Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

# Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка програмного комплексу (пакету програм) для навчальної системи з методів паралельного програмування»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1812»

_____ /Сергій БАГНО/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /ст.викл. Сергій САМОЙЛОВ/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОПЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

# Explanatory Note
to Bachelor's Thesis

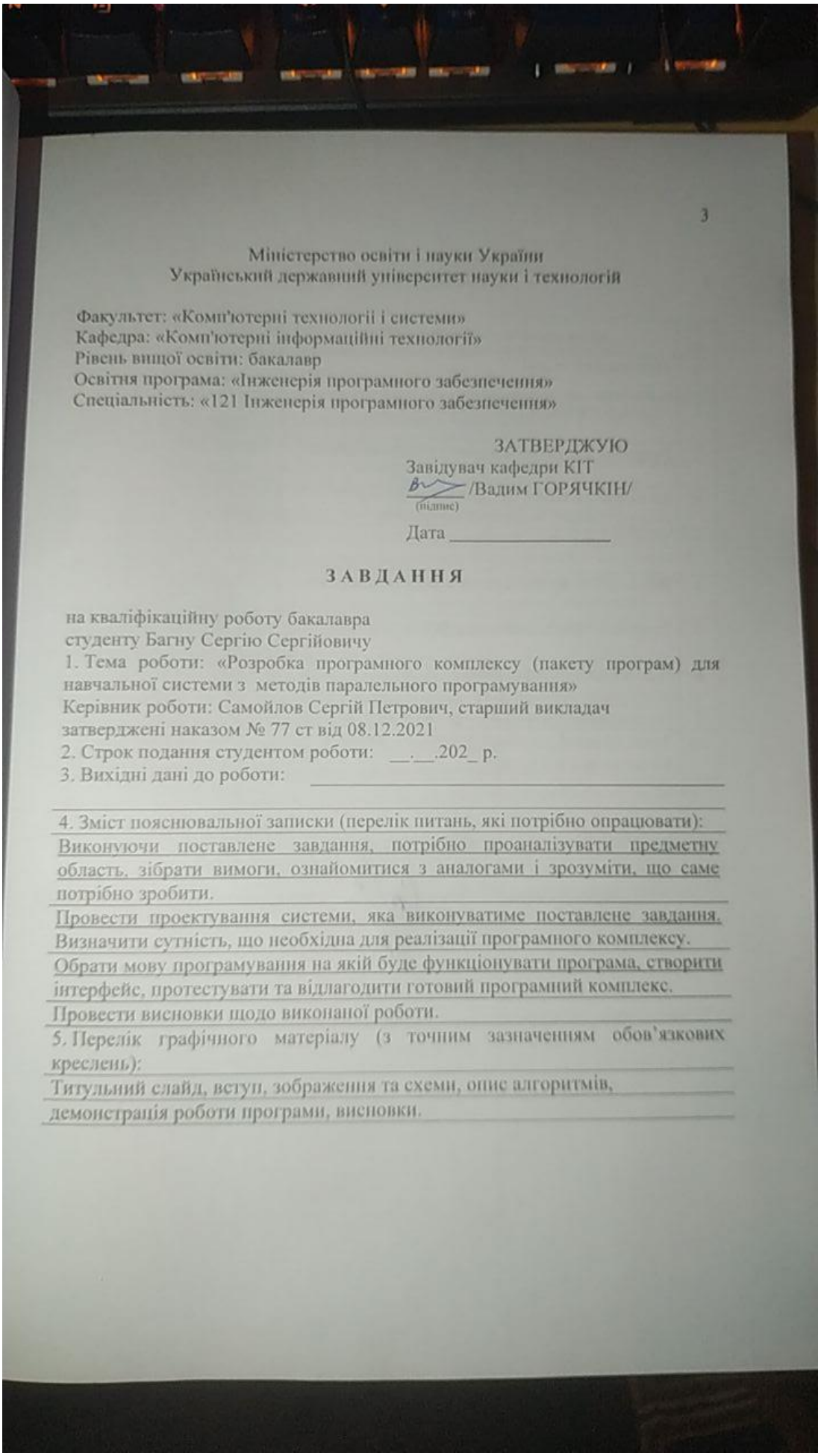
on the topic: «Development of a software package (software package) for a training system on parallel programming methods»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group ПІ31812: /Serhii BAHNO/

Scientific Supervisor: /Serhii SAMOILOV/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022





Міністерство освіти і науки України
Український державний університет науки і технологій

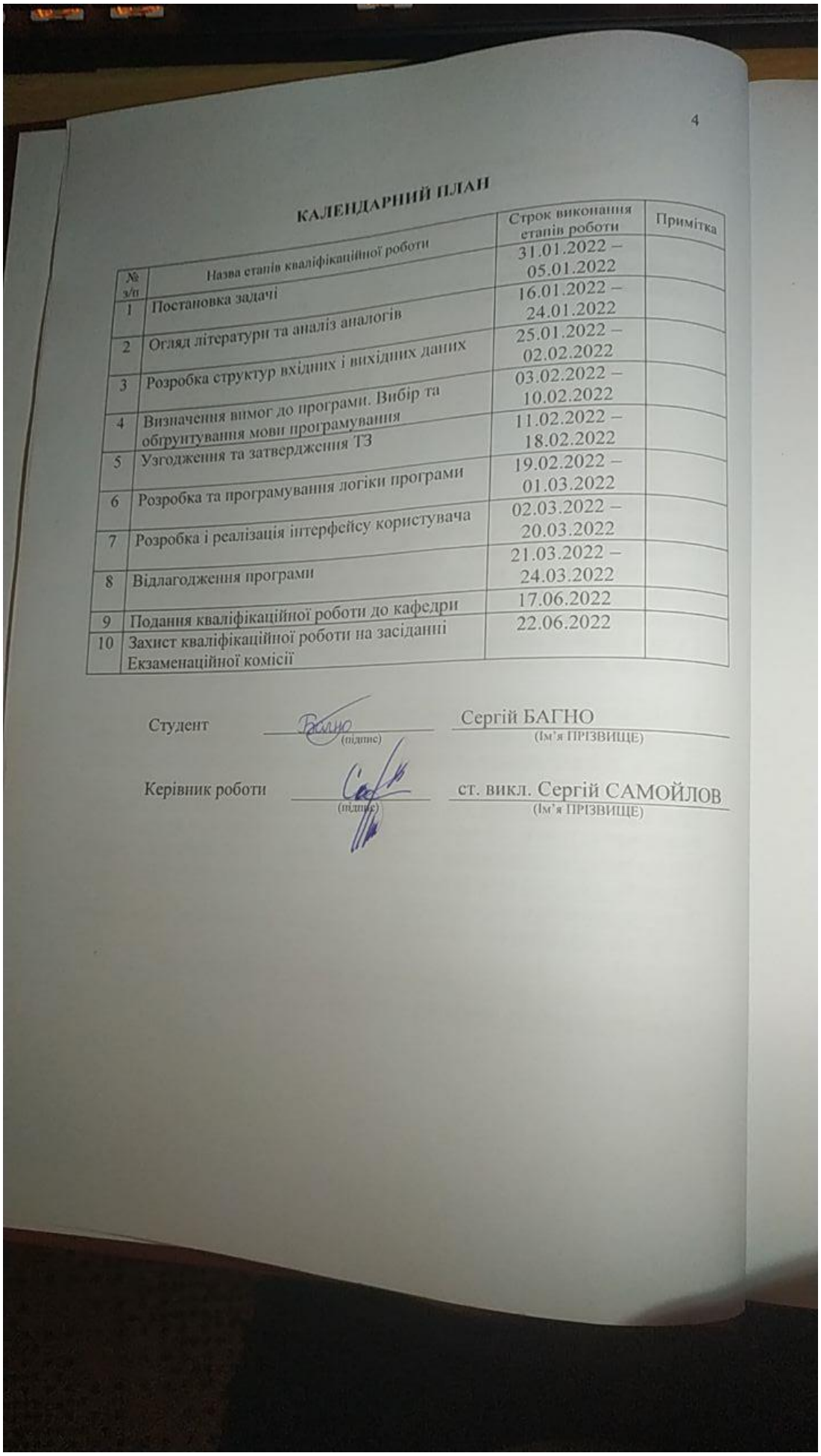
Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)
Дата _____

**З А В Д А Н Н Я**

на кваліфікаційну роботу бакалавра
студенту Багну Сергію Сергійовичу
1. Тема роботи: «Розробка програмного комплексу (пакету програм) для навчальної системи з методів паралельного програмування»
Керівник роботи: Самойлов Сергій Петрович, старший викладач
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: __.__.202_ р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Виконуючи поставлене завдання, потрібно проаналізувати предметну область, зібрати вимоги, ознайомитися з аналогами і зрозуміти, що саме потрібно зробити.
Провести проектування системи, яка виконуватиме поставлене завдання. Визначити сутність, що необхідна для реалізації програмного комплексу.
Обрати мову програмування на якій буде функціонувати програма, створити інтерфейс, протестувати та відлагодити готовий програмний комплекс.
Провести висновки щодо виконаної роботи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Титульний слайд, вступ, зображення та схеми, опис алгоритмів, демонстрація роботи програми, висновки.



4

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Постановка задачі | 31.01.2022 – 05.01.2022 | |
| 2 | Огляд літератури та аналіз аналогів | 16.01.2022 – 24.01.2022 | |
| 3 | Розробка структур вхідних і вихідних даних | 25.01.2022 – 02.02.2022 | |
| 4 | Визначення вимог до програми. Вибір та обгрунтування мови програмування | 03.02.2022 – 10.02.2022 | |
| 5 | Узгодження та затвердження ТЗ | 11.02.2022 – 18.02.2022 | |
| 6 | Розробка та програмування логіки програми | 19.02.2022 – 01.03.2022 | |
| 7 | Розробка і реалізація інтерфейсу користувача | 02.03.2022 – 20.03.2022 | |
| 8 | Відлагодження програми | 21.03.2022 – 24.03.2022 | |
| 9 | Подання кваліфікаційної роботи до кафедри | 17.06.2022 | |
| 10 | Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії | 22.06.2022 | |

Студент _____ (підпис) Сергій БАГНО (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) ст. викл. Сергій САМОЙЛОВ (Ім'я ПРІЗВИЩЕ)

# РЕФЕРАТ

Пояснювальна записка складається із вступу, основної частини, висновків, списку використаних джерел та додатків.
– вступ. У вступі описується сутність розробки, її актуальність, визначається предмет, об’єкт, мета і завдання дослідження. Складається із 2 сторінок;
Основна частина містить три розділи.
– розділ 1. Теоретичні основи паралельного програмування. Огляд наявних аналогів. Цей розділ є теоретичним, у ньому досліджується предметне середовище, описується процес діяльності, надається огляд і аналіз середовищ для розробки застосунка та наявних застосунків для навчальної системи з методів паралельного програмування. Складається із 21 сторінки;
– розділ 2. Опис програмного за стосунку. Цей розділ є проектним, в ньому розробляється технічне завдання, визначаються вимоги до технічного забезпечення, надається опис архітектури програмного забезпечення. Складається із 23 сторінок;
– розділ 3. Етапи розробки і створення програмного додатку для навчальної системи з методів паралельного програмування. Цей розділ є практичним, в ньому описані застосовувані алгоритми, основні структури даних, особливості реалізації окремих методів, представлено функціонал розробленого додатку, проведено дослідження щодо ефективності розпаралелювання на різних системах за допомогою розробленого додатку. Складається із 38 сторінок.
– висновки. У висновках узагальнюються результати проведеного дослідження. Складається із 5 сторінок;
– список використаних джерел – включає в себе бібліографічний список використаної літератури та посилань. Складається із 2 сторінок;
– додатки – містять код основних модулів.
Кількість таблиць: 6 штук.
Кількість рисунків: 51 штука.
Ключові слова: паралельне програмування, багатопоточність, бібліотека TPL, бібліотека PLINQ, низькорівнева синхронізація.

# ЗМІСТ

# ВСТУП

*Актуальність теми дослідження*. За останні десятиліття ми стали свідками подальшого прогресу мікроелектроніки та випереджального розвитку багатопроцесорних та багатоядерних обчислювальних систем, включаючи хмарні обчислення. При цьому перед розробниками програмного забезпечення було поставлено завдання створення ефективних паралельних додатків.

Успішність розпаралелювання визначається прискоренням програми щодо її послідовної версії. Тут на перший план виступають сучасні математичні і алгоритмічні методи організації обчислень, що потребує високого рівня кваліфікації розробників ПЗ.

Разом з тим, незважаючи на великі обсяги наукових та дослідницьких робіт у галузі паралельних обчислень, завжди існує потреба в інтерактивних додатках, які поєднували б у собі подачу теоретичного матеріалу, демонстрацію переваг окремих методів розпаралелювання при вирішенні конкретних завдань та перевірку отриманих знань. У цій роботі пропонується саме такий додаток.

*Предметом дослідження* є паралелізм як основа підвищення продуктивності сучасних багатоядерних систем.

*Об'єктом дослідження* виступають особливості реалізації паралельних обчислень в ОС Windows з використанням технологій .NET.

*Мета дослідження* – розглянути теоретичні основи розпаралелювання обчислень, вивчити особливості технології паралельного програмування на платформі .NET, вирішити практично ряд завдань і розробити навчально-методичний комплекс для перевірки знань студентів з даної тематики.

Відповідно до визначеного предмету, об’єкту та мети, перед роботою поставлені такі *завдання*:

— в теоретичному розділі: дослідити предметне середовище, описати процес діяльності, надати огляд середовищ для розробки застосунка та огляд наявних застосунків для навчальної системи з методів паралельного програмування;
— в проектному розділі: розробити технічне завдання, сформулювати вимоги до технічного забезпечення, дати опис архітектури програмного забезпечення;
— в практичному розділі: описати застосовувані алгоритми, основні структури даних, особливості реалізації окремих методів, представити функціонал розробленого додатку, провести дослідження щодо ефективності розпаралелювання на різних системах за допомогою розробленого додатку.

*Методи досліджень*. Для виконання поставлених завдань були використані методи теорії обчислювальних систем, теорії графів, теорії множин, а також емпіричного узагальнення на основі даних, отриманих під час випробувань.

*Наукова новизна* роботи полягає у проведеному аналізі залежності швидкостей розроблених паралельних додатків від різних конфігурацій ПК та збірок на IDE різного року випуску (Visual Studio 2015 і 2017).

*Наукова та практична значимість*. Проведене дослідження буде корисним для науковців, розробників, аспірантів та студентів, що спеціалізуються або цікавляться проблематикою паралельних обчислень. Запропонований навчально-методичний комплекс може широко застосовуватися як у практиці викладання, так і в організації самостійного навчання.

*Структура дослідження*. Робота складається з вступу, основної частини із трьох розділів, у кожному з яких послідовно вирішуються поставлені завдання; наприкінці кожного розділу і в кінці роботи наводяться висновки, де узагальнено результати дослідження, подано список використаної літератури та додатки.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ З ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ. ОГЛЯД НАЯВНИХ АНАЛОГІВ

## 1.1. Теоретичні основи

### 1.1.1. Опис предметного середовища

Не зважаючи на бурхливий розвиток обчислювальних систем, вирішувані завдання давжди його випереджають за складністю, а отже, досягти необхідної продуктивності можна лише структурними методами.

Ще в XX сторіччі виникла ідея використання у обчислювальних комплексах замість розділеної оперативної пам'яті (ОП) загальної (розподіленої) ОП, яка пов'язує кілька центральних процесорів. Так вирішувалася проблема витрат часу на обмін даними. Відповідно, з’явилося поняття паралельної обчислювальної системи з наступними рівнями паралелізму:

- паралелізм на рівні команд (InstructionLevelParallelism, ILP) дозволяє процесору виконувати кілька команд за один такт. Залежності між командами обмежують кількість доступних до виконання команд і знижують обсяг паралельних обчислень. Технологія ILP дозволяє процесору переупорядковувати команди оптимальним чином, щоб унеможливити зупинення обчислювального конвеєра та збільшити кількість виконуваних команд за один такт. Сучасні процесори підтримують певний набір команд, які можуть виконуватись паралельно.
- паралелізм на рівні потоків процесу дозволяє виділити незалежні потоки виконання команд у межах одного процесу. Потоки підтримуються лише на рівні операційної системи. Операційна система розподіляє потоки процесів по ядрах процесора з урахуванням пріоритетів. За допомогою потоків програма може максимально використовувати вільні обчислювальні ресурси.

– паралелізм на рівні додатків. Одночасне виконання кількох програм здійснюється у всіх операційних системах, що підтримують режим розподілу часу. Навіть на однопроцесорній системі незалежні програми виконуються одночасно. Паралельність досягається з допомогою виділення кожному додатку кванта процесорного часу.

У 1966 р. М.Флінн (США) запропонував наступну класифікацію обчислювальних систем на основі взаємодії потоку команд та потоку даних (операндів та результатів):

– SISD (Single Instruction, Single Data) - системи, в яких існує одиночний потік команд та одиночний потік даних.
– SIMD (Single Instruction, Multiple Data) – системи з одиночним потоком команд та з множинним потоком даних.
– MISD (Multiple Instructions, Single Data) - системи, в яких існує множинний потік команд і одиночний потік даних
– MIMD (Multiple Instructions, Multiple Data) – системи з множинним потоком команд та множинним потоком даних; саме до даного класу належить більшість паралельних обчислювальних систем.

Для виділення різних типів паралельних обчислювальних систем застосовується класифікація Джонсона, у якій подальший поділ багатопроцесорних систем ґрунтується на використовуваних способах організації оперативної пам'яті (рис.1.1).

Класифікація Джонсона заснована на структурі пам'яті (global - глобальна або distributed - розподілена) і механізмі комунікацій та синхронізації (shared variables - змінні, що розділяються, або message passing - передача повідомлень). Системи GMSV (global-memory-shared-variables) часто називаються також мультипроцесорами з пам'яттю (shared-memory multiprocessors). Системи DMMP (distributed-memory-message-passing) також називають мультикомп'ютерами з розподіленою пам'яттю (distributed-memory multicomputers).

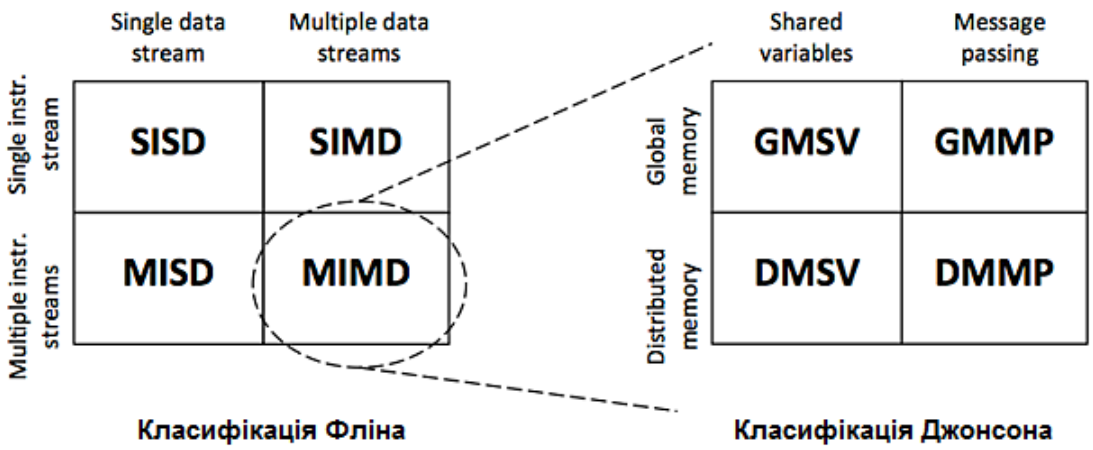



Рисунок 1.1 — Класифікації обчислювальних систем за Фліном та Джонсоном

Таким чином, при розгляді проблеми організації паралельних обчислень слід розрізняти наступні можливі режими виконання незалежних частин програми:

1) Режим поділу часу (багатозадачний режим)

Режим поділу часу передбачає, що число підзадач (процесів або потоків одного процесу) більше, ніж кількість виконавчих пристроїв. Даний режим є псевдопаралельним, коли активною може бути лише одна підзадача, а всі інші процеси (потоки) перебувають у стані очікування своєї черги на використання процесора.

2) Розподілені обчислення

Задачі виконуються на машинах, пов'язаних локальною або глобальною мережею. Багато розподілених систем організовані як системи типу клієнт-сервер. Компоненти розподілених систем часто є багатопотоковими.

3) Синхронні паралельні обчислення

Програмне забезпечення з використанням такої архітектури потребує ефективного використання доступних обчислювальних ресурсів системи. Число підзадач має бути оптимізовано з урахуванням кількості виконавчих пристроїв у системі (процесорів та/або ядер процесорів).

### 1.1.2 Опис процесу діяльності

Розробка паралельних додатків для вирішення складних науково-технічних завдань розбивається на такі етапи (рис. 1.2):

- етап аналізу. Виконати аналіз наявних обчислювальних схем та здійснити їх поділ (декомпозицію) на частини (підзадачі), які можуть бути реалізовані значною мірою незалежно один від одного;
- етап дизайну. Виділити для сформованого набору підзадач інформаційні взаємодії, які мають здійснюватися в ході вирішення вихідного завдання;
- етап реалізації. Визначити доступну для вирішення задачі обчислювальну систему та виконати розподіл набору підзадач між процесорами системи:
    - розділити обчислення на незалежні частини;
    - визначити інформаційну взаємодію між частинами;
    - провести масштабування завдань;
    - здійснити розподіл всіх завдань між процесорами.
- етап налагодження. Виправити помилки, перевірити правильність роботи програми та її відповідність вимогам ТЗ..
- етап тестування та налаштування. Робочі характеристики та правильність роботи розпаралеленого додатка перевіряються на етапі тестування наступним чином:
    - використовується аналізатор потоків (наприклад, Intel Thread Checker) для перевірки правильності роботи розпаралеленої програми за умови, що послідовне виконання програми завершується коректно.

