Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Педнко Данила Ігоровича*
(ПІБ)

академічної групи *121-18-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему:

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*Доц. Бердник М.Г*			
розділів:				
спеціальний	*Доц. Бердник М.Г*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента 121-18-2 *Педенко Данило Ігорович*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення для математичних обчислень на GPU мовою програмування C++*

затверджена наказом ректора НТУ «ДП» від №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *Доц. Бердник М.Г*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Педекно Д.І*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: _53__ с., _18__ рис., _0__ табл., _3_ дод., _14__ джерел.

Мета кваліфікаційної роботи: за допомогою технології CUDA реалізувати алгоритм арифметичних операцій для виконання обчислень на відеокарті.

У вступі розглядається важливість та ефективність підтримки обчислень на відеокарті та вивчення алгоритмів оптимізації, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розробляється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці додатку, який підвищує коефіцієнт корисної дії.

Актуальність програмного продукту визначається ростом попиту на наукову сферу, яка акцентує увагу на математичних розрахунків у програмних додатків, тому найважливішим у такій сфері є збільшення пропускної спроможності, на що було направлена ця розробка

Список ключових слів: CUDA, GPU, CPU, МАТЕМАТИЧНА ОПЕРАЦІЯ, ДОДАТОК.

ABSTRACT

Explanatory note: _53__ pp., _18__ fig., _0__ tab., _3_ appendix, _14__ sources.

The purpose of the qualification work: with the help of CUDA technology to implement the algorithm of arithmetic operations to perform calculations on the video card.

The introduction considers the importance and effectiveness of video card computing support and the study of optimization algorithms, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and the purpose of development is determined, the statement of the task is developed.

The second section selects the platform for development, performs program design and development, describes the algorithm and structure of the system, determines the input and output data, provides characteristics of the parameters of technical means, describes the program.

The economic section determines the complexity of the developed software product, calculates the cost of work to create an application and calculates the time to create it.

Of practical importance is the development of an application that increases efficiency.

The relevance of the software product is determined by the growing demand for the scientific field, which focuses on mathematical calculations in software applications, so the most important in this area is to increase the bandwidth, which was aimed at this development

List of keywords: CUDA, GPU, CPU, MATHEMATICAL OPERATION, APPENDIX.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GPGPU – техніка використання графічного процесора відеокарт

GPU – графічний процесор

CPU – центральний процесор

CUDA – програмно-апаратна архітектура

ЗМІСТ

ВСТУП

Графічний процесор є одним з найважливіших типів обчислень, доступних сьогодні як на персональних, так і на комерційних комп'ютерах. Розширені графічні процесори використовують переваги широкого спектру розширених функцій, таких як графіка та візуалізація. Графічні процесори, які хочуть відчути весь потенціал гри, стають все більш популярними серед креативних індустрій та штучного інтелекту.

Ця розробка безпосередньо пов'язана з математикою і матиме велике значення в галузі науки. У науці дуже важлива швидкість, з якою можна виконати велику кількість завдань за короткий проміжок часу. Нині найбільшою проблемою є неефективність сучасної обчислювальної техніки.

Для вирішення цієї проблеми необхідно обчислення виконувати на GPU, але на даний момент акцент робиться лише на CPU. В цьому і полягає моя задача у цій дипломній роботі.

Мета цієї роботи це за допомогою технології CUDA реалізувати алгоритм арифметичних операцій для виконання обчислень на відеокарті(GPU)

Дипломна робота складається з таких частин:

- вступ, який обгрунтовує актуальність роботи; визначає цілі проведення наукового дослідження; галузь дослідження; методи дослідження або розрахунків;
- постановка задачі;
- огляд предметної галузі;
- порівняння відомих методів реалізації;
- розробка програмного забезпечення для візуалізації;
- приклади роботи програмного додатку та аналіз результатів реалізації моделі;
- висновки;

— список використаної літератури;

— додатки.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

Графічний процесор, як і GPU, сьогодні є одним з найважливіших видів обчислювальної техніки, як сособистих, так і ділових обчислень. Розширений для паралельної обробки графічний процесор використовує переваги широкого спектру розширених функцій, включаючи графіку та візуалізацію. Бажаючи відчути свій ігровий потенціал в цілому, графічні процесори стають все дужче популярними поміж творчого виробництва та у штучному інтелекті.

Графічні процесори були оновлені для швидкого відтворення 3D-графіки. З роками вони стали більш гнучким і програмованим, розширюючи його можливості. Це дозволило розробникам графікам створювати більш реалістичні та візуальні ефекти, використовуючи передові методи освітлення та затінення. Інші розробники також почали використовувати здатність графічних процесорів значно прискорювати додаткові робочі процеси у високопродуктивних обчисленнях.

1.2. Призначення розробки та галузь застосування

На сьогоднішній день програмна розробка суцільно на відеокартах зростає. Це пов'язано з ростом кількості задач які потрібно виконати за обмежений проміжок часу. Основним критерієм якості програмного забезпечення є швидкість виконання великого об'єму інформації та задач які були поставлені. При вивченні технологій використання цпу у порівнні гпу було виявлено значну різницю у пропускній спроможності в 1 проміжок часу. Елементарний зразок маємо легковий автомобіль зі швидкістю 90

км/год та місткістю 4 особи, і автобус зі швидкістю 60 км/год та місткістю 20 осіб. Якщо за операцію порахувати перевіз 1 людини на 1 кілометр, то затримка легкового автомобіля – 3600/90=40с – за стільки секунд 1 людина подолала дистанцію 1 кілометр, пропускна спроможність автомобіля – 4/40=0. 1 операцій/секунду; затримка автобуса - 3600/60 = 60с, пропускна спроможність автобуса - 20/60 = 0. 3 (3) операцій / секунду.

