USER EXPERIENCE PERFORMANCE AND EXPEDIENCY 218

І. Ярош, Д. Полюніна
РОЗРОБЛЕНА СИСТЕМА ПІДБОРУ ТА ОРГАНІЗАЦІЇ EVENT-ПОДІЙ
I. Yarosh, D. Polunina
THE SYSTEM FOR DELIVERY AND ORGANIZATIONS EVENT-SUPPORT IS UPDATED 220

В. Рудак, Ю. Стоянов
ПРОЕКТУВАННЯ ТА РОЗРОБКА БАЗИ ДАНИХ ДЛЯ ПОЛІКЛІНІКИ З ВИКОРИСТАННЯМ МЕТОДОЛОГІЇ RUP ТА МОВИ ПРОГРАМУВАННЯ C#
V. Rudak, Y. Stoyanov
DESIGN AND DEVELOPMENT OF DATABASE FOR POLYCLINIC USING RUP METHODOLOGY AND C# PROGRAMMING LANGUAGE 222

Стебніцький А., Биковий П.Є.
РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ДЛЯ ЕЛЕКТРОННОГО СУДДІ У ФУТБОЛІ ДЛЯ АВТОМАТИЗОВАНОГО ВИЗНАЧЕННЯ ОФСАЙДУ
Stebnitskiy A., Bykovy P.
Development of a software module for an electronic referee in football for automated offside determination 223

Татарчук І., Биковий П. Є.
ОПТИМІЗОВАНИЙ МЕТОД ПАРАЛЕЛЬНОГО ШВИДКОГО СОРТУВАННЯ ДЛЯ ОБРОБКИ ВЕЛИКИХ ОБСЯГІВ ДАНИХ
Tytarchuk I., Bykovy P.
OPTIMIZED METHOD OF PARALLEL FAST SORTING FOR PROCESSING LARGE VOLUMES OF DATA 224

Ілля Федорович, Галина Осуховська
ВИКОРИСТАННЯ SPARK STREAMING ТА SPARK STRUCTURED STREAMING ДЛЯ ОБРОБКИ ДАНИХ В РЕАЛЬНОМУ ЧАСІ
Illia Fedorovych, Halyna Osukhivska
USING SPARK STREAMING AND SPARK STRUCTURED STREAMING FOR REAL TIME DATA PROCESSING 225

В.В. Хапор, Д.М. Михалик
ПЕРЕВАГИ ВИКОРИСТАННЯ АРХІТЕКТУРИ МОДУЛЬНОГО МОНОЛІТА
V.V. Khatsior, D.M. Mykhalyk
BENEFITS OF USING MODULAR MONOLITH ARCHITECTURE 226

УДК 001
М34

ПРОГРАМНИЙ КОМІТЕТ

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Матеріали XI науково-технічної конфер «Інформаційні моделі, системи та
М34 технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 13-14 грудня 2023 р.). – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2023. – 271 с.

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001,
тел. (0352) 52-41-33, факс (0352) 254983.
E-mail: **confit2023@gmail.com**

Редагування, оформлення, верстка: Семенишин Г.М.

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання;
- Інформаційні системи та технології;
- Комп'ютерні системи та мережі;
- Програмна інженерія та моделювання складних розподілених систем;
- Новітні фізико-технічні та освітні технології.

В збірнику надруковано тези доповідей XI науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 13-14 грудня 2023 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

УДК 004.41

О.В.Михайлівський, Г.Б Цуприк, канд. техн. наук, доц.
Тернопільський національний технічний університет імені Івана Пулюя, Україна
Н.Є. Андрійчук
Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету ім.І.Пулюя» , Україна

РОЗРОБКА ПРОГРАМНОЇ ПІДСИСТЕМИ АЛГОРИТМІЧНОГО ПЕРЕТВОРЕННЯ ДАНИХ

O.V. Mykhailivskyi, H.B. Tsupryk, PhD, Assoc.Prof., N. Ye. Andriychuk

DEVELOPMENT OF THE SOFTWARE SUBSYSTEM FOR ALGORITHMIC DATA CONVERSION

Актуальність теми та доцільність проведеної роботи аргументовано тим, що ніби і банальні слова «інформація» та «сучасність», проте, вони твердо увійшли у буремне сьогодення, що все що нас оточує несе якусь, хоча б мінімальну, інформацію зі змістом і не випадково знаходиться в тому чи іншому місці. Однак, не до всього будь-хто та будь-де може мати доступ, і цьому є різні причини. Одну із них, і напевно чи не найголовнішу, на мою думку, можна охарактеризувати словом «не наша/ой». А вже розмір тієї «нашої», безпосередньо може залежати від того, де отримані відомості будуть застосовані і яке призначення чи напрям (плановане чи якесь своє) буде обрано користувачем.

Основне призначення розроблюваною програмної підсистем, є пряме та зворотне алгоритмічне перетворення даних, які не призначені для широкого загалу, тобто містять інформацію конфіденційного толку, чи можуть стосуватись комерційної таємниці підприємства чи організації. Передбачається, що програмний продукт може застосовуватися в будь–якій галузі, де виникає подібна потреба із-за ненадійності сховищ, а також для приховування інформації з метою її передачі по незахищеним каналам зв'язку.

Об'єктом дослідження та розробки є програмна підсистема призначеним якої є алгоритмічне перетворення даних, які не призначені для широкого загалу та містять інформацію конфіденційного толку, чи можуть стосуватись комерційної таємниці підприємства чи організації. Вже безпосередньо предмет дослідження міститься в межах об'єкту, зокрема програмної підсистеми.

Для реалізації запланована використати технології фреймворку OpenCL метою якого є доповнення Open GL і OpenA L, які можна назвати відкритими галузевими стандартами прямим призначенням яких є тривимірна комп'ютерна графіка і звук, при використанні можливостей GPU. Важливим моментом є те, що OpenCL розробляєся та підтримується некомерційною організацією Khronos Group – консорціум, до якого входять такі відомі IT компанії як Apple,AMD, Intel,SunMicrosystems, Sony CE та деякі інші. Відкрита мова обчислень OpenCL вигідно виділяється також бо «дозволяє» програмам динамічно визначати доступність процесорів, з багатоядерними центральними процесорами та графічними процесорами включно. Саме завдяки цьому розробники можуть, проте залежать від можливостей апаратного забезпечення самих клієнтів, достатньо інтенсивно покращувати продуктивність своїх програм.

Література

1. Моделювання та видобуток даних (високопродуктивні обчислення у великих алгебраїчних та числових системах, комбінаторному аналізі): навчальний посібник. Тернопіль: ТНТУ 2019 – 62 с.
2. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – "Інженерія програмного забезпечення" для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.