о для оцінки можливого приросту продуктивності порівнюють продуктивність багатопотокового додатка з продуктивністю послідовного додатка.

о виявлені проблеми усуваються шляхом вибору оптимального інтерфейсу програмування багатопотокових додатків або шляхом внесення змін до структури програми.

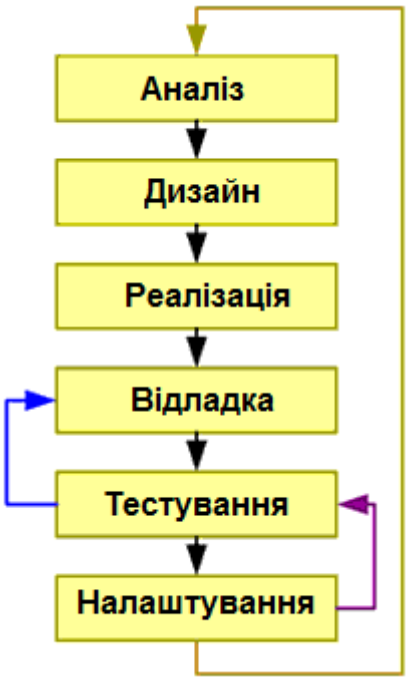



Рисунок 1.2 — Блок-схема типового циклу розробки паралельної програми

Для підвищення ефективності паралельної програми використовуються наступні рішення:

- довжини паралельно виконуваних гілок мають бути якомога рівнішими між собою.
- виключаються простої через очікування даних, передачі управління та виникнення конфліктів під час використання спільних ресурсів.
- обмін даними синхронізується із обчисленнями[1].

Контрольні питання, виходячи з яких проводиться оцінка ефективності паралельного додатку на етапі виконання підзадач, представлені в табл.1.1.

[1] Duffy, Joe. Concurrent Programming on Windows. – с. 109

Таблиця 1.1 — Критерії ефективності паралельного додатку на етапі виконання підзадач[2]

| Підзадача етапу виконання | Критерії |
|---|---|
| 1 | 2 |
| Розділити обчислення на незалежні частини | – чи не збільшує виконана декомпозиція обсяг обчислень та обсяг пам'яті, що використовується?<br>– чи можливе при вибраному способі декомпозиції рівномірне завантаження всіх наявних процесорів?<br>– чи достатньо виділених частин процесу обчислень для ефективного завантаження наявних процесорів (з урахуванням можливості збільшення їх кількості)? |
| Визначити інформаційну взаємодію між частинами | – чи відповідає обчислювальна складність підзадач інтенсивності їх інформаційних взаємодій?<br>– чи однакова інтенсивність інформаційних взаємодій для різних підзадач?<br>– чи є схема інформаційної взаємодії локальною?<br>– чи не перешкоджає виявлена інформаційна залежність паралельному виконанню підзадач? |
| Провести масштабування задач | – чи не погіршиться локальність обчислень після масштабування наявного набору підзадач?<br>– чи мають підзадачі однакову обчислювальну та комунікаційну складність після масштабування?<br>– чи відповідає кількість задач числу наявних процесорів?<br>– чи залежать правила масштабування параметрично від кількості процесорів? |
| Здійснити розподіл усіх завдань між процесорами | – чи не призводить розподіл кількох задач на один процесор до зростання додаткових обчислювальних витрат?<br>– чи існує необхідність динамічного балансування обчислень?<br>– чи процесор-сервер не є «вузьким» місцем при використанні схеми «сервер-клієнт»? |

Як зазначалося вище, досягнення високої ефективності при розробці реальних паралельних програм вимагає багаторазових змін програми з метою

[2] Ringler R. C# Multithreaded and Parallel Programming – с.116

пошуку найкращої схеми її розпаралелювання. Успішність такого пошуку визначається простотою модифікації програми та наявними у проекта ресурсами.

## 1.2 Ознайомлення з середовищами для розробки застосунка

В даній роботі застосунок буде розроблятися для ОС Windows з використанням технологій .NET на мові C#.

Windows називають багатопотоковою операційною системою з витісняючою багатозадачністю, тому що кожен потік може бути зупинений у довільний момент часу і замість нього обраний для виконання інший.

Під «потоком» розуміють «потік команд», тобто послідовність інструкцій, які зчитує та виконує процесор. Всі сучасні ОС дозволяють запустити багато потоків, які виконуватимуться паралельно. Безперечно, один процесор може одночасно виконувати лише один потік інструкцій. Тому "справжня" багатопоточність, коли потоки команд виконуються дійсно паралельно і незалежно, можлива лише в багатопроцесорній машині. Зазвичай число потоків у системі набагато більше числа, і ОС доводиться емулювати багатопоточність, змушуючи процесор по черзі перемикатися між потоками (рис. 1.3).

Кожному потоку призначається рівень пріоритету з нульового (найнижчого) до 31 (найвищого). При виборі потоку, який буде переданий процесору, спочатку розглядаються потоки з найвищим пріоритетом та ставляться у чергу у циклі. При виявленні потоку пріоритетом 31 він передається процесору. Після завершення такту відбувається пошук наступного потоку з аналогічним пріоритетом, щоб переключити його контекст. За наявності черги потоків з пріоритетом 31 система ніколи не передасть процесору потік з меншим пріоритетом, що призводить до

зависання (starvation) - потоки з високим пріоритетом споживають практично весь час процесора і не дають виконуватися потокам нижчого пріоритету.

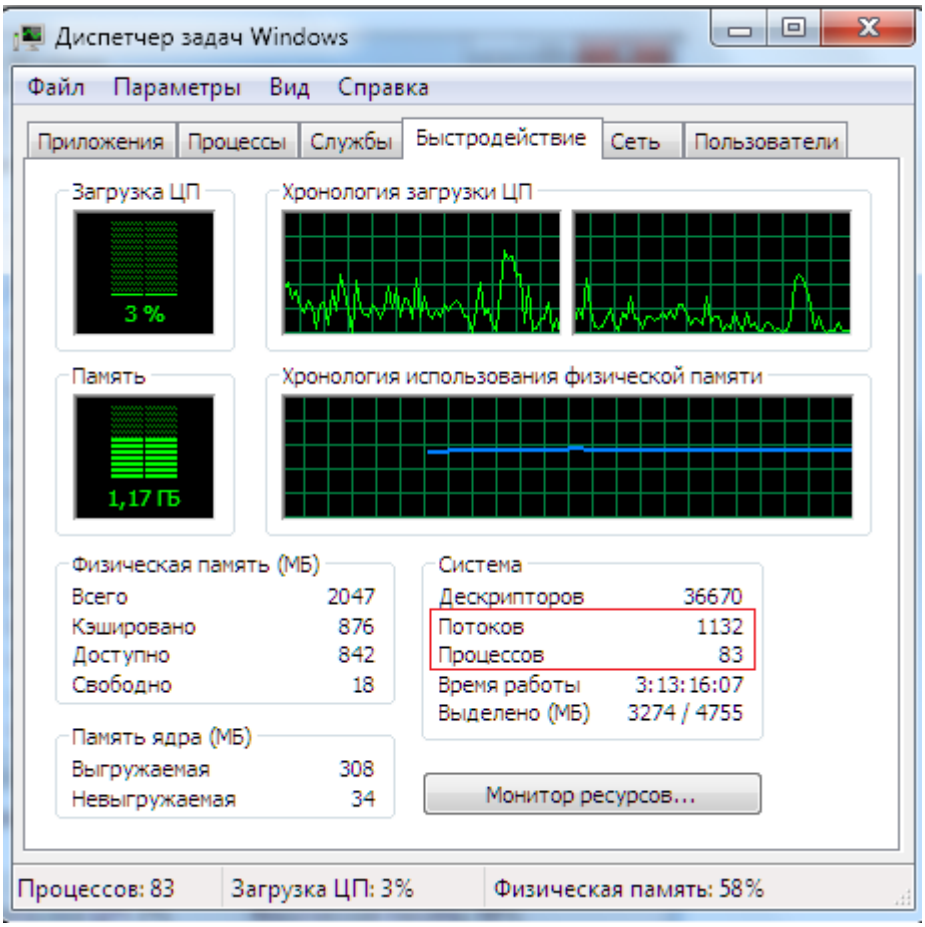



Рисунок 1.3 — Кількість потоків (1132) перевищує кількість процесів (83) у системі

При розробці програми слід вирішити, чи має вона реагувати швидше або повільніше, ніж інші запущені на цій же машині програми. Відповідно до цього рішення вибирається клас пріоритету для процесу. Windows підтримує шість класів пріоритетів: *Idle* (холостого ходу), *Below Normal* (нижче звичайного), *Normal* (звичайний), *Above Normal* (вище звичайного), *High* (високий) і *Real time* (реального часу). За замовчуванням вибирається пріоритет *Normal*.

Також підтримуються сім відносних пріоритетів потоків: *Idle* (холостого ходу), *Lowest* (найнижчий), *Below Normal* (нижче звичайного),

*Normal* (звичайний), *Above Normal* (вище звичайного), *Highest* (найвищий) та *Time-Critical* (що вимагає) негайної обробки). Ці пріоритети співвідносяться із класами пріоритетів процесу наступним чином (табл.1.2).

Таблиця 1.2 — Визначення рівня пріоритету на основі класу пріоритету та відносного пріоритету потоку[3]

| Відносний пріоритет потоку | Клас пріоритету процесу | | | | | |
|---|---|---|---|---|---|---|
| | Idle | Below Normal | Normal | Above Normal | High | Realtime |
| Time-Critical | 15 | 15 | 15 | 15 | 15 | 31 |
| Highest | 6 | 8 | 10 | 12 | 15 | 26 |
| Above Normal | 5 | 7 | 9 | 11 | 14 | 25 |
| Normal | 4 | 6 | 8 | 10 | 13 | 24 |
| Below Normal | 3 | 5 | 7 | 9 | 12 | 23 |
| Lowest | 2 | 4 | 6 | 8 | 11 | 22 |
| Idle | 1 | 1 | 1 | 1 | 1 | 16 |

Нульовий рівень пріоритету. зарезервований для потоку обнулення сторінок, тому система не дозволяє присвоїти його якомусь іншому потоку. Недоступні також рівні пріоритету: 17, 18, 19 , 20, 21, 27, 28, 29 і 30. Вони зарезервовані під драйвери пристроїв, що працюють в режимі ядра, а тому не присвоюються користувачам додатків.

Потік у класі пріоритету Real time не може мати рівень пріоритету нижче 16. У той же час потоки в інших класах пріоритету не можуть отримати рівень вище 15.

У CLR всі потоки поділяються на активні (fogeground) та фонові (background). При завершенні активних потоків у процесі CLR примусова завершує всі запущені на цей момент фонові потоки. При цьому завершення фонових потоків відбувається негайно і без вкидання винятків. Отже, активні потоки можна використовувати для виконання завдань, які обов'язково потрібно завершити, наприклад, для переміщення на диск даних з буфера

[3] Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET – с.765

обміну. Фонові потоки можна залишити для таких некритичних задач, як перерахунок комірок електронних таблиць або індексування записів: ця робота може бути продовжена і після перезавантаження програми.

Існує кілька категорій засобів синхронізації потоків:

- Призупинення виконання потоку (Suspend (), Resume (), Sleep (), Yield (), Join ());
- Блокуючі конструкції;
- Конструкції подачі сигналів;
- Затримуючі засоби синхронізації.

Зупинений потік практично не споживає часу процесора, але його легко відновити системою при часі "пробудження".

Блокуючі конструкції забезпечують винятковий доступ до ресурсу (поле, фрагмент коду). Зокрема, в .NET цю роботу виконують класи Monitor, Mutex, Semaphor, SemaphorSlim, а в мові C# оператор lock.

Синхронізуючі об'єкти ядра важливо доступні потокам будь-яких процесів. Їх недоліком є менша швидкодія, тобто. для роботи з ними потік повинен перейти з режиму користувача в режим ядра, на що потрібно 1000 процесорних такти.

У .NET Framework підтримка паралельного програмування здійснюється за допомогою типів бібліотек класу (TPL) та засобів діагностики (Concurrency Visualizer). Ці можливості полегшують паралельну розробку, у тому числі звільняючи при необхідності від безпосередньої роботи з потоками або пулом потоків.

Також реалізовано ряд низькорівневих конструкцій, що використовуються при звичайній роботі з багатопоточністю:

- Сигнальні конструкції з низьким часом очікування (SemaphoreSlim, ManualResetEventSlim, CountdownEvent та Barrier).
- Маркери відміни (cancellation tokens) для реалізації кооперативної відміни (cooperative cancellation).

– Класи для відкладеної ініціалізації (lazy initialization).
– TreadLocal<T>.

Бібліотека PFX (Parallel Extensions to the .NET Framework) включає два рівні функціональності (рис.1.4):

– верхній рівень складається з двох API структурного паралелізму даних: PLINQ та класу Parallel;
– нижній рівень містить класи для паралелізму задач - плюс додатковий набір конструкцій, що спрощують вирішення завдань паралельного програмування.

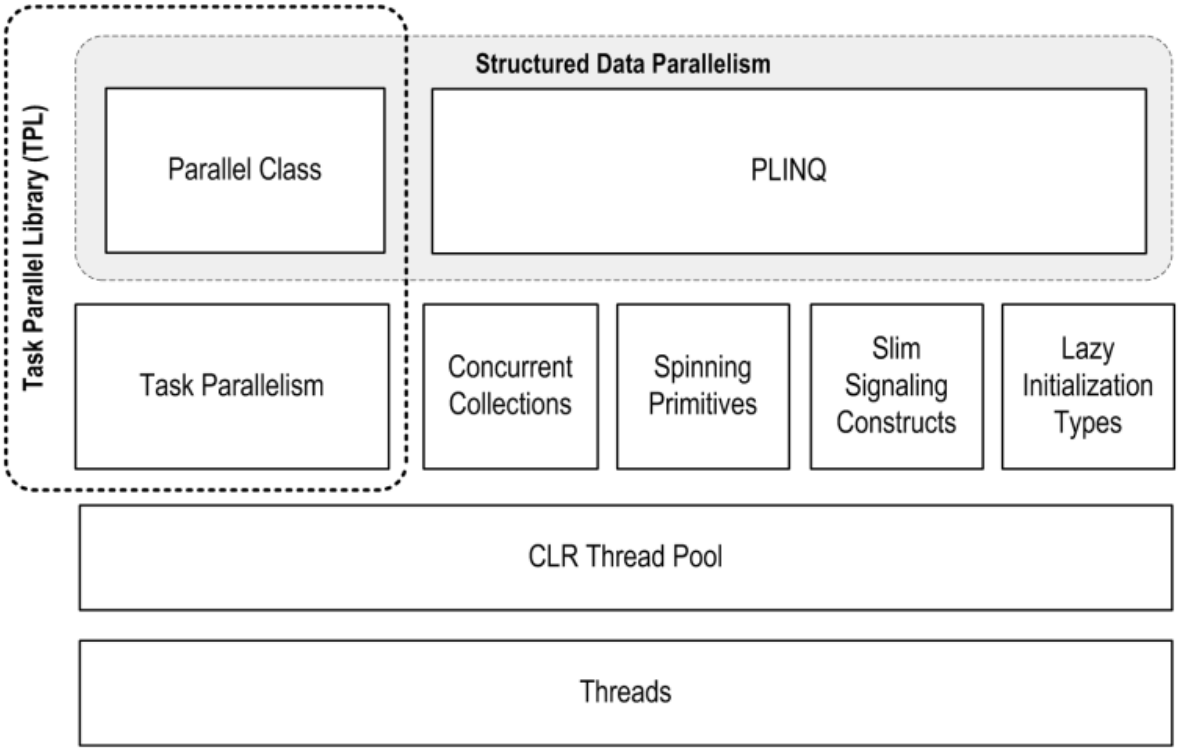



Рисунок 1.4 — Компоненти бібліотеки PFX

Бібліотека PLINQ надає найбільшу функціональність: вона автоматизує всі етапи розпаралелювання обчислень, включаючи поділ роботи на завдання, виконання цих завдань різними потоками та об'єднання результатів в одну вихідну послідовність. Її використання називають *декларативним* (declarative), оскільки ви просто оголошуєте (declare) те, що ви хочете

виконати паралельно (подібно до запитів LINQ), і не піклуєтесь про деталі реалізації.

На противагу цьому, інші підходи є *імперативними*: у цьому випадку необхідно явно написати код із поділу завдання та об'єднання результатів. У випадку класу Parallel, об'єднати результати потрібно самостійно; у разі конструкцій паралелізму завдань необхідно, до того ж, самостійно виділити підзадачі (табл.1.3).

Таблиця 1.3 — Ступінь автоматизації розпаралелювання обчислень при використанні різних інструментів

| Інструмент | Автоматичний поділ задачі | Автоматичне об'єднання результатів |
|---|---|---|
| PLINQ | Так | Так |
| Клас Parallel | Так | Ні |
| Паралелізм із PFX | Ні | Ні |

Зручним інструментом у середовищі .NET Framework є бібліотека розпаралелювання завдань (TPL - Task Parallel Library). Ця бібліотека вдосконалює багатопотокове програмування двома основними способами:

1. спрощує створення та застосування багатьох потоків;

2. дозволяє автоматично використовувати кілька процесорів.

Тобто TPL надає можливості для автоматичного масштабування додатків з метою ефективного використання всієї множини доступних процесорів.

Бібліотека TPL визначена у просторі імен System.Threading.Tasks (рис. 1.5).

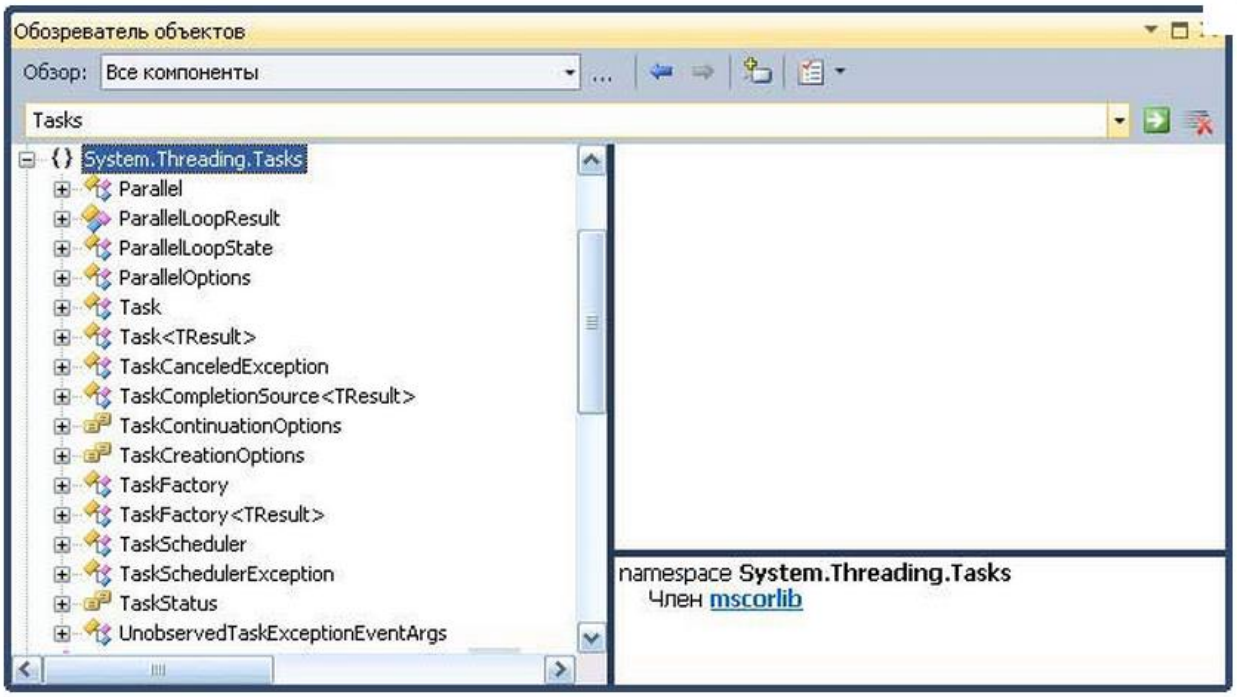



Рисунок 1.5 — Простір імен *System.Threading.Tasks*

Для роботи з бібліотекою TPL рекомендовано також підключати клас System.Threading, оскільки він підтримує синхронізацію та інші засоби багатопотокової обробки, у тому числі й ті, що входять до класу Interlocked.

Алгоритм створення багатопотокового додатку наступний:

1. Створюється метод, який буде точкою входу нового потоку.

2. Створюється новий делегат ParametrizedThreadStart (або ThreadStart), що передає конструктору метод, визначений на попередньому кроці.

3. Створюється об'єкт Thread для передачі в якості аргументу конструктора ParametrizedThreadStart/ThreadStart.

4. Встановлюються початкові характеристики потоку (ім'я, пріоритет тощо).

5. Викликається метод Thread.Start(). Ця дія запустить потік за методом, вказаному у делегаті, створеним на другому кроці, як тільки це буде можливо.

При використанні різних коротких завдань, що підлягають виконанню, можна заздалегідь створити набір - пул потоків - і потім просто надсилати відповідні запити, коли настає черга їх виконання.

Для управління списком потоків передбачений клас ThreadPool, який при необхідності зменшує та збільшує кількість потоків у пулі до максимально допустимого значення. Значення максимально допустимої кількості потоків у пулі може змінюватись. Наприклад, у випадку двоядерного ЦП воно за замовчуванням становить 1023 робочих потоків та 1000 потоків введення-виводу.