Так от, CPU - це автомобіль, GPU - автобус: він має велику затримку, але також і велику пропускну здатність. Якщо для вашої задачі затримка кожної конкретної операції не така важлива як чисельність цих операцій на секунду — не завадить розглянути застосування GPU. З цього виходить необхідність використовувати гпу для підвищення ККД. Особливо велике значення воно має при вирішенні математичних операцій задля більшої ефективності через велике навантаження на центральний процесор.

Велике значення ця розробка буде мати у науковому напрямку, адже напряму пов'язана з математичними операціями. У науковій сфері велике значення має швидкість виконання великої кількості задач за короткий проміжок часу. Великою проблемою на даний момент є низька ефективність сучасного приладдя яке використовується для обчислень.

Для вирішення цих проблем на даний момент на ринку мало достатньо ефективного способу виконання таких завдань, адже в основному ці обчислення виконується на CPU.

Метою розробки є підняття коефіцієнта корисної дії у виконанні математичних операцій за допомогою GPU на короткий проміжок часу

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та

графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 317 від 07.06.2021 р;
- завдання на кваліфікаційну роботу на тему «Розробка програмного забезпечення для математичних обчислень на GPU мовою програмування C++»

1.4. Постановка завдання

Завдання цієї роботи буде полягати в розробки програми яка буде виконувати математичні обчислення з використанням технологій Cuda.

Результатом цієї розробки повинна бути програма яка може відворити математичні операції без використання центрального процесору.

Для розробки свої програми я використовую мову програмування C++ з підтримкою NvidiaCuda, так як для використання GPU замість CPU необхідні специфічні технології.

Cuda дозволяє організувати доступ до набору інструкцій графічного прискорювача і керувати його пам'яттю разом з організацією паралельних обчислень.

Так як нам необхідно отримувати доступ до графічного процесору логічним вибором стало використанням технології CUDA.

1.5 Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розробити програму для математичних обчислень на графічному процесорі яка буде працювати на C++ підтримувати Windows 10 та Mac і працювати без інтернет підключення. Збільшення пропускної спроможності.

1.5.2. Вимоги до інформаційної безпеки

Для надійної роботи додатку необхідно:

- конфіденційність інформації;
- цілісність даних;
- використовувати ліцензійне програмне забезпечення на сервері;
- здійснювати захист від вірусів на сервері;
- здійснювати захист від несанкціонованого доступу.

1.5.3. Вимоги до складу та параметрів технічних засобів

Необхідні мінімальні технічні вимоги для роботи з даним програмним продуктом мають такі характеристики:

- Операційна система: Windows 7. 8. 10;
- Процессор 1.3 ГГц;
- Оперативная память: 2GB;
- Роздільна здатність монітору: 1280x800;
- Наявність засобів вводу (клавіатура, мишка);

  Вільне місце на жорсткому диску: 10 Мб.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програмний код для реалізації даної кваліфікаційної роботи має бути створений за допомогою мови програмування C++ у середовищі розробки програмного забезпечення Microsoft Visual Studio. Програма має бути скомпільована та запускатися за допомогою виконуваного файлу в усіх версіях родини операційних систем Windows.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Функціональне призначення програми

Для досягнення поставленої задачі зроблено програмне забезпечення та підтримує виконання таких операцій: обчислення різних математичних виразів та функцій, обчислення масивів.

2.2 Опис застосованих математичних методів

В першу чергу додаємо вектора паралельно(рис**. 2.1)**.

```
cudaError_t cudaStatus = addWithCuda(c, a, b, d, arraySize);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "addWithCuda failed!");
    return 1;
}
```

Рис.2.1 — Параллель вектора.

Після цього cudaDeviceReset повинен бути викликаний перед виходом для профілювання та інструментами трасування, такі як Nsight та VisualProfiler для відображення повного трасування(рис. 2.2).

```
cudaStatus = cudaDeviceReset();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaDeviceReset failed!");
    return 1;
}
cudaError_t cudaStatusMultiply = multiplyWithCuda(c, a, b, d, arraySize);
if (cudaStatusMultiply != cudaSuccess) {
    fprintf(stderr, "multiplyWithCuda failed!");
    return 1;
}
```

Рис. 2.2 — Відображення повного трасування.

Вибираємо, якому графічному процесорі працювати, змінюємо це у системі з кількома графічними процесорами(рис**.** 2.**3**)**.**

```
cudaStatus = cudaSetDevice(0);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU installed?");
    goto Error;
```

Рис. 2.3 — Вибір графічного процесору.

Виділяємо буфер GPU для чотирьох векторів (два входи, один вихід).(рис. 2.4).

```
// Выделить буферы GPU для 4 векторов (два входа, один выход).
cudaStatus = cudaMalloc((void**)&dev_c, size * sizeof(int));
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}
cudaStatus = cudaMalloc((void**)&dev_d, size * sizeof(int));
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}
cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}
```

Рис. 2.4 — Вибір буфера.

Копіюємо вхідні вектори з пам'яті хоста у буфери графічного процесора(рис. 2.5).

```
cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(int), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}

cudaStatus = cudaMemcpy(dev_b, b, size * sizeof(int), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}
cudaStatus = cudaMemcpy(dev_d, d, size * sizeof(int), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}
```

Рис. 2.5 — Копіювання вектора**.**

Запускаємо ядро GPU з одним потоком на кожен елемент **(рис. 2.6).**

```
addKernel2<<<1, size>>>(dev_c, dev_a, dev_b, dev_d);
```

Рис. 2.6 — Запуск ядра.