Переваги пулів потоків CLR:

- пул потоків управляє потоками оптимально, зменшуючи кількість створюваних, запущених та зупинених потоків;
- використовуючи пул потоків, можна зосередитись на вирішенні задачі, а не на управлінні потоками програми.

Таким чином, інструменти .NET в цілому та мови C# зокрема для організації багатопоточності та синхронізації в Windows досить потужні і дозволяють ефективно вирішувати завдання, що виникають у процесі розробки паралельних додатків.

## 1.3 Огляд наявних застосунків для навчальної системи з методів паралельного програмування

Паралельному програмуванню приділяється велика увага при підготовці фахівців рівня Middle та Senior.

Основними джерелами для навчання та закріплення навичок є різноманітні ресурси Інтернету.

Одним із базових курсів для вивчення особливостей реалізації потоків в C# є ресурс Джозефа Альбахарі (Joseph Albahari)[4].

[4] https://www.albahari.com/threading/part5.aspx

П’ята глава цього інтерактивного посібника повністю присвячена паралельному програмуванню (рис.1.6).

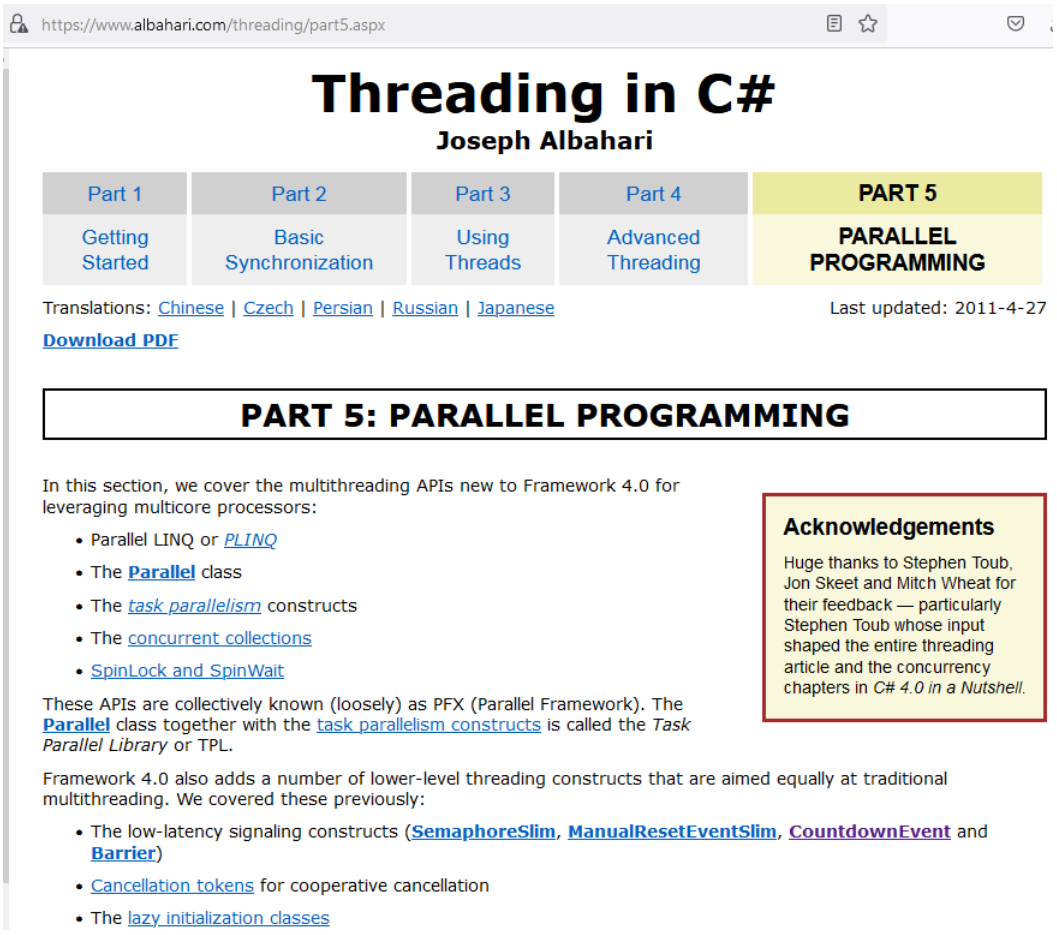



Рисунок 1.6 — Інтерактивний посібник Джозефа Альбахарі

На спеціалізованих навчальних платформах також представлені різноманітні курси з вивчення паралельного програмування, що базуються на різних мовах програмування.

Наприклад, курс «Parallel Programming» Віктора Кунчака на Coursera[5] є частиною спеціалізації програмування на Scala.

[5] https://www.coursera.org/learn/scala-parallel-programming

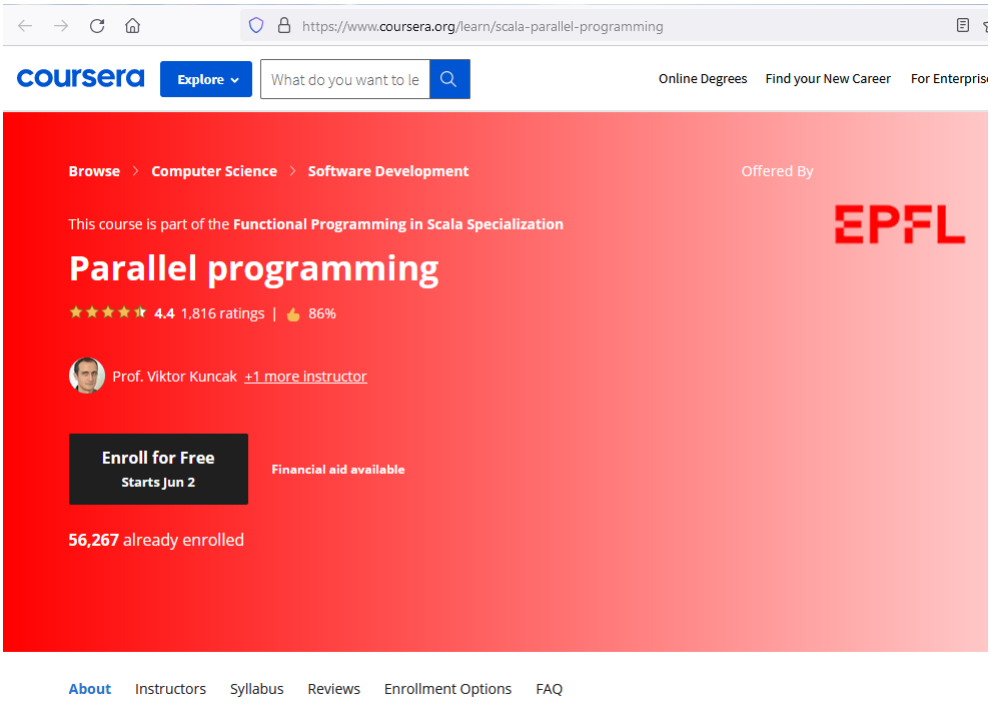



Рисунок 1.7 — Інтерактивний курс паралельного програмування на Scala

Крім цього курсу, на Coursera представлені ще більше 100 інших курсі з паралельного програмування (рис.1.8).

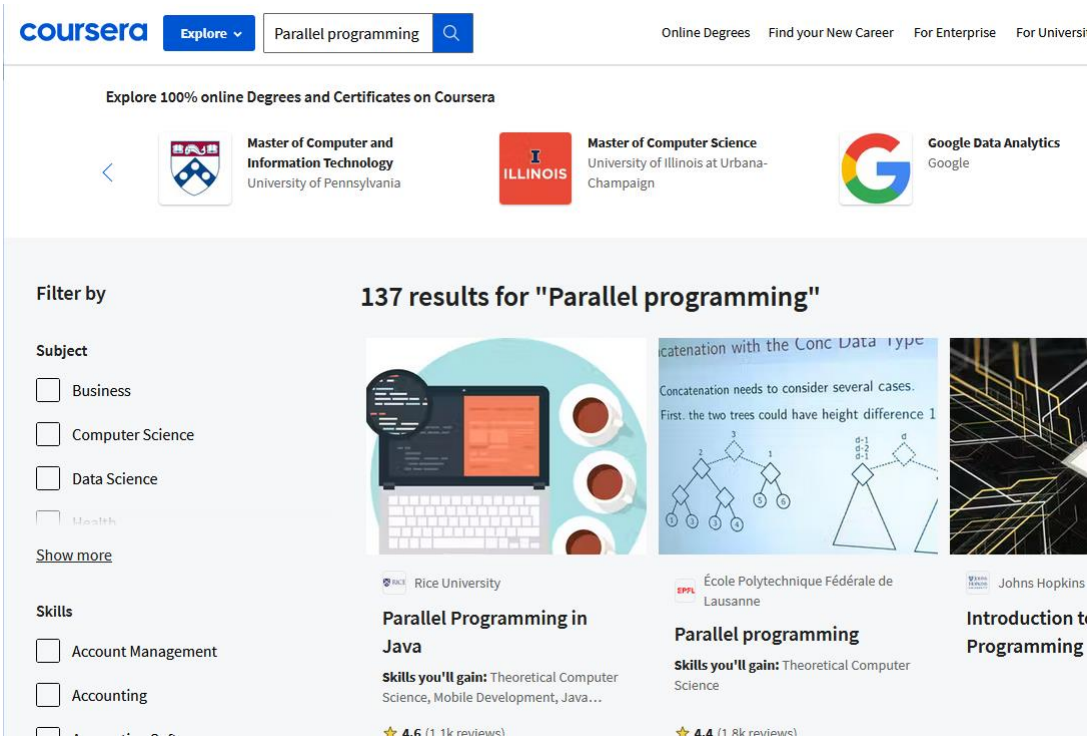



Рисунок 1.8 — Більше 100 різноманітних курсів з паралельного програмування на Coursera

На іншому популярному навчальному ресурсі edx.org за запитом "parallel programming" пропонується 11 різних курсів (рис.1.9).

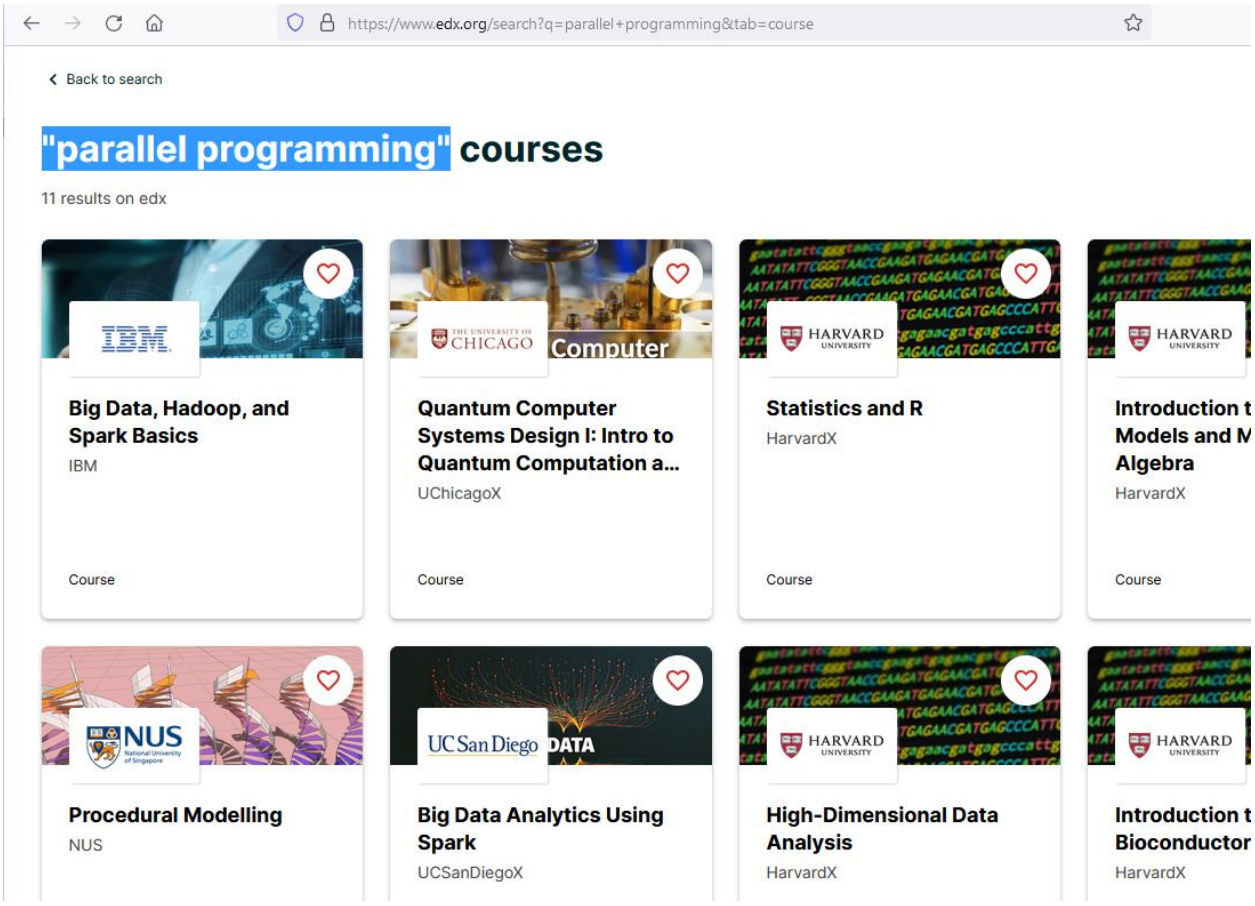



Рисунок 1.9 — 11 курсів з паралельного програмування на EDX

Курси на цих міжнародних платформах MOOC (Massive Open Online Courses) звичайно складаються із:

- інтерактивних посібників;
- відео з лекціями;
- задач для закріплення знань із кожного теоретичного розділу;
- вікна для розробленого коду, що перевіряється в режимі online;
- системи тестів та рейтингів, видачі сертифікатів.

Можливості перевірки коду та отримання рейтингів і сертифікатів, як правило, платні.

Вибрати задачі у вільному доступі для відпрацьовування навичок паралельного програмування можна на платформі LeetCode (рис.1.10).

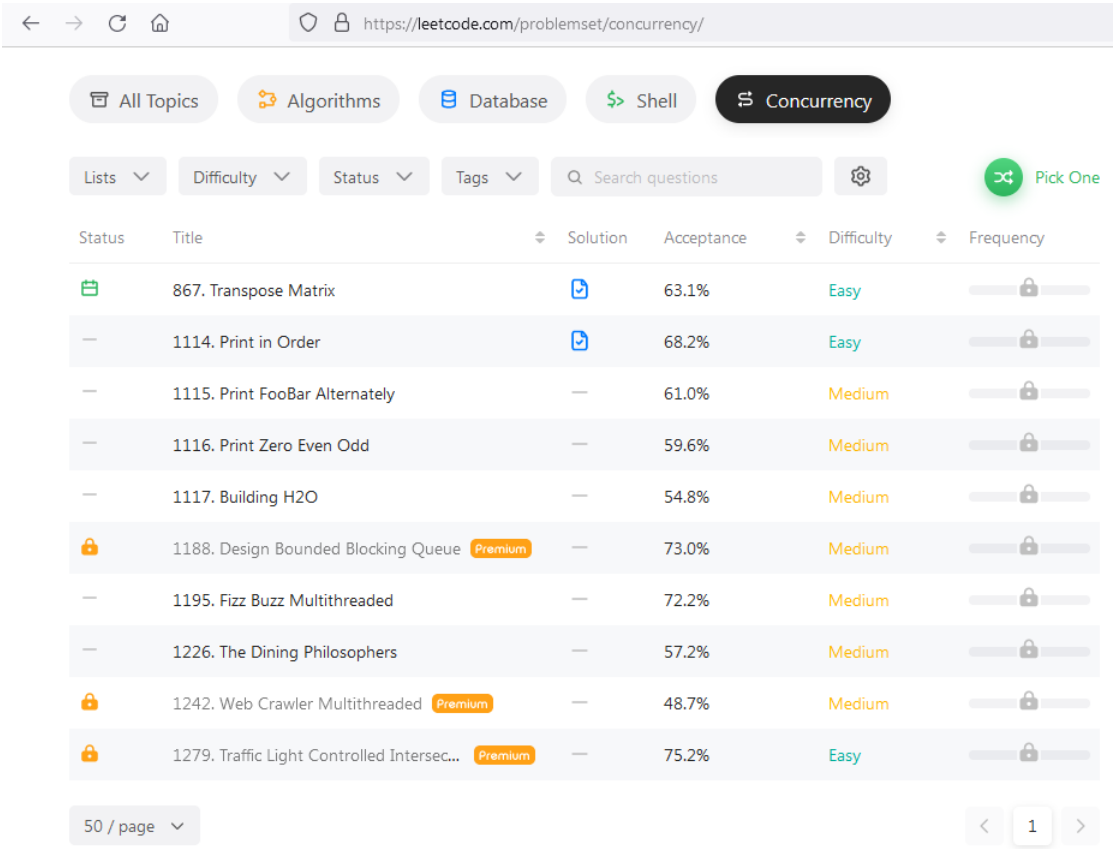



Рисунок 1.10 — Задачі на LeetCode, що пропонуються для паралельного програмування

Перевагою платформи LeetCode є надання вікна для коду та перевірки Online, а також унікальні бази даних для тестування, що дозволяє отримувати надійні програмні модулі не тільки для навчання, а для професійних задач (рис.1.11).

У вишах нашої країни розробляються і пропонуються різноманітні навчально-програмні комплекси. Це можуть бути як веб-додатки, так і десктоп-програми.

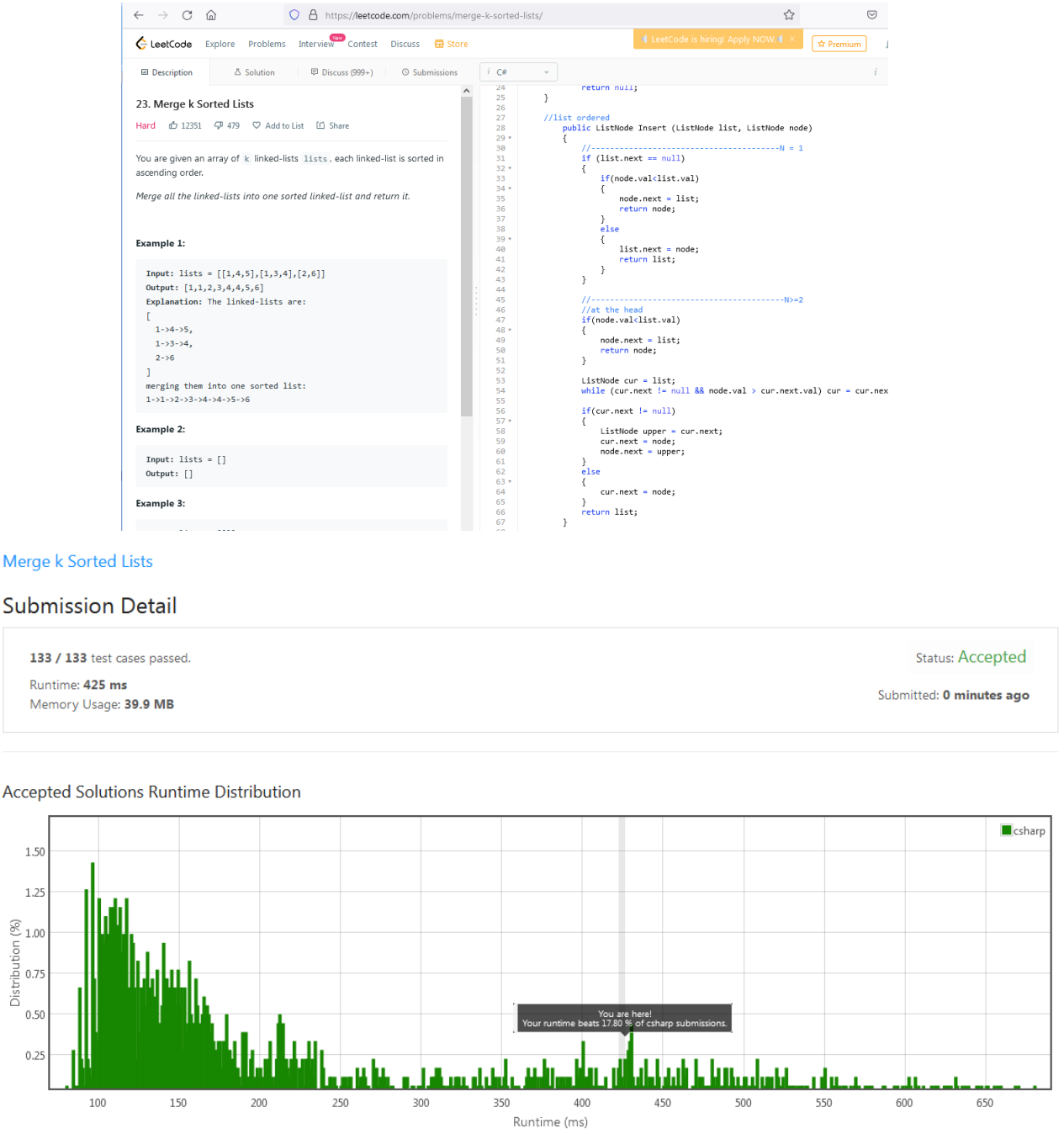



Рисунок 1.11 — Вікно для виконання додатку online (зверху) та результати з використанням бази тестів LeetCode (знизу)

В цій роботі нами також розроблено навчально-методичний комплекс для вивчення основ паралельного програмування та тестування знань студентів. Детальний опис програмного продукту та його можливості представлені в наступних розділах дослідження.

**Висновки до розділу 1**

При класифікації обчислювальних систем за М.Фліном основним визначальним архітектурним параметром є взаємодія потоку команд та потоку даних (операндів та результатів). Відповідно, виділяють наступні чотири класи :

- SISD (Single Instruction, Single Data)
- SIMD (Single Instruction, Multiple Data)
- MISD (Multiple Instructions, Single Data)
- MIMD (Multiple Instructions, Multiple Data) .

З розвитком багато поточних систем виникла необхідність більш детальної класифікації, яка була запропонована Джонсоном. Класифікація Джонсона заснована на структурі пам'яті (global - глобальна або distributed - розподілена) і механізмі комунікацій та синхронізації (shared variables - змінні, що розділяються, або message passing - передача повідомлень). Системи GMSV (global-memory-shared-variables) часто називаються також мультипроцесорами з пам'яттю (shared-memory multiprocessors). Системи DMMP (distributed-memory-message-passing) також називають мультикомп'ютерами з розподіленою пам'яттю (distributed-memory multicomputers).

Для визначення ефективних способів організації паралельних обчислень було запропоновано розбивати завдання на такі етапи: аналіз, дизайн, реалізація, налагодження, тестування та налаштування.

Збільшення ступеня ефективності паралелізму (зменшення тимчасових витрат на накладні витрати) досягається такими способами:

- укрупненням одиниць розпаралелювання;
- зменшенням складності алгоритмів генерації паралельних процедур (підпрограм);
- початковою підготовкою пакету різних варіантів вихідних даних;

– розпаралелювання алгоритмів генерації паралельних процедур (підпрограм).

Отже, оцінювати рівень ефективності при виконанні паралельного додатку можна за чотирма напрямами: виділення незалежних частин, визначення інформаційної взаємодії між частинами, масштабування завдань та їх розподілу між процесорами.

Windows – це багатопотокова операційна системою з витісняючою багатозадачністю, тому що кожен потік може бути зупинений у довільний момент часу і замість нього обраний для виконання інший.

Огляд інструментів .NET в цілому та мови C# зокрема показує, що для організації багатопоточності та синхронізації в Windows створені досить потужні бібліотеки, які дозволяють ефективно вирішувати завдання із розробки паралельних додатків.

Актуальність паралельних обчислень зумовила появу безлічі навчальних та інформаційних ресурсів у цій галузі. На наш погляд, використання переваг інтерактивних систем навчання суттєво підвищує ефективність процесу підготовки розробників, а також сприяє їх подальшій освіті та самоосвіті.

# РОЗДІЛ 2. ОПИС ПРОГРАМНОГО ЗАСТОСУНКУ

## 2.1. Опис технічного завдання

Комплекс програм для навчальної системи з методів паралельного програмування, що розробляється, має наступні призначення:

- навчального посібника;
- засобу для демонстрації прикладів паралельних обчислень;
- засобу для тестування студентів і перевірки засвоєння учбового матеріалу.

Користувачами програмного продукту є:

- викладач, котрий може розробляти, доповнювати та редагувати навчальний посібник, тести і відповіді до них;
- студент, котрий вивчає посібник, запускає на виконання демонстраційні приклади, виконує тестові завдання і отримує оцінку за результатами виконання.

Програмний продукт поширюється викладачем серед студентів, котрі проходять навчання за відповідною спеціальністю. Таким чином, продукт є безкоштовним, але його поширення обмежене.

Функціональне призначення – програмний продукт має забезпечити виконання наступних дій для користувачів:

- для викладача:
  - здійснювати вхід за логіном та паролем з правами викладача;
  - редагувати навчальний посібник;
  - додавати, редагувати і видаляти тести;
  - додавати, редагувати і видаляти відповіді до тестів.
- для студента:
  - здійснювати вхід за логіном та паролем з правами

студента;

- користуватись навчальним посібником із вбудованими гіперпосиланнями;
- запускати на виконання демонстраційні приклади, отримувати відповідні результати;
- проходити тести за пройденим матеріалом;
- отримувати результати тестування.

Експлуатаційне призначення – за допомогою програмного продукту виконується підготовка студентів 2-3 курсів, що вивчають тему «Паралельне програмування». Поєднання навчального посібника, демонстраційних прикладів та тестування в одному комплексі дозволяє покращити якість засвоєння досить складного матеріалу.

Додаток має складатися з трьох незалежних частин:

- теоретичні відомості;
- демонстраційна частина;
- тести (з двома режимами роботи - окремо для викладача і для студентів).

Вибір кожної з цих опцій у будь-якій послідовності має бути забезпечений за допомогою головного меню.

*Теоретичний матеріал*

Теоретичний матеріал є HTML-проектом, який містить сторінки з текстом, гіперпосиланнями та ілюстраціями для повного та наочного пояснення про можливості паралельних обчислень.

Теоретичний матеріал поділено на окремі сторінки: стартову та інші за розділами змісту. Користувачеві надано можливість переходу по сторінках трьома способами:

- натисканням на гіперпосилання з назвами "на попередню сторінку", "на наступну сторінку".

- натисканням на гіперпосилання під назвою «зміст» для повернення на стартову сторінку;
- зі стартової сторінки – на будь-яку обрану за змістом.

Важливою перевагою такої організації теоретичного матеріалу є можливість його самостійного поповнення як викладачем, так і студентами – для цього достатньо початкових знань мови HTML-розмітки.

*Демонстраційна частина*

Багато відомих задач можна використати в якості демонстрації можливостей розпаралелювання. На наш погляд, всі ці завдання можна поділити на три категорії:

- елементарні задачі для ознайомлення з азами паралельного програмування: множення матриць та векторів, пошук простих чисел, обчислення числа π;
- задачі середньої складності, для розуміння алгоритмів яких необхідні знання спеціальних розділів математики: рішення СЛАР, диференціальних рівнянь, інтегрування, реалізація клітинних автоматів, геометричні перетворення в просторі;
- складні задачі, вирішення яких потребує комплексного підходу та суттєвих витрат часу: завдання на графах, моделювання фізичних та біологічних об'єктів, обробка зображень та звуку.

Враховуючи основне призначення даного продукту – стати зручним підручником для вивчення основ паралельних обчислень, - для демонстрації пропонується задача елементарного рівня – пошук простих чисел (реалізувавши два алгоритми: решето Ератосфена та решето Аткіна), та завдання середнього рівня – клітинний автомат «Життя» Конвея.

Для кожного з цих завдань представлено кілька способів розв'язання: «звичайні» послідовні обчислення, розпаралелювання за допомогою інструментів бібліотеки TPL та за допомогою PLINQ. Здійснюються виміри часу виконання залежно від максимальної кількості при пошуку простих

чисел або від розмірності ігрового поля в клітинному автоматі. Тим самим студент отримує можливість провести емпіричні дослідження ефективності представлених рішень для різних вхідних даних та різних конфігурацій ПК та зробити відповідні висновки.

*Тест на закріплення матеріалу*

Типи питань: тестувальна частина складається із закритих питань з однією правильною відповіддю.

Формат подання тестових питань: Питання та відповіді зберігатимуться у спеціальному текстовому файлі. Файл з базою відповідей має бути недоступним для перегляду студентом поза програмою (зашифрований).

Викладачеві необхідно надати можливість роботи з файлами питань у незашифрованому вигляді: створювати, редагувати, видаляти файли, зашифровувати та розшифровувати тести.

Алгоритм представлення тестів студенту для вирішення

- з файлу завантажується 10 питань. Відповідно, для того, щоб забезпечити різноманітність при наповненні тестів, викладачеві рекомендується готувати 20-25 питань у кожному тесті.
- питання вибираються у випадковому порядку та виводяться на екран.

Перед початком тестування всі питання будуть зчитуватись до списку об'єктів, а потім випадкові об'єкти з цього списку будуть передані до іншого списку. Вибір випадкових об'єктів буде здійснюватися за допомогою спеціальної функції, яка забезпечує відсутність повторюваних елементів у новому списку.

Таким чином розділ тестів також може постійно поповнюватися новими файлами, а вже підготовлені тести можуть змінюватися і вдосконалюватися.

*Вимоги до вхідних даних:*

1. HTML – проект з теорією.

2. Дані, отримані від користувача під час роботи програми (у демонстраційній частині: максимальне число для пошуку простих чисел, розмір

ігрового поля для гри «Життя»; у тестовій частині: ім'я, пароль викладача, ім'я, група, що тестується, варіанти відповідей).

Демонстраційний розділ: параметри вводяться користувачем із клавіатури.

Розділ тестування:

1. Ім'я та пароль викладача, ім'я та група студента вводяться з клавіатури та обробляються програмою як рядки.

2. Вихідний текст файлу тестів запроваджується викладачем і запам'ятовується як файл *.rtf.

3. Варіанти відповіді, які вибирає користувач, встановлюються за допомогою кліків миші і зберігаються в одному з класів програми протягом тестування.

*Вимоги до вихідних даних:*

Зашифрований файл із тестами, який обробляється в одному з вікон програми та запам'ятовується як файл *.rtf.

Текстовий файл statistics.txt з результатами тестування студента у форматі: ім'я, група, відсоток правильних відповідей.

## 2.2 Вимоги до технічного забезпечення

*Вимоги до надійності*

Вимоги до надійності наступні:

- наявність архівної копії тексту програми на зовнішньому носії;
- наявність резервної копії бази даних на зовнішньому носії.

*Вимоги експлуатації*

Програмний продукт повинен використовуватись у приміщеннях які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.13-99 (див. табл. 1).

Таблиця 2.1 — Кліматичні умови

| Пора року | Категорія робіт згідно з ГОСТ 12.01-005-88 | Температура повітря, град.С | Відносна вологість повітря, % | Швидкість руху повітря, м/с |
|---|---|---|---|---|
| | | Оптимальна | Оптимальна | Оптимальна |
| Холодна | легка-1-а | 22-24 | 40-60 | 0,1 |
| | легка-1-б | 21-23 | 40-60 | 0,1 |
| Тепла | легка-1-а | 23-25 | 40-60 | 0,1 |
| | легка-1-б | 22-24 | 40-60 | 0,2 |

Працювати з програмою може людина, що має навички роботи з комп’ютером та ознайомлена з керівництвом користувача програмного продукту.

*Вимоги до складу та параметрів технічних засобів*

Продукт, що розробляється повинен використовуватись на комп’ютерах, що мають наступні мінімальні характеристики:

- діагональ екрану – 17 дюймів;
- роздільна здатність дисплею – HD (1280x720);
- оперативна пам’ять – 6 ГБ;
- вбудована пам’ять – 16 ГБ;
- операційна система – WINDOWS 7;
- частота процесора – 1.6 ГГц;
- 2-ядерний процесор;
- наявність відеокарти;
- USB-порт для завантаження/використання додатку з носія.

*Вимоги до інформаційної та програмної сумісності*

Програмний продукт розробляється для всіх видів операційних систем сімейства “WINDOWS” починаючи від версії 7 та наступні версії.

*Вимоги до маркування і упаковки*

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

На упаковці повинно бути вказана назва продукту, номер версії, мінімальні системні вимоги.

На зворотній стороні упаковки вказується розробник та його юридична адреса.

*Вимоги до транспортування та зберігання*

Транспортування повинне забезпечувати збереження програмного продукту його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на фізичному носії та використовується на ПК через USB-порт.

*Інсталяція та виконання*

Для роботи з програмою, необхідно скопіювати папку ParallelOptimization на жорсткий диск або будь-який інший носій. Щоб розпочати роботу, запускаємо файл ParallelOptimization.exe, який знаходиться у кореневій папці програми (рис.2.1).


Рисунок 2.1 — Структура кореневої папки додатку

Опис файлів подано у табл.2.2.

Таблиця 2.2

Файли і папки дистрибутиву

| Ім’я | Призначення |
|---|---|
| ParallelOptimization.exe | Виконуваний файл програми |
| theory | Папка з html-проектом для ознайомлення з теорією з цього питання. Обов'язково повинна знаходитися в робочій папці, з exe-файлом. |
| HashTables.vshost.exe<br>HashTables.vshost.exe.manifest | Необхідні системні файли (від IDE) |
| tutor | Папка з матеріалами викладача: зразком незашифрованого файлу тесту |
| student | Папка з матеріалами студента: зразком зашифрованого файлу тесту |
| statistics.txt | Текстовий файл зі статистикою виконаних тестів |

Для коректної роботи програми необхідні файли перших трьох позицій.

Текстові файли мають демонстраційний характер і необов'язкові – користувач може створити власні тести, розташувати з довільним чином та отримати власну статистику.

## 2.3 Опис архітектури програмного забезпечення

Склад проекту *ParallelOptimization* представлено на рис.2.2.

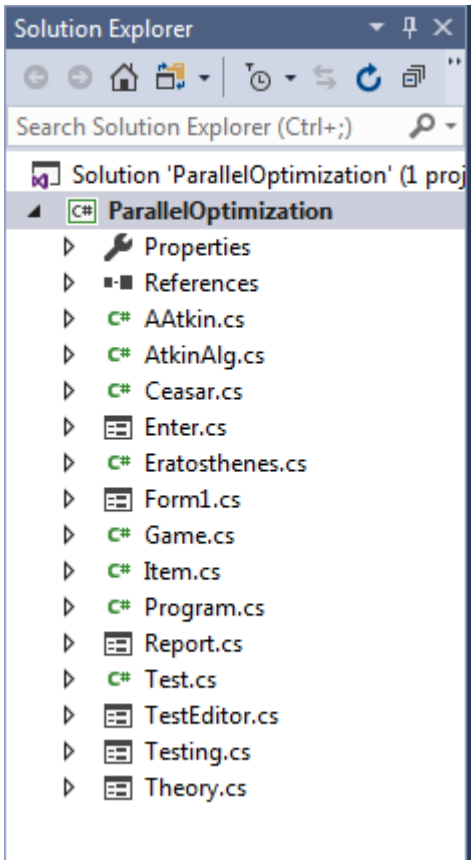



Рисунок 2.2 — Склад проекту *ParallelOptimization*

Проект містить

— 1 клас, що є точкою входу для IDE (Program.cs);

— 5 класів користувача, що є формами і наслідують Windows.Form;

— 7 класів типів користувача.

Діаграма залежності класів, розроблених у додатку, представлено на рис.2.3.

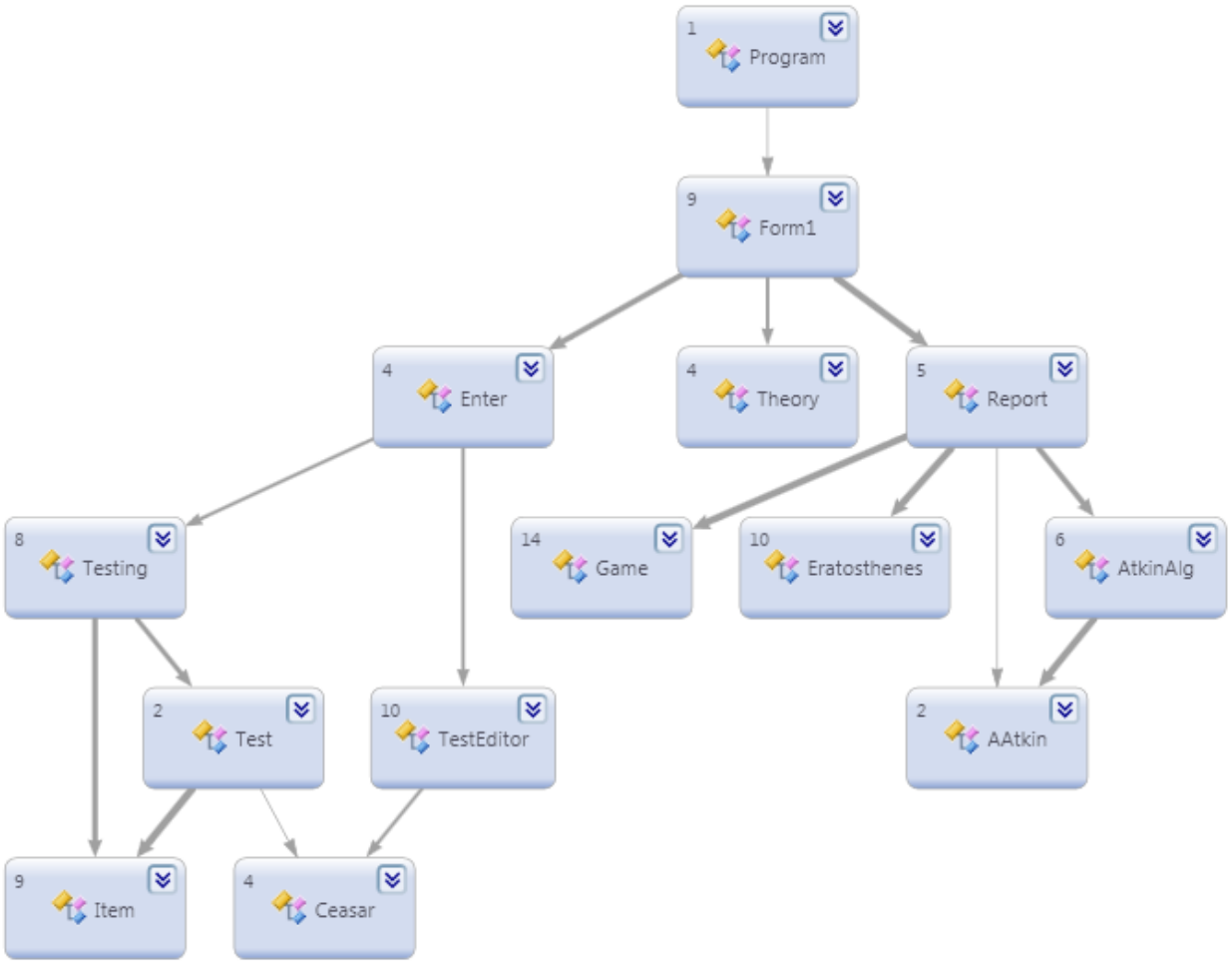



Рисунок 2.3 — Діаграма залежності класів додатку

Клас Program містить клас Main – точку входу до програми.

Клас Form1 – головна форма програми з опціями меню (рис.2.4).

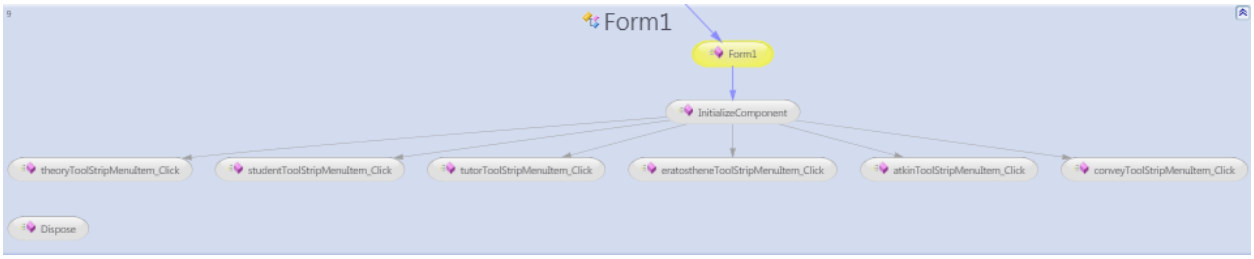



Рисунок 2.4 — Події класу Form1 – вибір опцій меню

Вибираючи опцію головного меню «Теоретичні відомості» відкривається дочірня форма, описана класом Theory (рис.2.5). Активізується вікно WebBrowser та завантажується стартова сторінка HTML-проекту з теоретичними відомостями

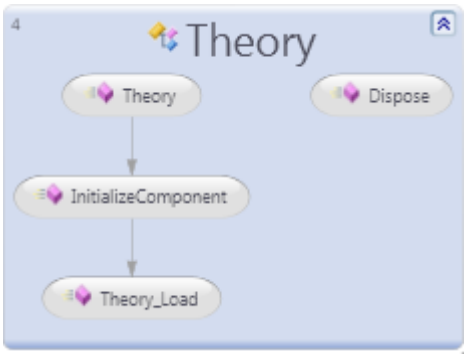



Рисунок 2.5 — Метод (подія) класу Theory.

При виборі опції головного меню "Демонстрація" відкривається дочірня форма, описана класом Report (рис.2.6).

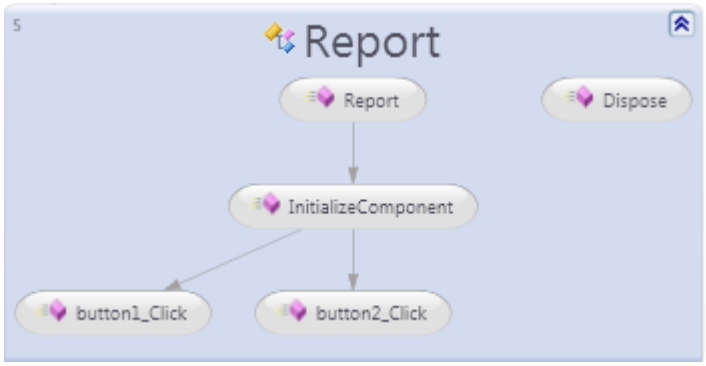



Рисунок 2.6 — Дочірня форма Report – події натискання кнопок введення даних та очищення форми

Методи розв’язку вибраних завдань розроблені у класах Eratosthenes, AAtkin, AtkinAlg, Game. Ці класи докладно розглянуті у Розділі 3 цієї роботи.

При виборі опції головного меню "Тест" → "Студент" або "Тест" → "Викладач" відкривається дочірня форма, описана класом Entrance (рис.2.7).