Перевіряємо наявність помилок під час запуску ядра(**рис. 2.7**).

```
cudaStatus = cudaGetLastError();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
    goto Error;
}
```

Рис. 2.7 — Перевірка помилок.

2.2.2 Математичні основи. Алгоритми розв'язання алгебраїчних систем рівнянь на відеокартах

Алгоритми розв'язування систем рівнянь можуть ефективно паралізувати обчислення. Через специфіку задачі найбільш логічним є використання векторних методів вирішення. Суть цих прийомів розпаралелювання полягає у виконанні однієї і тієї ж операції над різними елементами даних. Цей метод прискорення обчислень притаманний графічному процесору (GPU), як відомо, нижче наведені найважливіші алгоритми для рішень, які використовують точковий добуток(x,y) $x^T y$,а через $r_j \equiv b - AX_{j\,n}$ означає невідповідність j-тому кроку. Операцію вигляду α+αB , де a і b.- вектори, а a- скаляр

Обрати початкове наближення x_0:

$$\tilde{\mathbf{r}}_0 \,(\mathbf{r}_0, \tilde{\mathbf{r}}_0) \neq 0 \qquad r_0 = b - Ax_0$$

(2.8)

$$\mathbf{p}_0 = \mathbf{r}_0, \quad \tilde{\mathbf{p}}_0 = \tilde{\mathbf{r}}_0$$

(2.9)

Виберіть вектор, який задовольняє умові

$$\begin{aligned} &j = 0,1,\ldots \\ &\alpha_j = (\mathbf{r}_j, \tilde{\mathbf{r}}_j)/(\mathbf{A}\mathbf{p}_j, \tilde{\mathbf{p}}_j) \\ &\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha \mathbf{p}_j \\ &\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha \mathbf{A}\mathbf{p}_j \\ &\tilde{\mathbf{r}}_{j+1} = \tilde{\mathbf{r}}_j - \alpha \mathbf{A}^T \tilde{\mathbf{p}}_j \\ &\beta_j = (\mathbf{r}_{j+1}, \tilde{\mathbf{r}}_{j+1})/(\mathbf{r}_j, \tilde{\mathbf{r}}_j) \qquad (\mathbf{r}_0, \tilde{\mathbf{r}}_0) \end{aligned}$$

(2.10)

Обчислити для

$$\|\mathbf{r}_{j+1}\|_2 < \varepsilon$$

(2.11)

$\tilde{\mathbf{r}}_0$ Якщо , то КІНЕЦ (рішення)

$\beta_j = 0,$ Якщо , то КІНЕЦ(при метод не збігається)

$$\mathbf{p}_{j+1} = \mathbf{r}_j + \beta_j \mathbf{p}_j$$
$$\tilde{\mathbf{p}}_{j+1} = \tilde{\mathbf{r}}_j + \beta_j \tilde{\mathbf{p}}_j$$

(2.12)

Збільшити j

Розглянутий метод базується переважно на операціях обчислення точкового добутку та множення форми вектора матриці Ар. Повні обчислювальні вимоги однієї ітерації алгоритму спряженого градієнта можна описати таким чином:

$((\mathbf{r}_j, \tilde{\mathbf{r}}_j)$ - Ap_j , один скалярний добуток вже відомо), один поділ;

- один скаляр;

$\mathbf{A}^T \tilde{\mathbf{p}}_j$ - один скаляр (Ap_j вже відомо);

- , один скаляр;

$((\mathbf{r}_j, \tilde{\mathbf{r}}_j)$ - один скалярний добуток, один поділ вже відомо);

- один скалярний добутки, два порівняння;

- один скаляр;

- один скаляр.

Загальна потреба в ітераційних обчисленнях: обчислення Ар і АТpj, три крапкових добутків, п'ять (скалярних) тріад, два поділки і два порівняння.

Для будь-якої серйозної проблеми, пов’язаної з обчислювальними витратами, розрахунок Ар і АТpj буде домінувати в методі двовимірного

градієнта, тому ефективна реалізація матриці векторного множення має ключове значення.

$$x_i = \sum_{j=1}^{n} a_{ij} p_j$$

(2.13)

Множення повної матриці A розміром n × n на p-векторі складається з n скалярних добутків матричних виразів рядків матриці на векторі

Розрахунок одного елементу результуючого вектора не залежить від розрахунків інших елементів, тому така операція є легко розпаралелюємою. Тому, виходячи зі структури сучасного графічного процесора, який складається з набору мультипроцесорів, кожна скалярна робота Aip, де Ai строка матриці A, може виконуватися на окремому мультипроцесорі. Для цього матриця розбивається на блоки, один блок містить строку Ai і підраховує один елемент результуючого вектора.

$$s_k + s_{k+m/2},\ 0 \le k < m/2,\ s_k - k$$

(2.14)

Але ця паралелізація ще не обмежена, так як кожен мультипроцесор складається з певної кількості процесорів, що поділяють спільну високошвидкісну пам'ять, що дозволяє розпаралелити розрахунок скалярного добутку. У кожному блоці створюємо по m = 2^z потоків, кількість потоків обмежена максимально можливим числом потоків в блоці. Потік j читає з глобальної пам'яті j -ті, j + m -ті компоненти вектора p і лінії матриці A_i, обчислює відповідні добутки і записує їх у спільній пам'яті. Потоки запускають паралельно. Потім перша половина потоків одночасно виконує $s_k + s_{k+m/2},\ 0 \le k < m/2,\ s_k - k$ додавання

- комірки спільної пам'яті. Аналогічні дії потім виконуються потоками m/4 і т.д. до тих пір, поки в S_0 не буде розрахованbq скалярный добуток.