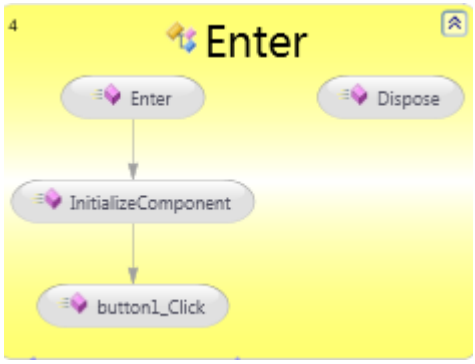



Рисунок 2.7 — Клас форми входу в режим редагування тестів або в режим тестування

При виборі опції головного меню "Тест" → "Викладач" після введення імені та пароля відривається вікно для створення, редагування та шифрування файлів з тестами (рис.2.8).

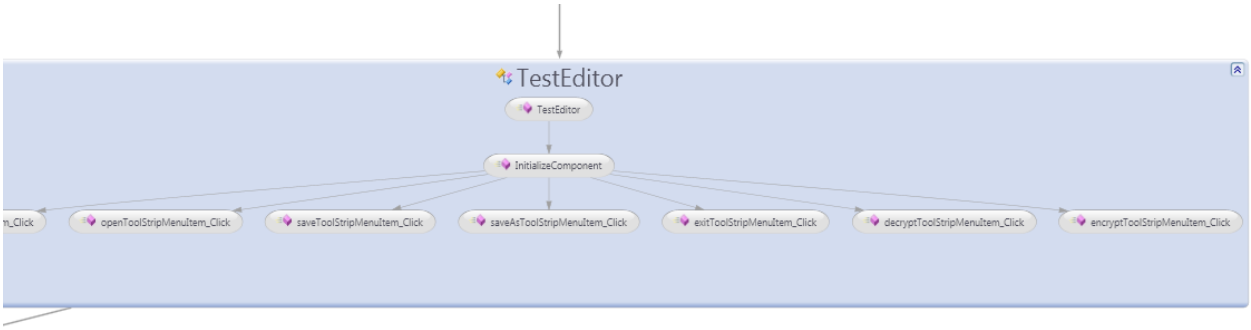



Рисунок 2.8 — Клас TestEditor – форма для створення, редагування та шифрування файлів із тестами

При виборі опції головного меню "Тест" → "Студент" після введення імені та групи відривається вікно для проходження тесту (рис.2.9).

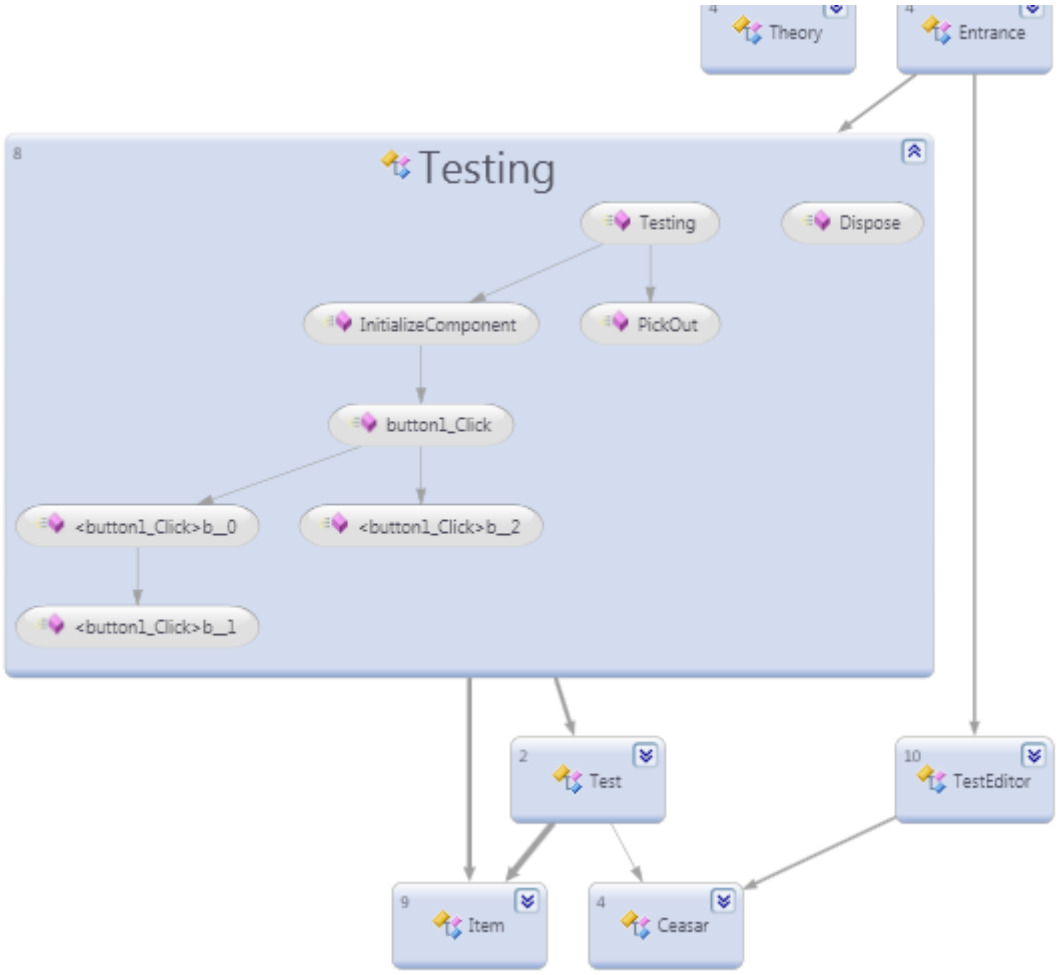



Рисунок 2.9 — Клас Testing – форма для проходження тесту

Зміст тесту завантажується із зашифрованого файлк. Відповідні методи містяться у класі Test (рис.2.10).

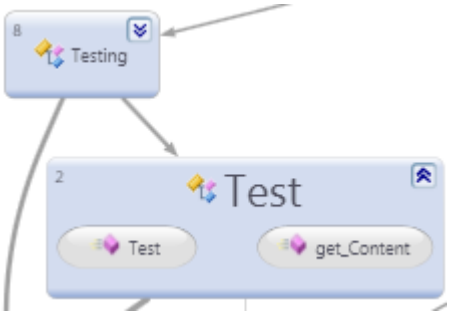



Рисунок 2.10 — Клас Test – завантаження тесту із зашифрованого файлу

Тест складається з об'єктів класу Item – питань та пов'язаної із ними інформації (кількості відповідей, номеру правильної відповіді, варіантів відповідей) – рис.2.11.

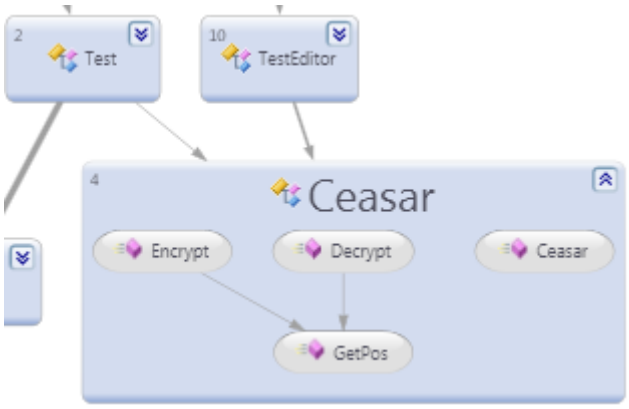



Рисунок 2.11 — Клас Ceasar – методи шифрування та дешифрування за шифром Цезаря

Всі представлені класи з певними змінними та властивостями разом з розробленими методами дають можливість реалізувати функціональні можливості навчального комплексу згідно із технічним завданням.

**Висновки до розділу 2**

Комплекс програм для навчальної системи з методів паралельного програмування, що розробляється, має наступні призначення:

- навчального посібника;
- засобу для демонстрації прикладів паралельних обчислень;
- засобу для тестування студентів і перевірки засвоєння учбового матеріалу.

Користувачами програмного продукту є:

- викладач, котрий може розробляти, доповнювати та редагувати навчальний посібник, тести і відповіді до них;
- студент, котрий вивчає посібник, запускає на виконання демонстраційні приклади, виконує тестові завдання і отримує оцінку за результатами виконання.

Для роботи з програмою користувачу необхідно скопіювати папку дистрибутиву ParallelOptimization на жорсткий диск або будь-який інший носій. Програма працює в ОС Windows, починаючи від 7 та вище.

Із вимог до ПК, обов’язковим для дослідження паралельних обчислень є наявність мінімум 2 ядер у процесорі та відеокарти.

Проект *Parallel Optimization* містить

— 1 клас, що є точкою входу для IDE (Program.cs);

— 5 класів користувача, що є формами і наслідують Windows.Form;

— 7 класів типів користувача.

Всі представлені класи з певними змінними та властивостями разом з розробленими методами дають можливість реалізувати функціональні можливості навчального комплексу згідно із технічним завданням.

# РОЗДІЛ 3. ЕТАПИ РОЗРОБКИ І СТВОРЕННЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ НАВЧАЛЬНОЇ СИСТЕМИ З МЕТОДІВ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ

## 3.1 Опис алгоритму

*Решето Ератосфена*

Для пошуку простих чисел на відрізку від 1 до $N$ найпростіший алгоритм було запропоновано ще давньогрецьким математиком Ератосфеном Кіренським. Метод отримав назву «Решето Ератосфена», оскільки, згідно з легендою, Ератосфен писав числа на дощечці, покритій воском, і послідовно виколював складені числа.

Алгоритм складається з наступних кроків:

1. Виписати підряд усі цілі числа від 2 до $N$;
2. Нехай змінна $p$ спочатку дорівнює першому простому числу, $p = 2$;
3. Закреслити у списку числа від $2p$ до $N$, рахуючи кроками по $p$;
4. Знайти перше незакреслене число у списку, більше ніж $p$, та присвоїти змінній $p$ це значення;
5. Повторювати кроки 3 і 4, поки це можливо.

Алгоритм можна оптимізувати, якщо на кроці 3 закреслювати числа, починаючи з $p^2$, оскільки всі складені числа менше за нього вже будуть закреслені. Відповідно, алгоритм зупиняємо, якщо $p^2 > N$. Також, всі $p$ більші ніж 2 - непарні числа, тому для них можна переміщатися кроками по $2p$, починаючи з $p^2$.

Нехай A - булевий масив, що індексується числами від 2 до $N$ і спочатку заповнений значеннями true.

Тоді псевдокод для оптимізованого алгоритму, що починається з $p^2$ має вигляд

```
For i:= 2,3,4,…, until i2<= N:                    (3.1)
     If A[i] = true:
          For j:= i2, i2+I, i2+2i,…, until j<=N:
               A[j] := false
```

Простими числами будуть усі індекси i, у котрих A[i] = true.

Складність алгоритму становить $O(Nlog(logN))$ операцій.

Розглянемо особливості паралельного алгоритму для решета Ератосфена.

Пошук розбивається на 2 етапи:

1 етап. Знайти всі прості числа в діапазоні від 2 до $\sqrt{N}$ послідовними обчисленнями, використовуючи просте або модифіковане "решето".

2 етап. Викреслити всі складені числа в діапазоні $\sqrt{N} \dots N$ в паралельних потоках. Паралелізація полягає в тому, що діапазон $\sqrt{N} \dots N$ розбивається на піддіапазони розміру $m$, де

$$m = round.up.to.even(\lfloor \sqrt{N} \rfloor) \quad (3.2)$$

Очевидно, що на окремих піддіапазонах пошук простих чисел може відбуватися незалежно.

Програмна реалізація алгоритму представлена у Додатку А.

*Решето Аткіна*

Основна ідея алгоритму полягає у використанні неприводимих квадратичних форм (подання чисел у вигляді $ax^2 + by^2$). До цього, пропоновані алгоритми в основному були різними модифікаціями решета Ератосфена, де використовувалося представлення чисел у вигляді редукованих форм (як правило - у вигляді добутку $xy$).

У спрощеному вигляді алгоритм Аткіна може бути представлений таким чином:

- Усі числа, рівні (за модулем 60) 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40 , 42, 44, 46, 48, 50, 52, 54, 56 або 58, діляться на два і точно не прості. Усі числа, рівні (за модулем 60) 3, 9, 15, 21, 27, 33, 39, 45, 51 або 57, діляться на три і теж не є простими. Всі числа, рівні (за модулем 60) 5, 25, 35 або 55, кратні 5 і не прості. Усі ці залишки (за модулем 60) ігноруються.
- Усі числа, рівні (за модулем 60) 1, 13, 17, 29, 37, 41, 49 або 53, мають залишок від ділення на 4, рівний 1. Ці числа є простими тоді і лише тоді, коли кількість розв'язків рівняння $4x^2 + y^2 = n$ непарна і саме число не кратне жодному квадрату простого числа.
- Числа, рівні (за модулем 60) 7, 19, 31, або 43, мають залишок від ділення на 6, рівний 1. Ці числа є простими тоді і лише тоді, коли кількість розв'язків рівняння $3x^2 + y^2 = n$ непарна і саме число не кратне жодному квадрату простого.
- Числа, рівні (за модулем 60) 11, 23, 47, або 59, мають залишок від ділення на 12, рівний 11. Ці числа є простими тоді і тільки тоді, коли кількість розв'язків рівняння $3x^2 - y^2 = n$ (для $x > y$ ) непарна і саме число n не кратне жодному квадрату простого.
- Окремий крок алгоритму викреслює числа, кратні квадратам простих чисел. Оскільки жодне з цих чисел не ділиться на 2, 3, або 5, то, відповідно, вони не діляться і на їх квадрати. Тому перевірка, що число не кратне квадрату простого числа, не включає $2^2$, $3^2$ і $5^2$.

В алгоритмі використовується два види оптимізації, які суттєво підвищують його ефективність. Алгоритм має асимптотичну складність $O\left(\frac{N}{log(logN)}\right)$ і вимагає $O\left(N^{\frac{1}{2}+o(1)}\right)$ біт пам'яті. У порівнянні з іншими алгоритмами, решето Аткіна не є швидшим, але потребує значно менше пам'яті.

Організація паралельних обчислень з використанням решета Аткіна також включає два етапи, представлені вище для решета Ератосфена.

Програмна реалізація алгоритму представлена у Додатку Б.

*«Життя» Конвея*

Гра "Життя" - клітинний автомат, запропонований математиком Джоном Конвеєм у 1970 році.

Місце дії цієї гри – розмічена на клітини поверхня чи площина – безмежна, обмежена, або замкнута. Кожна клітина на цій поверхні може бути в двох станах: «живою» або «мертвою» (порожньою). Клітина має вісім сусідів (навколишніх клітин).

Розподіл живих клітин на початку гри називається першим поколінням. Кожне наступне покоління розраховується на основі попереднього за такими правилами:

— у порожній (мертвій) клітині, поряд з якою рівно три живі клітини, зароджується життя;
— якщо жива клітина має дві або три живі сусідки, то ця клітина продовжує жити; в іншому випадку (якщо сусідів менше двох або більше трьох) клітина помирає (від самотності або від перенаселеності);
— гра припиняється, якщо на полі не залишиться жодної «живої» клітини, якщо при черговому кроці жодна з клітин не змінює свого стану (складається стабільна конфігурація) або якщо конфігурація на черговому кроці в точності (без зрушень та поворотів) повторить себе саму на одному з ранніх кроків (складається періодична конфігурація).

Ці прості правила призводять до величезної різноманітності форм, які можуть виникнути у грі.

У комп'ютерних реалізаціях гри поле обмежено і, як правило, є емуляцією поверхні тора, але на екрані поле завжди відображається у вигляді рівномірної сітки.

Найпростіший алгоритм «зміни покоління» послідовно переглядає всі осередки ґрат і для кожного осередку підраховує сусідів, визначаючи долю кожної клітини (не зміниться, помре, народиться). Такий найпростіший алгоритм використовує два двовимірні масиви – один для поточного покоління, другий – для наступного.

Більш складний, але й швидший алгоритм складає списки клітин перегляду у наступному поколінні; клітини, які можуть змінитися, до списків не вносяться. Наприклад, якщо якась клітина та жодна з її сусідів не змінилися на попередньому ході, то ця клітина не зміниться і на поточному ході.

Оскільки обчислення нових станів кожної клітини є незалежними, вони можуть виконуватися паралельно. Також можна розпаралелити зовнішній цикл за рядками або стовпцями.

Програмна реалізація алгоритму представлена у Додатку В.

## 3.2 Розробка власного продукту

### 3.2.1 Розробка структури даних

Розглянемо структуру класів, що виконують паралельні обчислення у додатку.

Клас *Eratosthenes*, що виконує пошук простих чисел за Решетом Ератосфена містить 5 методів (рис. 3.1).

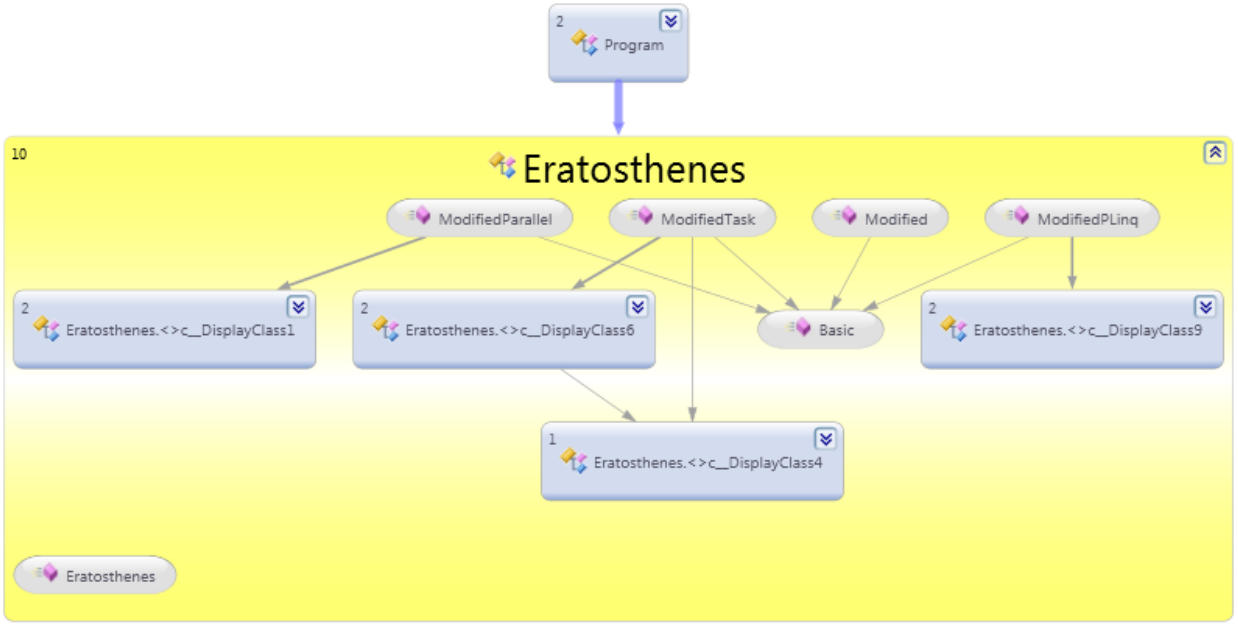



Рисунок 3.1 — Структура і методи класу *Eratosthenes*

– *Basic* - класичний алгоритм, послідовні обчислення;
– *Modified* - оптимізований алгоритм, послідовні обчислення;
– *ModifiedParallel* – оптимізований алгоритм, паралельні обчислення з використанням низькорівневої синхронізації (SpinLock). Структура SpinLock є низькорівневим взаємовиключним примітивом синхронізації, який виконує цикл в очікуванні отримання блокування. На багатоядерних комп'ютерах у разі, коли при невеликому завантаженні процесора передбачається, що період очікування буде недовгим, SpinLock може виявитися значно ефективнішим у порівнянні з іншими різновидами блокувань;
– *ModifiedTask* – оптимізований алгоритм, паралельні обчислення з використанням бібліотеки TPL;
– *ModifiedPlinq* – оптимізований алгоритм, паралельні обчислення з використанням PLINQ.

Клас *AtkinAlg*, що виконує пошук простих чисел за методом Аткіна, містить 4 методи (рис.3.2).

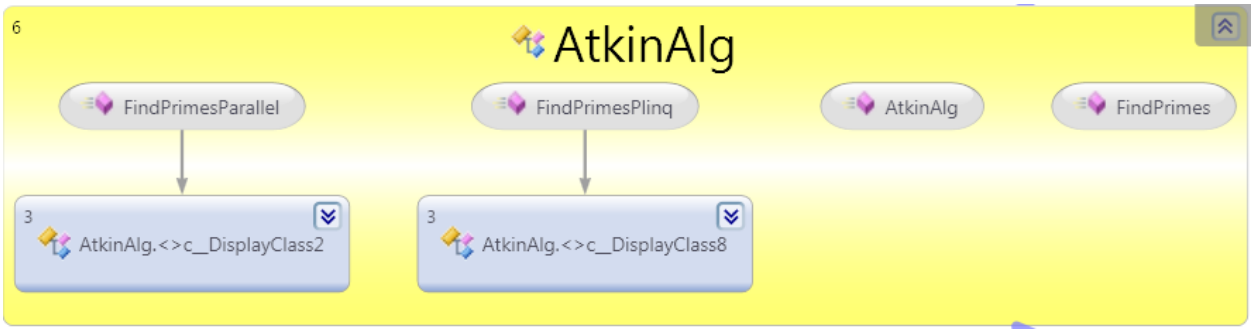



Рисунок 3.2 — Структура і методи класу *AtkinAlg*

— *AtkinAlg* - конструктор, успадкований від абстрактного класу AAtkin;

— *FindPrimes* - послідовний пошук простих чисел;

— *FindPrimesParallel* – паралельний пошук простих чисел із використанням бібліотеки TPL;

— *FindPrimesPlinq* – паралельний пошук простих чисел з використанням PLINQ.

Клітковий автомат для гри «Життя» Конвея реалізований в класі *Game* (рис.3.3).

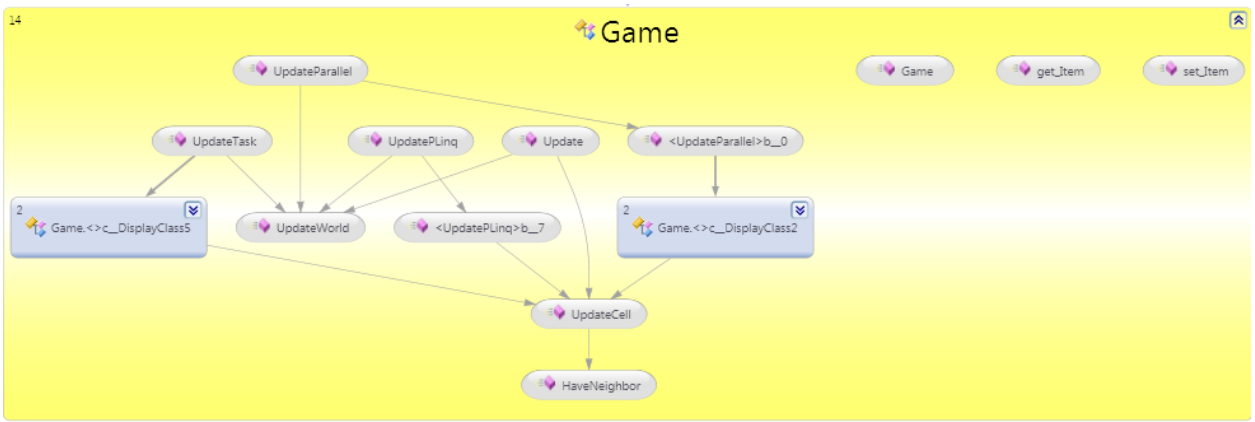



Рисунок 3.3 — Структура і Методи класу *Game*

— *UpdateCell* – метод для обчислення змін у кожній комірці

— *Update* – послідовне виконання переходу на наступний крок
— *UpdateParallel* – параллельна обробка для кожної комірки з використанням *Tasks.Parallel*
— *UpdatePlinq* – параллельна обробка для кожної комірки з використанням PLINQ

*Структура даних для обробки тестів*

Типи питань: тестувальна частина складається із закритих питань з однією правильною відповіддю.

Формат подання тестових питань: Питання та відповіді зберігатимуться у спеціальному текстовому файлі. Файл з базою має бути недоступним для перегляду студентом поза програмою (зашифрований).

Викладачеві необхідно надати можливість роботи з файлами питань у незашифрованому вигляді: створювати, редагувати, видаляти файли, зашифровувати та розшифровувати тести.

Структура питань, що будуть записані у незашифрованому файлі представлена на рис.3.4:

– перший рядок починається з символу # - питання тесту;
– другий рядок – кількість варіантів відповіді
– третій рядок – номер правильної відповіді
– інші рядки - варіанти відповідей.

Між питаннями може бути будь-яка кількість порожніх рядків.

Алгоритм представлення тесту користувачу (Студенту) є наступним:

– з файлу випадковим чином обирається 10 питань. Відповідно, для того, щоб забезпечити різноманітність при наповненні тестів, викладачеві рекомендується готувати 20-25 питань у кожному тесті.
– питання вибираються у випадковому порядку та виводяться на екран. Перед початком тестування всі питання будуть зчитуватись до списку об'єктів, а потім випадкові об'єкти з цього списку будуть передані до

іншого списку. Вибір об'єктів буде здійснюватися за допомогою спеціальної функції, яка забезпечує відсутність повторюваних елементів у новому списку.

```
#Укажите неправильное утверждение
4
2
SISD - это обычные последовательные компьютеры
SIMD - большинство современных ЭВМ относятся к этой категории
MISD - вычислительных машин такого класса мало
MIMD -это реализация нескольких потоков команд и  потоков данных

#Для конвейерной обработки присуще
4
3
загрузка операндов в векторные регистры
операции с матрицами
выделение отдельных этапов выполнения общей операции
сложение 2-х операндов одновременным сложением всех их двоичных разрядов

#Стек - это...
4
3
"память", в адресном пространстве которой работает процесс
тот или иной способ передачи инструкции из одного процесса в другой
область памяти для локальных переменных, аргументов и возвращаемых
функциями значений
организация доступа 2х (или более) процессов к одному и тому же блоку
памяти
```

Рисунок 3.4 — Фрагмент незашифрованого файлу (у папці викладача) з питаннями тесту

Таким чином розділ тестів також може постійно поповнюватися новими файлами, а вже підготовлені тести можуть змінюватися і вдосконалюватися.

### 3.2.2 Написання програмного продукту

В цьому розділі розглянемо особливості програмної реалізації паралельних обчислень при вирішенні поставлених задач.

*Решето Ератосфена*

Код методів, що реалізують пошук простих чисел за допомогою решета Ератосфена, подано у Додатку А.

Метод *Basic* реалізує класичний алгоритм з послідовним пошуком

```
public List<long> Basic(long n)
        {
            var basic = new List<long>();

            for (var i = 2; i <= n; ++i)
            {
                var prime = true;
                for (var j = 0; j < basic.Count && prime; ++j)
                    prime = i % basic.ElementAt(j) != 0;

                if (prime)
                    basic.Add(i);
            }
            return basic;
        }
```

Метод *Modified* обчислює послідовно, за оптимізованим алгоритмом

```
public List<long> Modified(long n)
        {
            var sqrt = (int)Math.Sqrt(n);
            var basic = Basic(sqrt);

            var primes = new List<long>(basic);

            for (int i = sqrt + 1; i <= n; ++i)
            {
                var prime = true;
                for (int j = 0; j < basic.Count() && prime; ++j)
                    prime = i % basic.ElementAt(j) != 0;

                if (prime)
                    primes.Add(i);
            }

            return primes;
        }
```

Метод *ModifiedParallel* – це оптимізований алгоритм, паралельні обчислення з використанням низькорівневої синхронізації (SpinLock)

```
public List<long> ModifiedParallel(long n)
        {
            var sqrt = (long)Math.Round(Math.Sqrt(n));
            var basic = Basic(sqrt);

            var primes = new List<long>(basic);
            var spinlock = new SpinLock();
            Parallel.For(sqrt + 1, n, (x, loopState) =>
            {
                var prime = true;
                for (int j = 0; j < basic.Count() && prime; j++)
                    prime = x % basic.ElementAt(j) != 0;

                if (prime)
                {
                    var lockTaken = false;
                    try
                    {
                        spinlock.Enter(ref lockTaken);
                        primes.Add(x);
                    }
                    finally
                    {
                        if (lockTaken) spinlock.Exit(false);
                    }
                }
            });
            re
```

*ModifiedTask* – це оптимізований алгоритм, паралельні обчислення з використанням бібліотеки TPL

```
public List<long> ModifiedTask(long n)
        {
            var sqrt = (long)Math.Round(Math.Sqrt(n));
            var basic = Basic(sqrt);

            var primes = new List<long>(basic);
            var tasks = new List<Task>();
            var spinlock = new SpinLock();
            for (long x = sqrt + 1; x <= n; x++)
            {
                var y = x;
                var task = Task.Factory.StartNew(() =>
                {
                    var prime = true;
                    for (int j = 0; j < basic.Count() && prime; j++)
                        prime = y % basic.ElementAt(j) != 0;

                    if (prime)
                    {
                        var lockTaken = false;
                        try
                        {
                            spinlock.Enter(ref lockTaken);
                            primes.Add(y);
                        }
                        finally
                        {
```

```
                        if (lockTaken) spinlock.Exit(false);
                    }
                }
            });
            tasks.Add(task);
        }

        Task.WaitAll(tasks.ToArray());
        return primes;
    }
```

Нарешті, метод *ModifiedPlinq* – це оптимізований алгоритм, паралельні обчислення з використанням PLINQ

```
public List<long> ModifiedPLinq(long n)
    {
        var sqrt = (long)Math.Round(Math.Sqrt(n));
        var basic = Basic(sqrt);

        var primes = new List<long>(basic);
        var aftersqrt = new List<long>();

        for (var i = sqrt + 1; i <= n; i++)
        {
            aftersqrt.Add(i);
        }

        var other = aftersqrt.AsParallel().Where(x =>
        {
            var prime = true;
            for (int j = 0; j < basic.Count() && prime; j++)
                prime = x % basic.ElementAt(j) != 0;
            return prime;
        });
        primes.AddRange(other);
        return primes;
    }
```

Таким чином, всі інструменти .NET дозволяють отримувати досить компактний код та забезпечувати оптимальну обробку даних під час виконання методів.

*Решето Аткіна*

Код методів, що реалізують пошук простих чисел за допомогою решета Аткіна, подано у Додатку Б.

Клас AtkinAlg наслідує від класу AAtkin

```
public class AtkinAlg : AAtkin
```

Його конструктор має вигляд

```
public AtkinAlg(int x)
        : base(x) {}
```

Метод *FindPrimes* - послідовний пошук простих чисел за медом Аткіна

```
public override void FindPrimes()
        {
            Primes = new bool[_limit + 1];
            double sqrt = Math.Sqrt(_limit);
            var limit = (ulong)_limit;
            for (ulong x = 1; x <= sqrt; x++)
                for (ulong y = 1; y <= sqrt; y++)
                {
                    ulong x2 = x * x;
                    ulong y2 = y * y;
                    ulong n = 4 * x2 + y2;
                    if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                        Primes[n] ^= true;

                    n -= x2;
                    if (n <= limit && n % 12 == 7)
                        Primes[n] ^= true;

                    n -= 2 * y2;
                    if (x > y && n <= limit && n % 12 == 11)
                        Primes[n] ^= true;
                }

            for (ulong n = 5; n <= sqrt; n += 2)
                if (Primes[n])
                {
                    ulong s = n * n;
                    for (ulong k = s; k <= limit; k += s)
                        Primes[k] = false;
                }
            Primes[2] = true;
            Primes[3] = true;
        }
```

Метод *FindPrimesParallel* – це паралельний пошук простих чисел із використанням бібліотеки TPL

```
public void FindPrimesParallel()
        {
            Primes = new bool[_limit + 1];
            var sqrt = Math.Ceiling(Math.Sqrt(_limit));
            var limit = (ulong)_limit;
            Parallel.For(1, (long)sqrt,
            (x) =>
            {
                Parallel.For(1, (long)sqrt, (y) =>
                {
                    ulong x2 = (ulong)(x * x);
                    ulong y2 = (ulong)(y * y);
                    ulong n = 4 * x2 + y2;
                    if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                        Primes[n] ^= true;

                    n -= x2;
                    if (n <= limit && n % 12 == 7)
                        Primes[n] ^= true;

                    n -= 2 * y2;
                    if (x > y && n <= limit && n % 12 == 11)
                        Primes[n] ^= true;
                });
```

```
        });
        for (ulong n = 5; n <= sqrt; n += 2)
            if (Primes[n])
            {
                ulong s = n * n;
                for (ulong k = s; k <= limit; k += s)
                    Primes[k] = false;
            }
        Primes[2] = true;
        Primes[3] = true;
    }
```

Метод *FindPrimesPlinq* – це паралельний пошук простих чисел з використанням PLINQ

```
public void FindPrimesPlinq()
    {
        Primes = new bool[_limit + 1];
        double sqrt = Math.Sqrt(_limit);
        var limit = (ulong)_limit;
        var a = new List<ulong>((int)Math.Ceiling(sqrt));
        for (ulong i = 1; i <= sqrt; i++)
        {
            a.Add(i);
        }

        a.AsParallel().ForAll(x =>
        {
            a.AsParallel().ForAll(y =>
            {
                ulong x2 = x * x;
                ulong y2 = y * y;
                ulong n = 4 * x2 + y2;
                if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                    Primes[n] ^= true;

                n -= x2;
                if (n <= limit && n % 12 == 7)
                    Primes[n] ^= true;

                n -= 2 * y2;
                if (x > y && n <= limit && n % 12 == 11)
                    Primes[n] ^= true;
            });
        });

        for (ulong n = 5; n <= sqrt; n += 2)
            if (Primes[n])
            {
                ulong s = n * n;
                for (ulong k = s; k <= limit; k += s)
                    Primes[k] = false;
            }
        Primes[2] = true;
        Primes[3] = true;
    }
```

*Життя «Конвея»*

Код методів, що реалізують гру «Життя» Конвея, подано у Додатку В.

Зміни в кожній комірці згідно алгоритму обчислюються методом UpdateCell

```
void UpdateCell(int x, int y)
 {
     var neighbors = HaveNeighbor(x, y, -1, 0)
                   + HaveNeighbor(x, y, -1, 1)
                   + HaveNeighbor(x, y, 0, 1)
                   + HaveNeighbor(x, y, 1, 1)
                   + HaveNeighbor(x, y, 1, 0)
                   + HaveNeighbor(x, y, 1, -1)
                   + HaveNeighbor(x, y, 0, -1)
                   + HaveNeighbor(x, y, -1, -1);

     var shouldLive = false;

     var isAlive = world[x, y];

     if (isAlive && (neighbors == 2 || neighbors == 3))
     {
         shouldLive = true;
     }
     else if (!isAlive && neighbors == 3)
     {
         shouldLive = true;
     }
     updated[x, y] = shouldLive;
 }

 private int HaveNeighbor(int x, int y, int offsetx, int offsety)
 {
     var alive = 0;

     var proposedOffsetX = x + offsetx;
     var proposedOffsetY = y + offsety;
     var outOfBounds = proposedOffsetX < 0 || proposedOffsetX >= size |
proposedOffsetY < 0 || proposedOffsetY >= size;
     if (!outOfBounds)
     {
         alive = world[x + offsetx, y + offsety] ? 1 : 0;
     }
     return alive;
 }
```

Для послідовного процесу перехід на наступний крок відбувається за методом Update

```
public void Update()
 {
     for (int x = 0; x < size; x++)
     {
         for (int y = 0; y < size; y++)
         {
             UpdateCell(x, y);
         }
     }
```

```
        UpdateWorld();
    }
```

При паралельній обробці з використанням *Tasks.Parallel* розпаралелюється кожна комірка

```
public void UpdateParallel()
       {
    Parallel.For(0, size, x =>
    {
        Parallel.For(0, size, y =>
        {
            UpdateCell(x, y);
        });
    });
    UpdateWorld();
}
```

При паралельній обробці з використанням *PLINQ* також обробляється кожна комірка

```
public void UpdatePLinq()
{
    var cells = new List<Tuple<int, int>>();
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            cells.Add(new Tuple<int, int>(i, j));
        }
    }
    cells.AsParallel().ForAll(x => UpdateCell(x.Item1, x.Item2));
    UpdateWorld();
}
```

Оновлення всього «світу» в кінці епохи здійснюється методом Update World

```
void UpdateWorld()
       {
           for (int x = 0; x < size; x++)
           {
               for (int y = 0; y < size; y++)
               {
                   world[x, y] = updated[x, y];
               }
           }
       }
```

Таким чином, в грі розпаралелено обробку комірок, що дозволяє швидко обробляти великі «світи», тобто матриці з комірками.

*Обробка тестів*

Структура тесту, представлена вище на рис.3.4, релазіована наступним чином

```
public Test(string path)
        {
            content = new List<Item>();
            RichTextBox rtb = new RichTextBox();
            string ext = Path.GetExtension(path);
            if (ext == ".rtf")
                rtb.LoadFile(path);
            else
                rtb.Text = File.ReadAllText(path);

            string[] all_lines = rtb.Lines;                 //считываем с rtb в массив
строк
            all_lines = Ceasar.Decrypt(all_lines);          //расшифровываем

            int i = -1;
            while (++i < all_lines.Length)
            {
                string current = all_lines[i];
                if (current.Length > 0)
                {
                    if (current[0] == '#') //новый вопрос
                    {
                        Item item = new Item();
                        item.Question = current;
                        item.Vars = Convert.ToInt32(all_lines[++i]);
                        item.Right = Convert.ToInt32(all_lines[++i]);

                        for (int j = 0; j < item.Vars; j++)
                            item.Answers.Add(all_lines[++i]);

                        content.Add(item);
                    }
                }
            }

        }
```

Шифрування за шифром Цезаря здійснюється методами класу Ceasar.cs (Додаток Г). Для шифрування обрано крок 17.

Метод Encypt зашифровує текст

```
public static string[] Encrypt(string[] content)
        {
            string[] encoded = new string[content.Length];

            for (int i = 0; i < content.Length; i++)
            {
                encoded[i] = string.Empty;
                for (int j = 0; j < content[i].Length; j++)
                {
                    int pos = GetPos(content[i][j]);
                    if (pos != -1)
                        encoded[i] += alphabet[(pos + crypto) % alphabet.Length];
```

```
                else
                    encoded[i] += content[i][j];
            }
        }
        return encoded;
    }
```

Метод Decrypt розшифровує текст

```
public static string[] Decrypt(string[] content)
    {
        string[] decoded = new string[content.Length];

        for (int i = 0; i < content.Length; i++)
        {
            decoded[i] = string.Empty;
            for (int j = 0; j < content[i].Length; j++)
            {
                int pos = GetPos(content[i][j]);
                if (pos != -1)
                    decoded[i] += alphabet[(alphabet.Length + pos - crypto) %
alphabet.Length];
                else
                    decoded[i] += content[i][j];
            }
        }
        return decoded;
    }
```

Результат шифрування представлено на рис. 3.5.


#аырчшБх юх67ртшьКюях ВБтх7чфхюшх
o
P
жZжU - ЛБя ясЙЁюЙх 6яАьхфятрБхьКюЙх ыяэ6КМБх7Й
жZбU - сяьКЖщюАБтя Аят7хэхююЙД kТЭ яБюяАНБАН ы ЛБяъ ырБхуя7щщ
бZжU - тЙЁшАьшБхьКюЙД эрЖшю Брыяуя ыьрААр эрья
бZбU -ЛБя 7хрьшшрЕшН юхАыяьКышД 6яБяыят ыяэрюф щ 6яБяыят
фрююЙД

#ФьН ыяютхъх7юяъ яс7рсяБыщ 67щАВ3х
o
Q
шру7Вшыр я6х7рюфят т тхыБя7юЙх 7хущАБ7Й
я6х7рЕшщ А эрБ7шЕрэщ
тЙфхьхющх яБфхьКюЙД ЛБр6ят тЙ6яьюхющН яс3хъ я6х7рЕщш
Аьячхющх Р-Д я6х7рюфят яфюят7хэхююЙэ Аьячхющхэ тАхД шД фтящЁюЙД
7рш7Нфят

#0Бхы - ЛБя...
o
Q
"6рэНБК", т рф7хАюяэ 67яАБ7рюАБтх ыяБя7яъ 7рсяБрхБ 67яЕхАА
БяБ щъщ шюяъ А6яАяс 6х7хфрЁщ шюАБ7ВыЕшщ щш яфюяуя 67яЕхААр т
ф7Вуяъ
ясьрАБК 6рэНБш фьН ьяырьКюЙД 6х7хэхююЙД, р7уВэхюБят щ
тяшт7р3рхэЙД ГВюыЕшНэш шюрЁхюшъ

Рис.3.5. Зашифрований тест у текстовому файлі

### 3.3 Опис функціоналу

*Головне вікно додатку*

При запуску додатку відкривається головне вікно з головним меню, що містить три опції: "Теоретичні відомості", "Демонстрація" та "Тести" (рис.3.6).

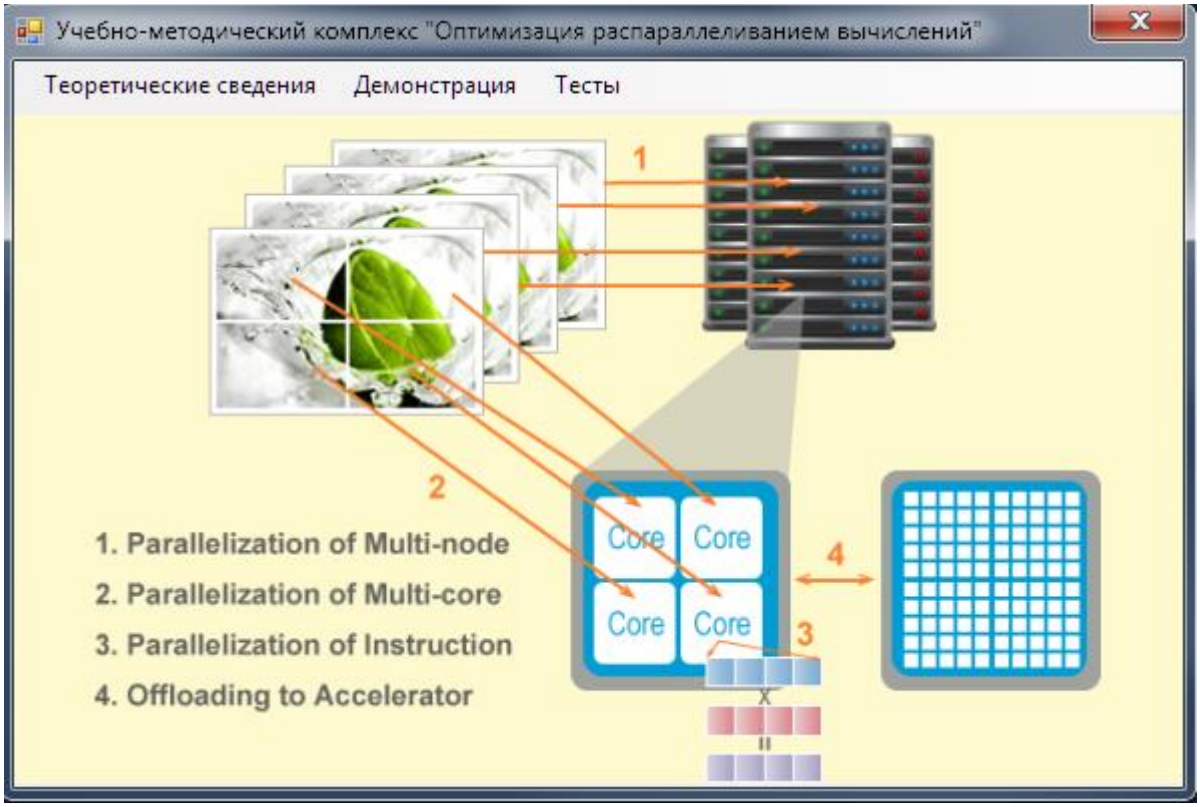



Рисунок 3.6 — Головне вікно додатку

Головне вікно успішно відкривається. Переходимо до тестування роботи головного меню.