Розрахунок одного скалярного добутку на векторному пристрої не дає прискорення в порівнянні з послідовним розрахунком на центральному процесорі, але при розрахунку декількох скалярних добутків ефективність збільшується пропорційно кількості цих добутків. З цього випливає, що описаний спосіб добутку матриці на вектор на графічному процесорі дає більше прискорення залежно від збільшення розміру матриці.

2.3 Опис використаної архітектури

Життєвий цикл програми є безперервним процесом. Він починається, коли приймається рішення про необхідність створення програми і закінчується в момент його повного вилучення з експлуатації.

Існує кілька способів визначення фаз та робіт життєвого циклу.програмного забезпечення (ЖЦПЗ), кроків програмного процесу, каскадна і спіральна моделі. Вони утворюють вихор, але всі вони мають одні основні компоненти: постановка завдання, проектування рішення, впровадження та супровід рішень.

Цей проект використовує процес програмування по Райлі.

Процес програмування складається з чотирьох кроків:

- постановка задачі, тобто отримання уявлення про те, що має робити програма;
- проектування рішення для заздалегідь визначеної задачі (зазвичай таке рішення є менш формальним, ніж остаточна програма);
- програмне кодування, тобто переклад спроектованого рішення у програму, яка може працювати на пристрої;
- Підтримка програми, тобто постійне видалення з програми проблеми та додавання нових функцій.

Програмування починається з того моменту, коли користувач, тобто той, хто потребує програми для вирішення завдання, викладає проблему

системного аналітику. Користувачі та системні аналітики спільно визначають постановку завдання. Потім ця відповідальність переходить до алгоритмісту, відповідальної за проектування рішення. Рішення (або алгоритм) є послідовністю операцій та її виконання ведуть до вирішення проблеми, оскільки часто алгоритм не пристосований для роботи на комп'ютері і потребує розробки з машинною програмою. Це робиться за допомогою кодера. Для наступного відповідальний за внесення змін до програми несе супроводжуючий програміст. Коли системні аналізатори, алгоритми, кодери пов'язані – усі вони є програмістами.

Кількість користувачів, аналітиків і алгоритмістів для великого програмного проекту можуть виявитися важливими. Також може виникнути змушений привід повернутися до попередніх кроків через непередбачувані обставини. Все це служить додатковим доказом обережності розробки програмного забезпечення: кожен крок повинен мати повний, точний і зрозумілий результат.

Одним з найважливіших кроків у програмуванні є постановка завдання. Він служить контрактом між користувачем і розробником

(програмістами). Добре описане завдання, як користувачем, так і розробником означає, що завдання, яке потрібно виконати, є чітким і стислим, тобто при цьому враховуються інтереси як користувача, так і розробника. Користувач може планувати використання програмного забезпечення, яке ще не розроблено на основі того, що воно може. Гарна постановка завдання служить основою для формування її рішення.

Постановка завдання (специфікація програми) означає дійсно точний, повний і чіткий опис того, що відбувається під час виконання конкретної програми. Користувач зазвичай думає про комп'ютер як про чорний ящик: йому не важливо, як працює комп'ютер, важливо, що комп'ютер може робити того, що цікавить користувача. Акцент робиться на взаємодії людини з машиною.

Д. Райлі пропонує використовувати стандарт для розв’язання задачі форма, яка забезпечує максимальну точність, повноту, ясність і включає:

- найменування завдання (схематичне визначення);
- загальний опис (короткий виклад завдання);
- введення;
- висновок;
- помилки (явно перераховані незвичайні варіанти введення, щоб показати користувачам і програмістам ті дії, які зробить машина в подібних ситуаціях);
- приклад (хороший приклад може передати сутність завдання, а також проілюструвати різні випадки).

Розробка програмного забезпечення – найважча частина. На цій стадії постанову задачі необхідно перетворити в алгоритм. Отже, алгоритміст повинен: мати достатній досвід програмування та підходити до кожного нового завдання на основі добре відпрацьованої методології проектування. Щоб уникати помилки в програмах, алгоритми повинні бути ретельно розроблені процедури проектування на основі правил логічного висновку.

Завданням проектувальника є створення алгоритму, який виконує функції зв'язку між постановкою завдання і готовою до запуску програмою. Перевірка створеного алгоритму, тобто наскільки він відображає постановку завдання, здійснюється системним аналітиком. Тому і системний аналітик, і дизайнер повинні вміти читати і розуміти алгоритм. Всі алгоритми записуються на деякій псевдонімові. Алгоритми, також відомі як псевдокод, не можна запускати на жодному комп’ютері.

Перш ніж завершити роботу, кодіровщик повинен переконатися, що програма відповідає псевдокод. Потім системний аналітик, алгоритміст і, що найголовніше, користувач повинні протестувати і підтвердити, що вона працює правильно. Після цього можна вважати, що програма готова для передачі користувачеві в комплекті з всією необхідною документацією.