*Опція «Теоретичні відомості»*

Тестування підсистеми роботи з теоретичним матеріалом виконується в наступній послідовності:

1. Запустити програму, натиснути на опцію меню «Теоретичні відомості».
2. Ознайомитись із теорією.
3. При переходах сторінками переконатися, що сторінки правильно змінюють своє значення.

4. Закрити вікно "Теоретичні відомості"

При виборі цієї опції відкривається дочірнє вікно, що містить WebBrowser, в який завантажується HTML-проект з теоретичними відомостями з паралельних обчислень (рис.3.7).

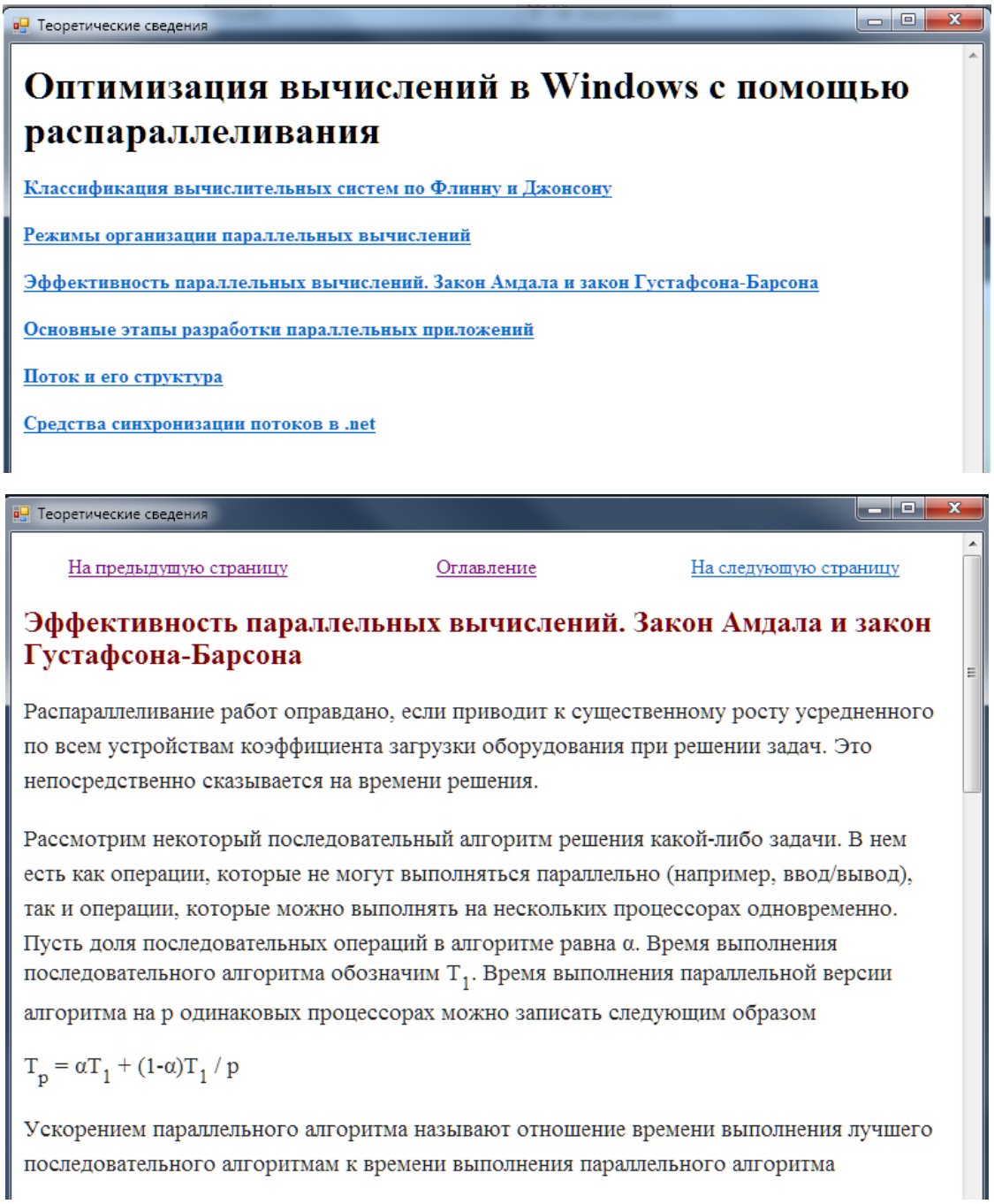



Рисунок 3.7 — Відкриття форми з WebBrowser, що містить теоретичні відомості про паралельні обчислення

Робота з теоретичним матеріалом здійснюється успішно, перехід від змісту до сторінок, з однієї сторінки на іншу за гіперпосиланнями працює.

*Опція «Демонстрація»*

Тестування підсистеми демонстрації проводиться у наступній послідовності:

1. Вибрати завдання демонстрації. Відкрити дочірнє вікно.
2. Ввести початкові дані.
3. Ознайомитись із результатами.
4. Очистити форму. Повторити з іншими початковими даними
5. Закрити вікно.

Демонстрацію ведеться за вибором одного із трьох доступних завдань: Решето Ератосфена, Решето Аткіна, «Життя» Конвея (рис.3.8).

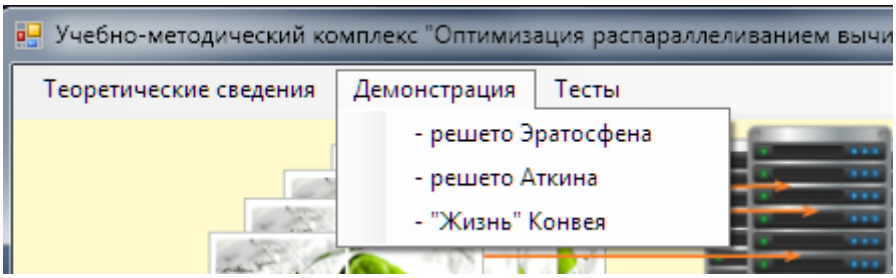



Рисунок 3.8 — Вибір завдання для демонстрації

Для всіх режимів демонстрації відкривається вікно Report, кількість рядків у звіті залежить від вибору (рис.3.9).

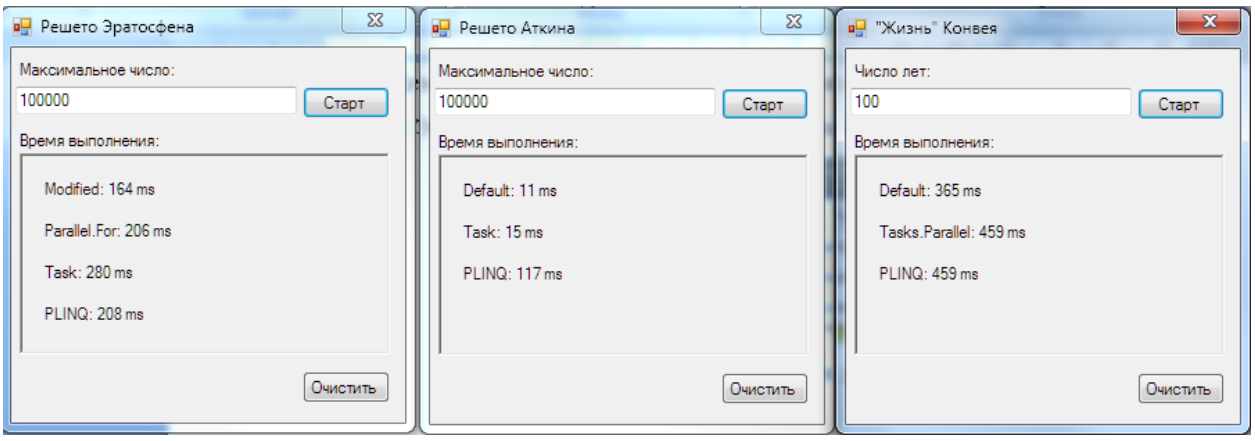



Рисунок 3.9 — Приклади замірів часу при розв’язку демонстраційних завдань

Розв’язок задач паралельного програмування у режимі демонстрація проходить успішно. Звіти із роботи на даній конфігурації генеруються для всіх обраних режимів.

*Опція «Тести»*

Для цієї опції пропонується два режими роботи: «Студент» та «Викладач» (рис.3.10).

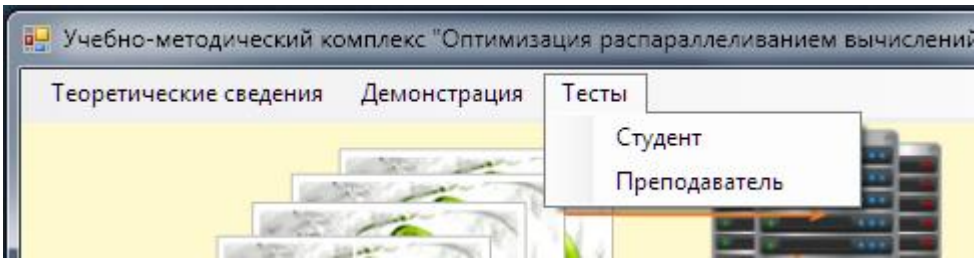



Рисунок 3.10 — Два режими роботи з підсистемою тестів

Перевірка підсистеми тестування проводиться в наступній послідовності:

1. Для входу як викладач ввести ім'я та пароль.
2. У вікні редагування створити файл із новим тестом. За бажанням – запам'ятати його: опція Файл-Зберегти. Також можна відкрити файл, створений раніше: опція Файл-Відкрити.
3. Зашифрувати файл: Шифрування – Зашифрувати
4. Зберегти зашифрований файл: Файл-Зберегти як
5. Для входу як студент ввести ім'я та пароль, натиснути кнопку "Почати тест".
6. Буде виведено 10 випадкових питань, у кожному з яких по 3-4 варіанти відповіді.
7. Для відповіді встановити прапорець біля варіанта відповіді.
8. За результатами тестування буде виведено вікно з кількістю правильних та неправильних відповідей.
9. Закрити вікно "Вхід".

Обираємо режим «Викладач», відкривається дочірнє вікно «Вхід», де необхідно ввести ім’я та пароль (рис.3.11).

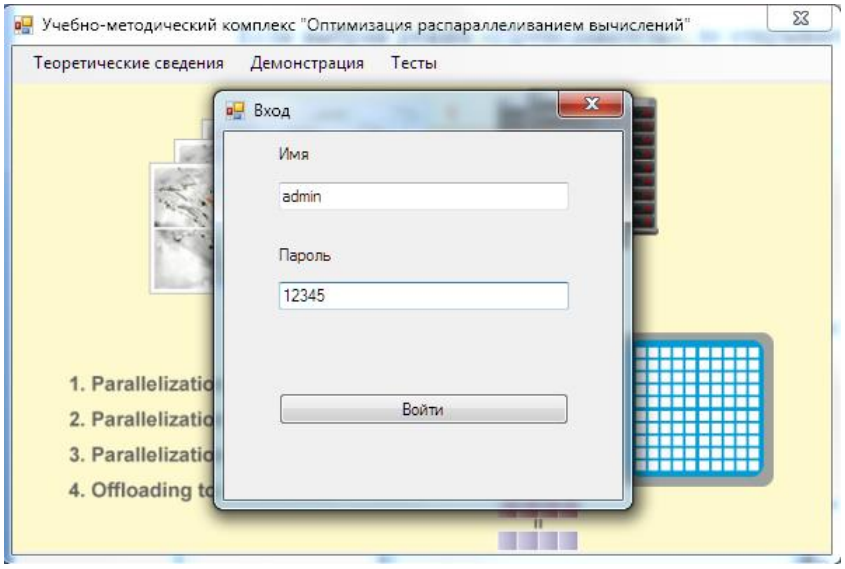



Рисунок 3.11 — Вхід в систему з правами викладача

Редактор тестів дає можливість створювати, відкривати, редагувати, запам'ятовувати файли, розшифровувати та зашифровувати їх (рис. 3.12).

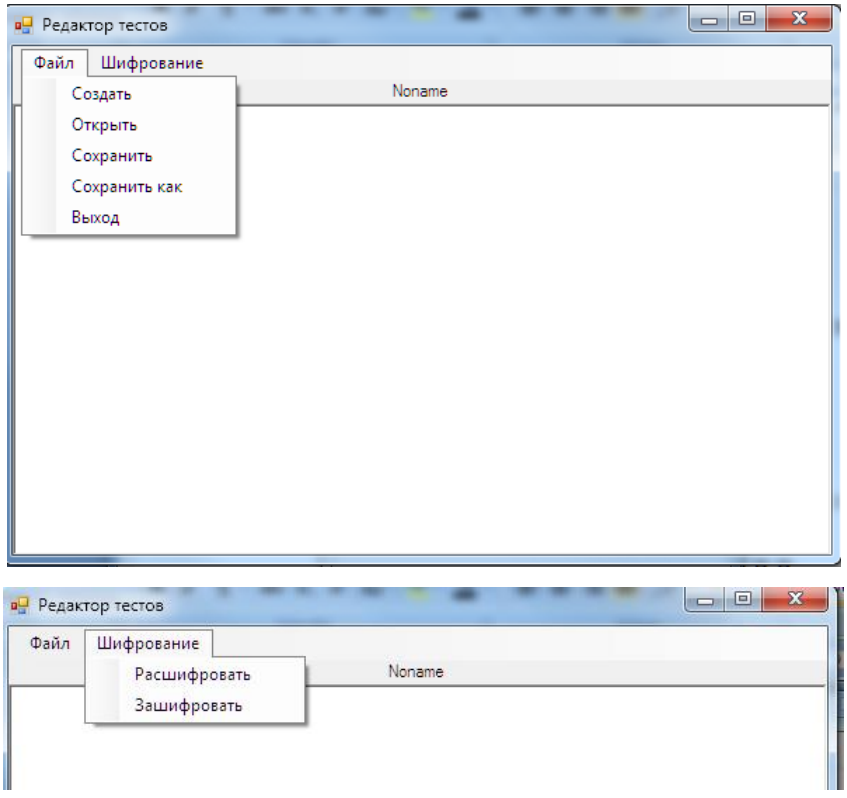



Рисунок 3.12 — Можливості роботи в редакторі тестів

У режимі «Студент» необхідно ввести ім’я та групу, щоб отримати можливість здати тест (рис.3.13).

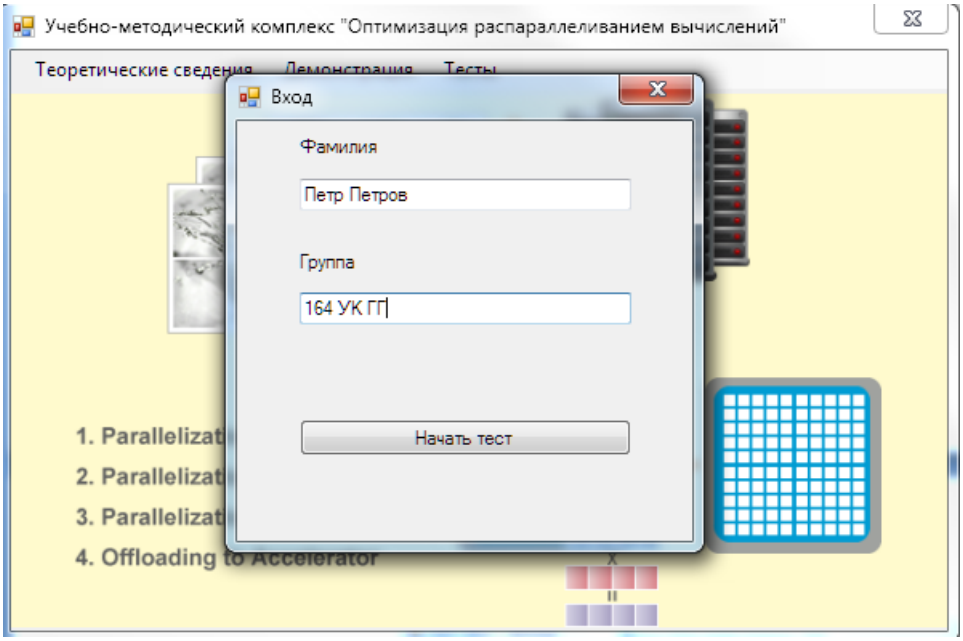



Рисунок 3.13 — Вхід в систему з правами студента

Після цього у файловому діалозі вибирається зашифрований файл із тестом. Незашифрований файл коректно не працюватиме (рис.3.14).

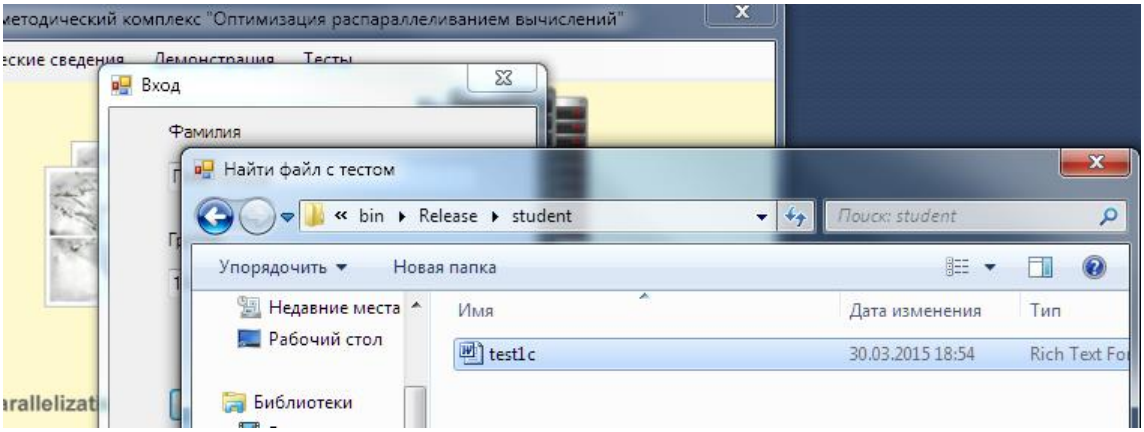



Рисунок 3.14 — Вибір файлу для проходження тесту

Після того, як файл обрано, відкривається вікно тестування з 10 випадково обраними питаннями (рис.3.15).

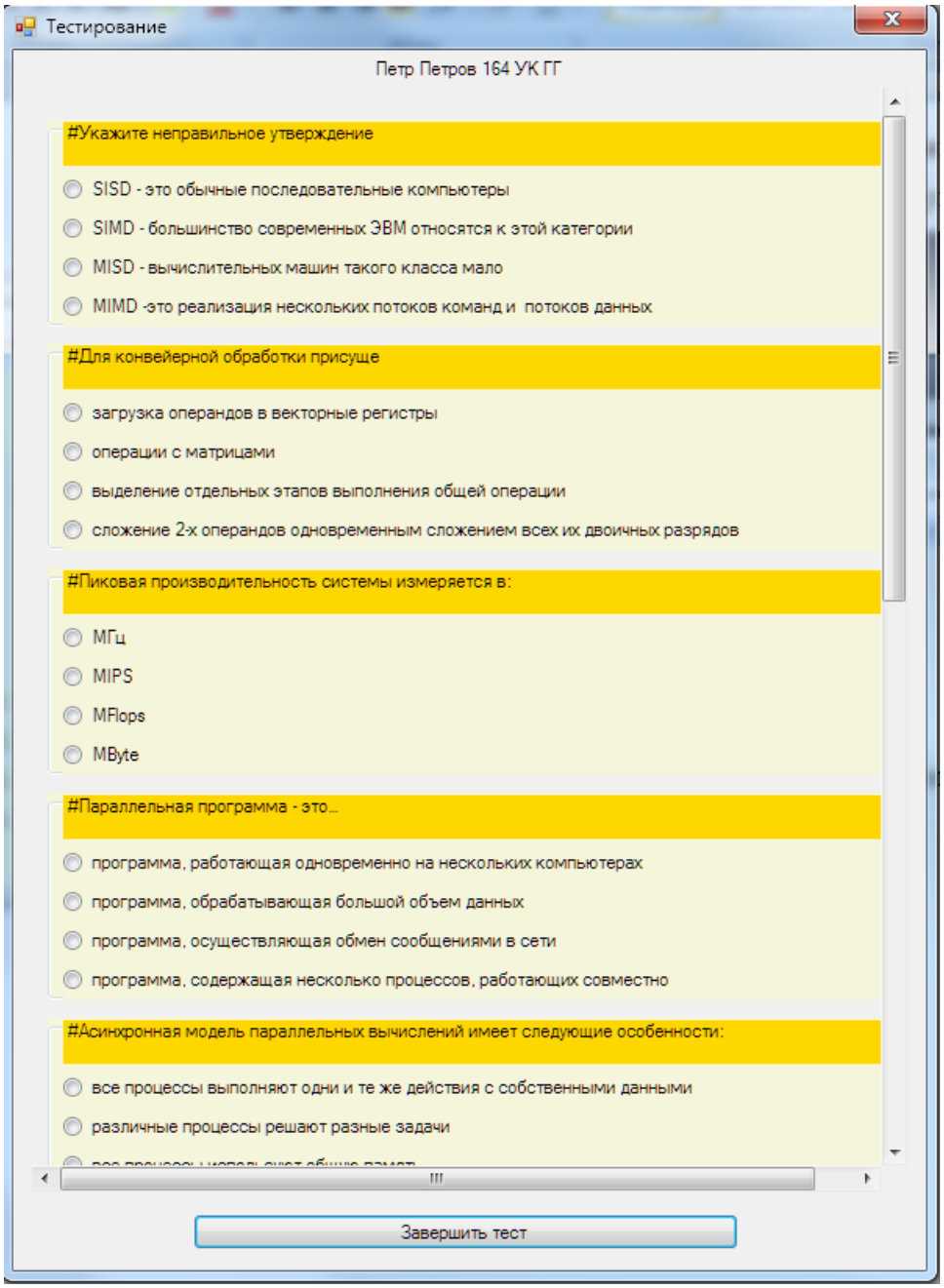



Рисунок 3.15 — Вікно тестування

Результат тестування з'являється як повідомлення MessageBox (рис.3.16) і записується у файл statisctics.txt (рис.3.17).