Проте на цьому програмування не закінчується і є наступним кроком у подальшому технічному обслуговуванні. Діло в тим, що в програмі можуть виникати помилки через недостатність директив команди або через те, що проект не відповідає робочому порядку або програма не відповідає проекту. Щоб користувач не очікував, поки програма працюватиме. Тому вони мають право вимагати зміни програми. Усунення несправностей є одним з основних завдань обслуговування програмного забезпечення. Іншою важливою функцією підтримки програми є модифікування, тобто додати нову функцію до програми або замінити існуючу функцію. Користувачі можуть змінювати умови програми, що змусить переписати складність підтримки програми яка залежить від типу змін, необхідно внести: у гіршому випадку. Програму можна оновлювати крок за кроком повністю. Резервне копіювання програми займає більше часу, ніж створення програми.

Останнім елементом процесу програмування є документація. Він містить багато пояснень, які полегшують процес програмування та роблять отриману програму більш повною. Постійна документація має бути невід'ємною частиною будь-якого процесу програмування. Опис проблеми, проектна документація, алгоритми та програми. Внутрішня документація, що надається безпосередньо в програмі, полегшує читання коду.

Відповідно до моделі, представленої в цьому проекті, програмування можна розділити на чотири фази: визначення проблеми, розробка рішення, кодування програми, обслуговування програми. Крім того, модель включає документування програми як дій, які будуть виконуватися під час процесу програмування.

Архітектура CUDA використовує модель пам'яті GRID, моделювання потоку кластера та інструкції SIMD. Він використовує не тільки

високопродуктивні графічні обчислення, а й різноманітні наукові обчислення з використанням відеокарт Nvidia. Вчені та дослідники широко використовують CUDA в різних областях, включаючи астрофізику, біологію та обчислювальну хімію, гідродинамічне моделювання, електромагнітні взаємодії, комп'ютерну томографію, сейсмічний аналіз тощо. CUDA має можливість підключатися до програм, які використовують OpenGL і Direct3D. CUDA — це кросплатформне програмне забезпечення для операційних систем, таких як Linux, Mac OS X і Windows.

2.4 Опис використаних технологій та мов програмування

Для програмної реалізації даної інформаційної системи була використана мова програмування C++. За середовище розробки програмного забезпечення був обраний Microsoft Visual Studio.

C++ в даний час є найпопулярнішою і перспективною мовою програмування галузі. Він містить найповніший набір функцій і можливостей, розроблених за всю історію 41 мови програмування. Найважливішими особливостями C++ є, перш за все, потужна підтримка об'єктно-орієнтованого підходу до розробки програм і механізму параметризації типів і алгоритми. Широкий асортимент типів і розроблені варіанти оформлення нестандартних типів дозволяють відповідне відображення характеристик предмета; суворі правила роботи з константними типами сприяють надійності програми.

Підвищення надійності створених програм — це простий і гнучкий пристрій для вирішення виняткових ситуацій. Розширені схеми перетворення та перетворення типів забезпечують достатній компроміс між суворим типізацією та ефективністю виконання програми. Інструменти явного керування обсягом надають зручний механізм для створення великих програм. C++ є прямим наступником C і фактично включає його як

підмножину. Таким чином, C++ повністю включає в себе усталену традиційну лінгвістичну модель обчислень, включаючи розроблену загальну алгоритмічну основу, широкі можливості створення нових типів і гнучкі способи роботи з пам'яттю, включаючи арифметику над покажчиками. Це гарантує збереження мільйонів рядків тексту в програмі C і забезпечує додаткові гарантії для широкого використання C ++. На додаток до широко поширеного й популярного програмування навіть вдома, C++ є технологічною основою для перспективної нещодавно з'явилася парадигми - узагальненого програмування. Основним інструментом реалізації узагальненого програмування на C++ є шаблонний механізм, а переконливим прикладом використання цієї парадигми є стандартна бібліотека шаблонів (STL), розроблена А. Степановим і М., що стала частиною стандарт бібліотеки C + у 1994 році. +. Ще однією обставиною, яка зумовила вибір C++, стало прийняття наприкінці 1998 року міжнародного мовного стандарту ANSI/ISO. Реальність стандартизації для такої великої, складної та сучасної мови, як C ++, неможливо переоцінити. Коротше кажучи, C++ стає інструментом для промислового програмування в усьому світі. Залучення все більшої кількості корпоративних розробників до використання стандартних рішень і стандартних інструментів дає сильну впевненість в успішних перспективах C++, принаймні, протягом наступних десяти-п'ятнадцяти років (приблизний час для створення мов програмування).

CUDA C - це мова програмування із синтаксисом C. Концептуально він сильно відрізняється від C. Проблема, яку він намагається вирішити, – це кодування кількох (схожих) команд для кількох процесорів. CUDA пропонує більше векторної обробки з одиночною інструкцією з декількома даними (SIMD), але потоки даних → потоки команд або набагато менші вигоди.

CUDA надає деякі механізми для цього та приховує деякі складнощі. CUDA не оптимізовано для декількох різних потоків команд, таких як багатоядерний x86. CUDA не обмежується одним потоком команд, наприклад, векторними інструкціями x86, або обмежений конкретними типами даних, такими як векторні інструкції x86. CUDA підтримує цикли, які можуть виконуватися паралельно. Це його найважливіша особливість. Система CUDA розділятиме виконання "циклів" і запускати тіло "loop" одночасно через масив ідентичних процесорів, забезпечуючи при цьому деяку ілюзію звичайного послідовного циклу (зокрема, CUDA управляє індексом циклу). Розробник повинен знати про структуру машини GPU, щоб ефективно писати "цикли", але майже все керування обробляється часом виконання CUDA. Ефект завершується сотнями (або навіть тисячами) "циклів" одночасно з одним "циклом".

CUDA підтримує те, що виглядає як гілки if. Тільки процесори, що працюють з кодом, який відповідає тесту if, можуть бути активними, тому підмножина процесорів буде активним для кожної "гілки" тесту if. Як приклад це if... else if... else..., має три гілки. Кожен процесор буде виконувати лише одну гілку і "повторно синхронізуватися", готуючись до роботи з іншими процесорами, коли це буде завершено. Можливо, деякі умови розгалуження не відповідають будь-якому процесору. Тому немає необхідності виконувати цю гілку (для цього прикладу три гілки – найгірший випадок). Потім виконується лише одна або дві гілки, виконуючи все if швидше.

CUDA не бере старого коду C/C++ і автоматично запускає обчислення в масиві процесорів. CUDA може компілювати і запускати звичайні C і більшу частину C++ послідовно, але цього дуже мало, тому що воно працюватиме послідовно та повільніше, ніж сучасний процесор. Це означає, що код у деяких бібліотеках поки що не підходить для CUDA. Програма CUDA могла одночасно працювати з бітовими векторами з кількома kByte.

CUDA не може автоматично модифікувати існуючий послідовний код бібліотеки C/C++ на щось, що б це зробило. CUDA надає відносно простий спосіб написання коду, використовуючи знайомий синтаксис C/C++, додає кілька додаткових понять та генерує код, який працюватиме через масив процесорів. Він може дати набагато більше, ніж 10-кратне прискорення проти, наприклад. багатоядерний x86.

Для кращої продуктивності CUDA бажає отримати інформацію під час компіляції. Таким чином, механізми шаблонів є найкориснішими, оскільки він дає розробнику спосіб сказати речі під час компіляції, які може використовувати компилятор CUDA. Як простий приклад, якщо матриця визначена (інстанціюється) під час компіляції як 2D і 4 x 8, тоді компилятор CUDA може працювати з цим, щоб організувати програму через процесори. Якщо цей розмір динамічний і змінюється під час роботи програми, набагато складніше виконати компілятор або систему часу виконання.

Вважаю, що реалізація шаблонів CUDA на GPU не є повною w.r.t. C++. Найскладніший матеріал, який ефективно працює на декількох процесорах, – це динамічний розгалуження багатьох альтернативних шляхів, тому що це ефективно серіалізує код; у гіршому випадку може працювати лише один процесор за раз, що забирає вигоду від використання графічного процесора. Таким чином, віртуальні функції здаються дуже складними.

Є деякі дуже розумні інструменти для аналізу усієї програми, які можуть виводити набагато більше інформації про тип, ніж може зрозуміти розробник. Існуючі інструменти можуть вивести достатньо усунення віртуальних функцій і, отже, перемістити аналіз розгалуження для компіляції часу. Існують також методи керування виконанням програми, які безпосередньо пов'язані з перекомпіляцією програм, які можуть досягти кращих рішень розгалуження. AFAIK (за модулем зворотний зв'язок) компилятор CUDA ще є сучасним у цих областях.

Microsoft Visual Studio - це програмне середовище розробці додатків для ОС Windows, як консольних, так і з графічним інтерфейсом.

Інтегроване середовище розробки (IntegratedDevelopmentEnvironment - IDE) Visual Studio пропонує ряд високорівневих функціональних можливостей, що виходять за рамки базового керування кодом.

Нижче наведено основні переваги IDE-середовища Visual Studio.

- Вбудований Web-сервер

  Наявність інтегрованого веб-сервера у Visual Studio дозволяє запускати веб-сайт безпосередньо із середовища розробки, а тестовий сервер може приймати лише внутрішні комп'ютерні з'єднання, що підвищує безпеку, за винятком випадків доступу до тестового сайту із зовнішнього комп'ютера;

- Підтримка безлічі мов під час розробки.

  Visual Studio дозволяє писати код вашою мовою або іншою мовою, використовуючи той самий інтерфейс (IDE). Єдиним недоліком є те, що веб-сайт можна використовувати лише однією мовою (інакше проблем зі компіляцією не уникнути);

- Менше за код для написання.

  Більшість програм вимагає відповідної кількості стандартного стереотипного коду та веб-сайту ASP. NET не є винятком. У Visual Studio такі деталі розміщуються автоматично;

- Інтуїтивний стиль кодування

  За замовчуванням Visual Studio форматує код, коли ви його вводите, автоматично вставляючи інструменти та використовує кольорове кодування для виділення таких елементів, як коментарі. Такі невеликі відмінності полегшують читання коду та зменшують кількість помилок. Параметри форматування, які використовує Visual Studio, можна навіть автоматично коригувати, що дуже зручно, якщо

виробник віддає перевагу іншому стилю ложки (наприклад, стиль K&R, де ложки розміщуються в рядку оголошень);

- Вища швидкість розробки.

  Більшість функцій Visual Studio призначені для того, щоб допомогти розробнику виконати роботу якомога швидше. Зручні функції, такі як IntelliSense (який виявляє помилки та пропонує правильні параметри), пошук і заміну (що дозволяє знайти ключові слова в одному файлі або проекті), а також додавати та видаляти коментарі (це може бути тимчасовим прихованим блоком). (з коду) дозволяє процесору працювати швидко та ефективно;

- Можливості налагодження.

  Інструменти редагування Visual Studio — найкращий спосіб виявити загадкові помилки та діагностувати дивну поведінку. Розробник може виконувати свій код рядок за рядком, позначати точки переривання, зберігати їх за бажанням для майбутнього використання та переглядати поточні дані з пам’яті в будь-який час.

2.6 Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними інформаційної системи можуть бути будь-які цілі числа.

Вихідними даними є результат обчислень.

2.6. Опис роботи розробленого програмного продукту

2.6.1. Використані технічні засоби

Для розробки програмного забезпечення використовувався комп’ютер з наступними параметрами:

– операційна система Windows 10;

– процессор Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz 3.41GHz;

– оперативна пам'ять 16 ГБ ОЗУ;

– відеокарта Nvidia GeForce GTX 1060 3GB;

– жорсткий диск 130 ГБ;

– DirectX версії 9.0c.

2.6.2. Використані програмні засоби

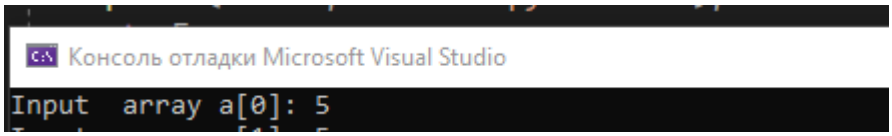
Программа була написана за допомогою мови програмування C++ у середовищі розробки програмного забезпечення Microsoft Visual Studio.

2.6.3. Виклик та завантаження програми

Для використання програми необхідно отримати доступ до папки потрібно відкрити CudaRuntime2\x64\Debug\CudaRuntime2.exe

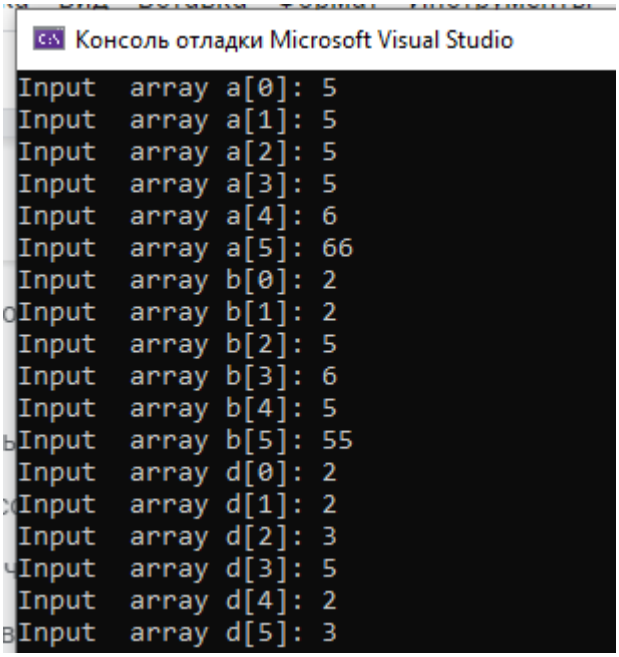
2.6.4. Опис інтерфейсу користувача

Розроблена програма використовує консольний додаток, який завантажується при відкритті файлу CudaRuntime2.exe.(рис. 2.15)



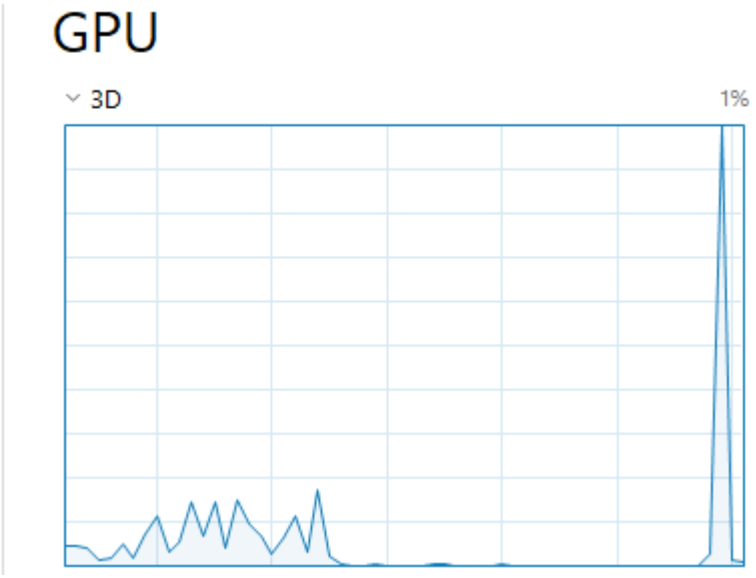
(рис. 2.15)

Користувач вручну вводить зміст масиву (рандомні числа).(рис. 2.16)



(рис. 2.16)

Система починає процес обрахування програми на графічному процесорі(рис. 2.17) за відповідним алгоритмом(рис. 2.18):



(рис. 2.17)

```
{5,5,5,5,6,66} + {2,2,5,6,5,55} + {2,2,3,5,2,3} = {9,9,13,16,13,124}
{5,5,5,5,6,66} * {2,2,5,6,5,55} * {2,2,3,5,2,3} = {20,20,75,150,60,10890}
```

Результат роботи (рис. 2.18)

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 250;
2. коефіцієнт складності програми – 1,2;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста – 55 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 15 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) Q = q \cdot C \cdot (1 + p), \quad (3.2)$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 250 \cdot 1{,}2 \cdot (1 + 0{,}05) = 315$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \quad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{315 \cdot 1{,}2}{85 \cdot 1{,}2} = 3{,}7 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K}, \tag{3.4}$$

$$t_a = \frac{315}{23 \cdot 1,2} = 11,41 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K}, \tag{3.5}$$

$$t_n = \frac{315}{21 \cdot 1,2} = 12,5 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K}, \tag{3.6}$$

$$t_n = \frac{315}{5 \cdot 1,2} = 52,5 \text{ людино-годин,}$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,2 \cdot t_{отл}, \tag{3.7}$$

$$t_{отл}^{k} = 1,2 \cdot 52,5 = 63 \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{дp} + t_{до}, \tag{3.8}$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{д} = \frac{Q}{(15...20)\cdot K}, \quad (3.9)$$

$$t_{б} = \frac{315}{20\cdot 1,2} = 13,12 \text{ людино-годин.}$$

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 \cdot t_{др}, \quad (3.10)$$

$$t_{бо} = 0,75 \cdot 11,41 = 8,55 \text{людино-годин.}$$

$$t_{б} = 11,41 + 8,55 = 19,96 \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 3,7 + 11,41 + 52,5 + 63 + 19,96 = 200,97$$

людино-годин.

У результаті ми розрахували, що в загальній складності необхідно близько 200,97 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ включають витрати на заробітну плату виконавця програми З/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ} \text{ грн.}, \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР} \text{ грн.}, \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 200{,}97 \cdot 55 = 11053{,}35 \text{грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot С_{М} \text{ грн.}, \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 63 \cdot 15 = 945 \text{ грн.}$$

$$К_{ПО} = 11053{,}25 + 945 = 11998{,}25 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.}, \quad (3.14)$$

де B_k- число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$T = \frac{200,97}{1 \cdot 176} = 1,1 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 200,97 близько людино-години. Тобто, ймовірна очікувана тривалість розробки складатиме 1,1 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 11998,25 грн.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено консольний застосунок, використовуючи об’єктно-орієнтовані підходи.

Перевагами використання таких підходів до створення програми є легкість написання коду та структурність розробленого додатку.

Додаток призначений для того, щоб розробити та показати необхідність користування графічними процесорами у додатках, які направлені на виконання складних математичних операцій за короткий проміжок часу.

При використанні додатку користувач може вводити числа, аби виконати математичні операції.

У результаті було отримано застосунок, який демонструє цінність переходу розробниками орієнтованих з CPU до GPU, тому що такий підхід заощаджує час та кошти у розробці.

Під час виконання даного дипломного проекту були виконані пройдені наступні етапи створення програмного продукту:

- аналіз предметної області задачі, що розв’язується;
- обрання раціональної архітектури та технології створення додатку;
- написання програмного коду;
- розробка рекомендацій щодо використання застосунку.

Під час виконання кваліфікаційної роботи також було визначено трудомісткість розробленого програмного продукту (200,97 людино-годин), проведений підрахунок вартості роботи по створенню програми (11998,25 грн) та розраховано час на його створення (1.1 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sanders, J. and E. Kandrot, 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, pp: 320.
2. Dr, B. T. Hands-On GPU Programming with Python and CUDA: Explore high-performance parallel computing with CUDA / B. T. Dr. — 2018 г. : Packt Publishing, . — 310 с. — Текст : непосредственный.
3. Nikolaos, Ploskas GPU Programming in MATLAB / Ploskas Nikolaos, Samaras Nikolaos. — 2016 : Morgan Kaufmann;, . — 318 с.

4. Gregory, Ruetsch CUDA Fortran for Scientists and Engineers: Best Practices for Efficient CUDA Fortran Programming / Ruetsch Gregory, Fatica Massimiliano. — 2013 : Morgan Kaufmann; , . — 338 с.
5. Офіційна документація NvidiaCuda. https://developer.nvidia.com/cuda-gpus
6. . Luebke D., Harris M., Kruger J., и др. GPGPU: General Purpose Computation On Graphics Hardware - SIGGRAPH, 2005. – 277 с.
7. Bott A. A positive definite advection scheme obtained by nonlinear renormalization of the advective fluxes - Monthly Weather Review , 1998
8. Göddeke D., GPGPU. – URL: http://gpgpu.org/
9. Сандерс Д., Кэндрот Эю. NVIDIA CUDA в примерах: введение в программирование графических процессоров – ДМК Пресс -2011.
10. Dawn, Griffiths Head First C: A Brain-Friendly Guide Paperback – Illustrated / Griffiths Dawn. — 2012 : O'Reilly and Associates, . — 629 с.
11. Andrew, Koenig Accelerated C++: Practical Programming by Example (Addison-Wesley C++ In-Depth) / Koenig Andrew. — 2000 : Addison-Wesley Professional, . — 352 с.
12. Bjarne, Stroustrup Programming: Principles and Practice Using C++ / Stroustrup Bjarne. — 2008 : Addison-Wesley Longman, Amsterdam, . — 1236 с.
13. Стаття про GPU. https://www.ixbt.com/video3/cuda-1.shtml
14. King, K. N. C Programming: A Modern Approach Paperback – Illustrated / K. N. King. — 2008 : W W NORTON & CO, . — 832 с.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
"Розробка програмного забезпечення для математичних обчислень на GPU мовою програмування C++"
студента групи 121-18-2 Педенко Данило Ігоровича

Керівник економічного розділу
Зав. каф. ПЕП та ПУ, д.е.н **О.Г.Вагонова**

ДОДАТОК В

Перелік файлів на диску

ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Педенко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Педенко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Педенко.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Педенко.ppt	Презентація кваліфікаційної роботи