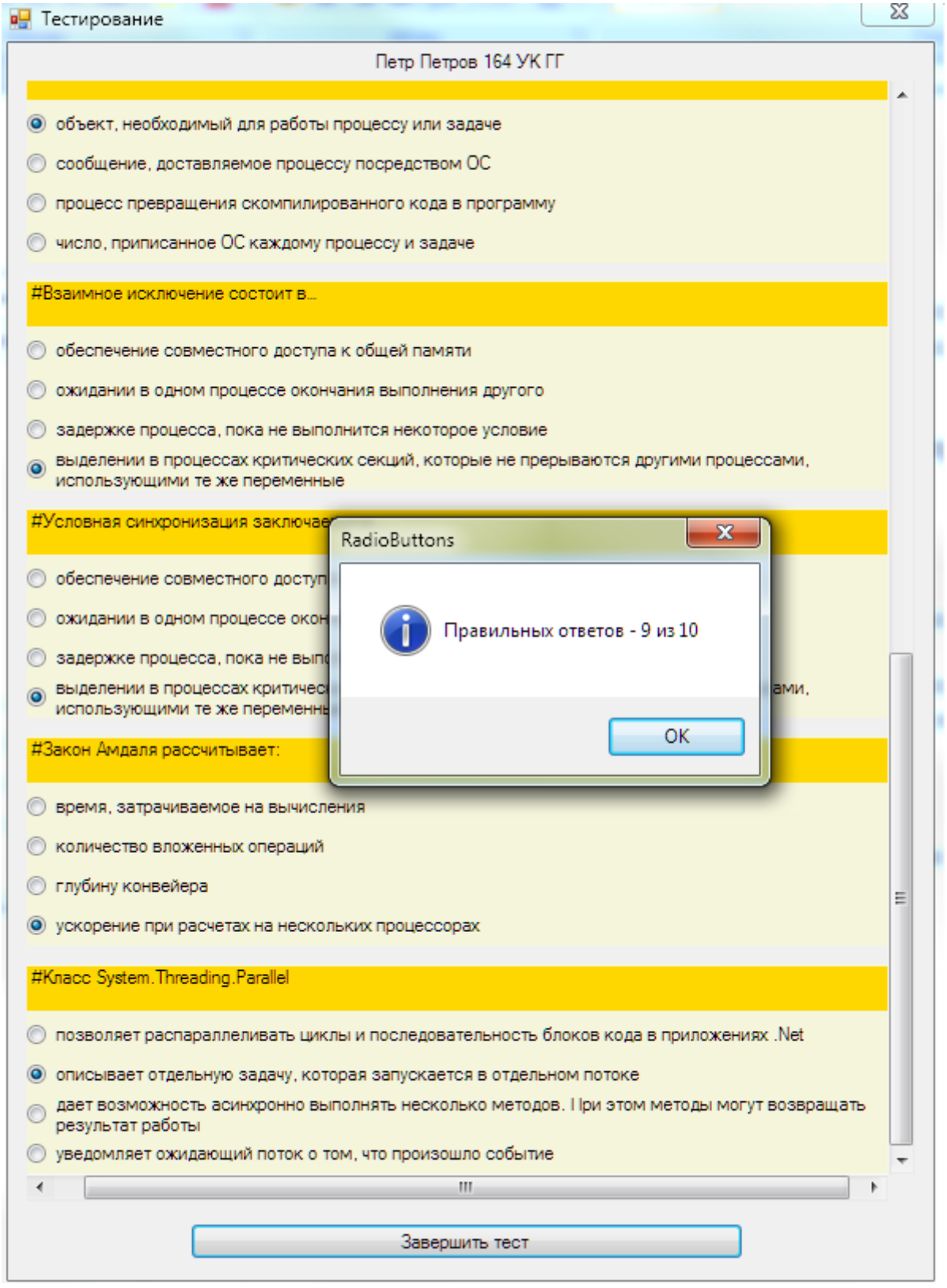



Рисунок 3.16 — Результат тестування

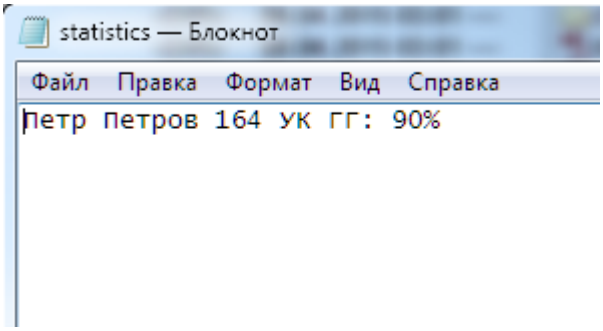



Рисунок 3.17 — Запис результатів у файл statisctics.txt

Записи у файл statisctics.txt ведуться як додавання (append), тобто. викладач має можливість підбити підсумки після того, як вся група здасть тести.

Робота у додатку може проводитися паралельно, оскільки. дочірні вікна за опціями головного меню (теорія, демонстрація, тест) є незалежними.

Це надає додаткові вигоди, під час підготовки до тесту можна заглянути в теорію або перевірити свої припущення на практиці, запустивши демонстрацію.

Таким чином, функціональні вимоги до додатку повністю реалізовані.

## 3.4 Тестування ефективності розпаралелювання

Розглянемо, які можливості надають паралельні обчислення для прискорення розв’язку задач, представлених у демонстраційному розділі.

Паралельні обчислення тестувалися на кількох ПК з різною конфігурацією (табл.3.1)

Таблиця 3.1 — Характеристики систем (CPU, платформа, компілятор), для котрих проводилось тестування паралельних обчислень

| Розв'язок | Кількість ядер CPU | Кількість логічних процесорів CPU | Платформа | Компілятор |
|---|---|---|---|---|
| S1 | 1 | 2 | 32 bit | VS 2015 |
| S2 | 2 | 2 | 32 bit | VS 2015 |
| S3 | 4 | 4 | 32 bit | VS 2015 |
| S4 | 4 | 4 | 64 bit | VS 2015 |
| S5 | 4 | 4 | 64 bit | VS 2017 |

*Решето Ератосфена*

Заміри часу виконання для системи S1 (1 ядро, 2 логічні процесори, 32 біт, VS 2015) представлені на рис.3.18.

Ми бачимо, що, незважаючи на 1 фізичне ядро, 2 логічні процесори виконують роботу з розпаралелювання обчислень. Рішення з використанням TPL найбільше ефективно для малих N ~ $10^4$. Зі зростанням N ефективність цього підходу втрачається. Рішення з використанням PLINQ є найбільш ефективним для N ~ $10^5$.

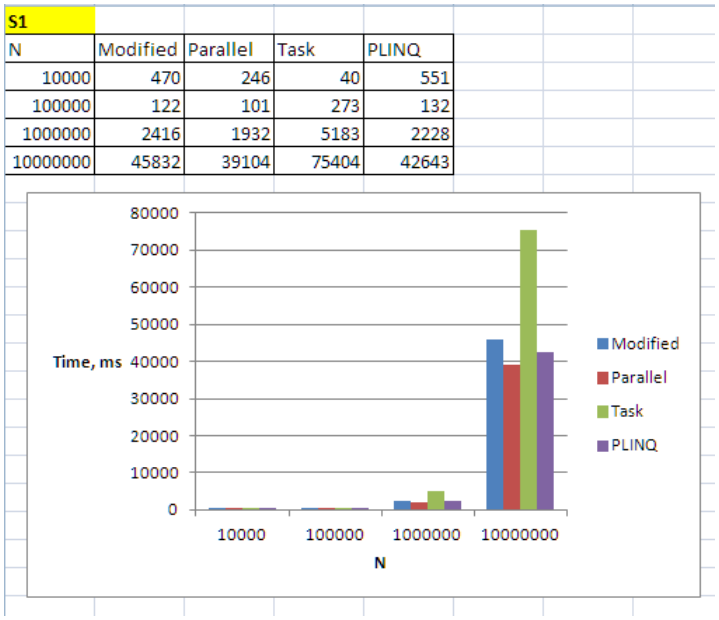



Рисунок 3.18 — Система S1 - час виконання обчислень для різних модифікацій решета Ератосфена

Для системи S2 зберігаються подібні закономірності (рис.3.19). Але в цілому, 2 фізичні ядра виконують обчислення швидше, незважаючи на меншу частоту процесора.

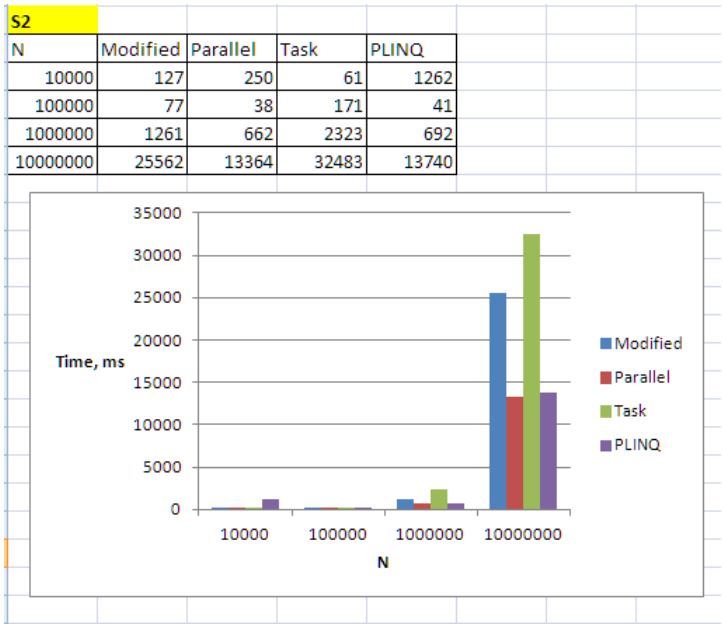



Рисунок 3.19 — Система S2 - час виконання обчислень для різних модифікацій решета Ератосфена

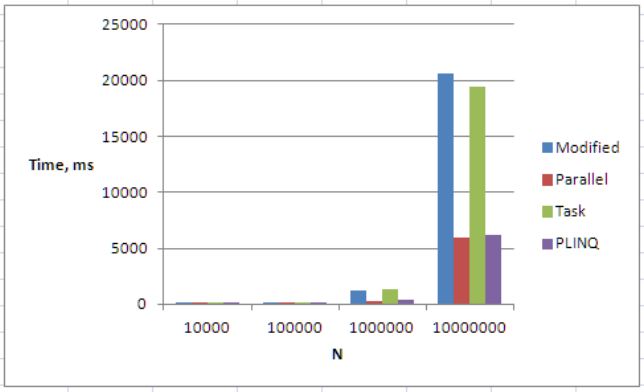
Системи S3-S5 (рис.3.20) реалізовані на одному і тому ж ПК з 4 ядрами та 4 логічними процесорами, частотою 2,66 GHz. Вони відрізняються платформою – 32 або 64 біти, та компілятором, в якому зібрані додатки – VS2015 та VS2017.

Збірка 32 біта дуже швидко працює на 64-бітній машині для малих $N \sim 10^4$. Для $N \sim 10^6$ і вище таке збірка стає повільнішою (рис.3.21). 64-бітова збірка в VS2015 працює повільніше, ніж у VS2017, особливо на малих N (рис.3.22).

Загалом 4 ядра помітно швидше справляються з обчисленнями, ніж 2. Рішення з використанням TPL, як і раніше, найефективніше для $N \sim 10^4$. А рішення з використанням PLINQ - для $N \sim 10^5$. Помітна також висока ефективність низькорівневої синхронізації.

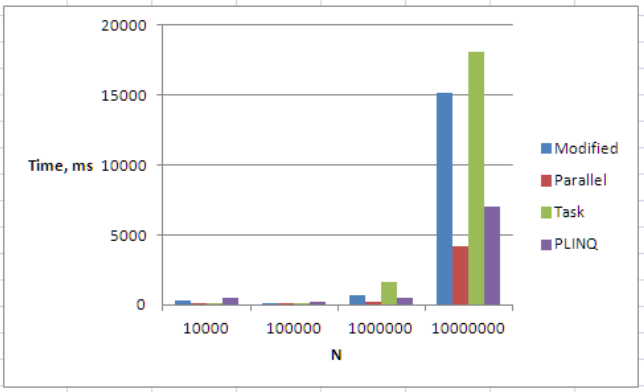
S3

| N | Modified | Parallel | Task | PLINQ |
|---|---|---|---|---|
| 10000 | 64 | 6 | 15 | 113 |
| 100000 | 54 | 23 | 84 | 35 |
| 1000000 | 1162 | 330 | 1307 | 377 |
| 10000000 | 20564 | 5978 | 19473 | 6211 |




S4

| N | Modified | Parallel | Task | PLINQ |
|---|---|---|---|---|
| 10000 | 360 | 163 | 82 | 491 |
| 100000 | 40 | 16 | 148 | 178 |
| 1000000 | 737 | 213 | 1641 | 490 |
| 10000000 | 15150 | 4224 | 18046 | 7004 |




S5

| N | Modified | Parallel | Task | PLINQ |
|---|---|---|---|---|
| 10000 | 188 | 169 | 18 | 463 |
| 100000 | 41 | 14 | 127 | 47 |
| 1000000 | 823 | 273 | 1496 | 350 |
| 10000000 | 15309 | 4381 | 20705 | 5628 |

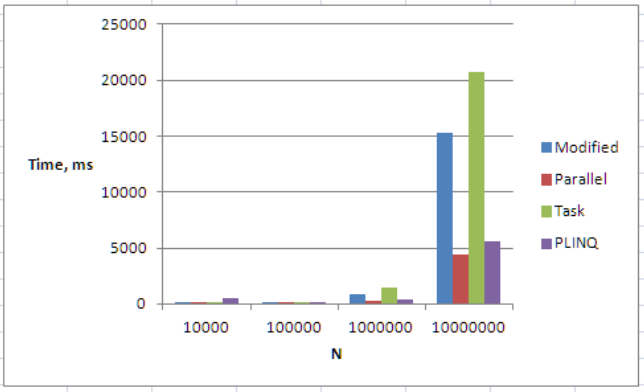



Рисунок 3.20 — Системи S3-S5 - час виконання обчислень для різних модифікацій решета Ератосфена

| N = 10000000 | | | | |
|---|---|---|---|---|
| S | Modified | Parallel | Task | PLINQ |
| S1 | 45832 | 39104 | 75404 | 42643 |
| S2 | 25562 | 13364 | 32483 | 13740 |
| S3 | 20564 | 5978 | 19473 | 6211 |
| S4 | 15150 | 4224 | 18046 | 7004 |
| S5 | 15309 | 4381 | 20705 | 5628 |

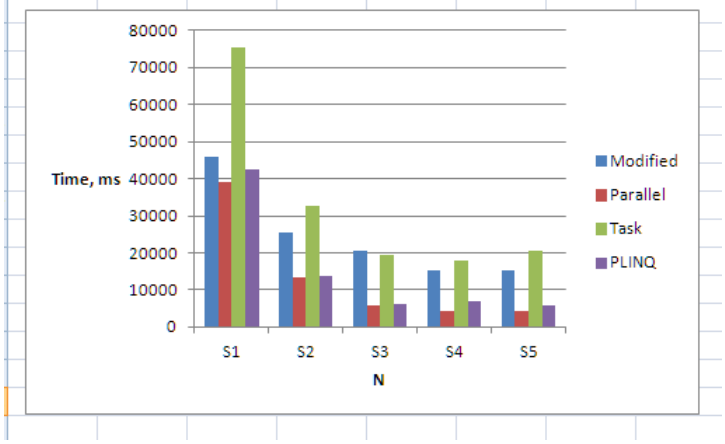



Рисунок 3.21 — Ефективність різних систем для решета Ератосфена при великих N ~ $10^7$

| N = 10000 | | | | |
|---|---|---|---|---|
| S | Modified | Parallel | Task | PLINQ |
| S1 | 470 | 246 | 40 | 551 |
| S2 | 127 | 250 | 61 | 1262 |
| S3 | 64 | 6 | 15 | 113 |
| S4 | 360 | 163 | 82 | 491 |
| S5 | 188 | 169 | 18 | 463 |

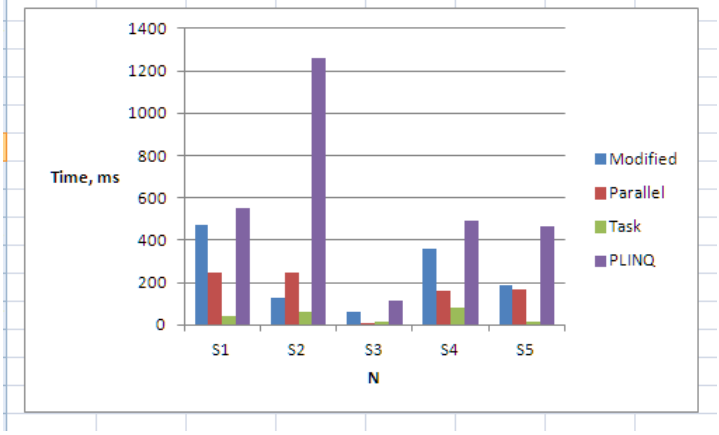



Рисунок 3.22 — Ефективність різних систем для решета Ератосфена при малих N ~ $10^4$

Для великих N~$10^7$ найбільш ефективним підходом є низькорівнева синхронізація. Трохи гірший час обчислень із використанням PLINQ. У той

самий час, для малих N ~ $10^4$ найбільш ефективним використання бібліотеки TPL.

Таким чином, вибір технології розпаралелювання завдання залежить від порядку N.

*Решето Аткіна*

Результати замірів часу на різних системах подані на рис.3.23-3.24.

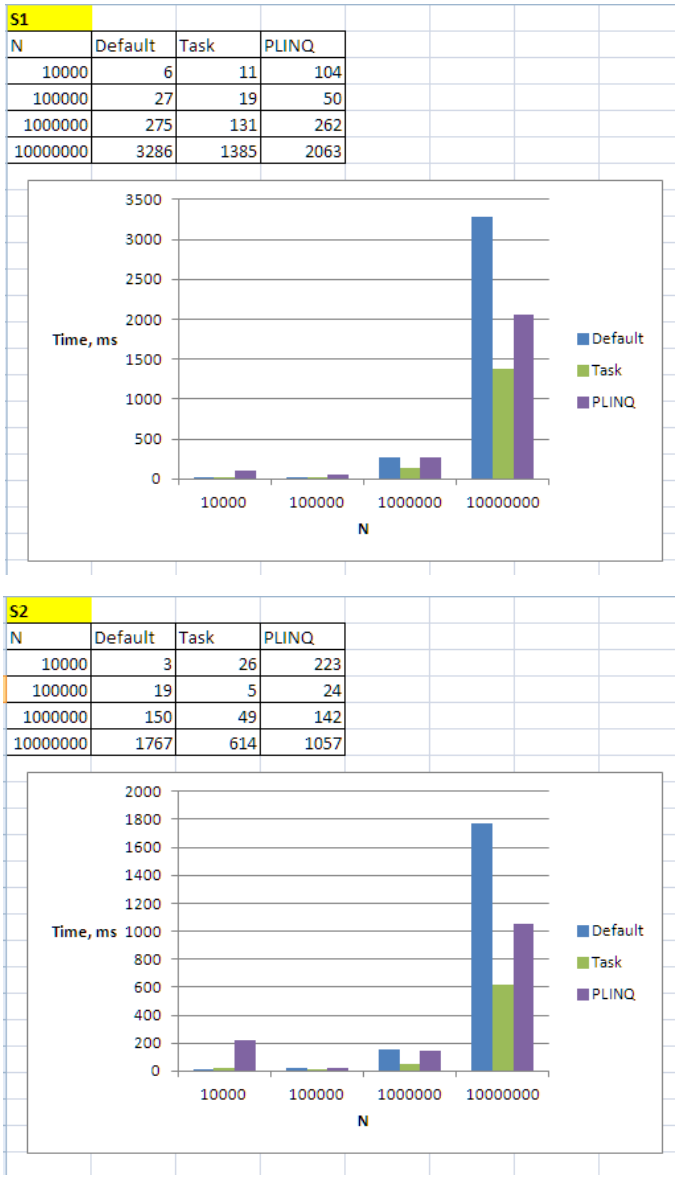



Рисунок 3.23 — Системи S1, S2 - час виконання обчислень для різних модифікацій решета Аткіна

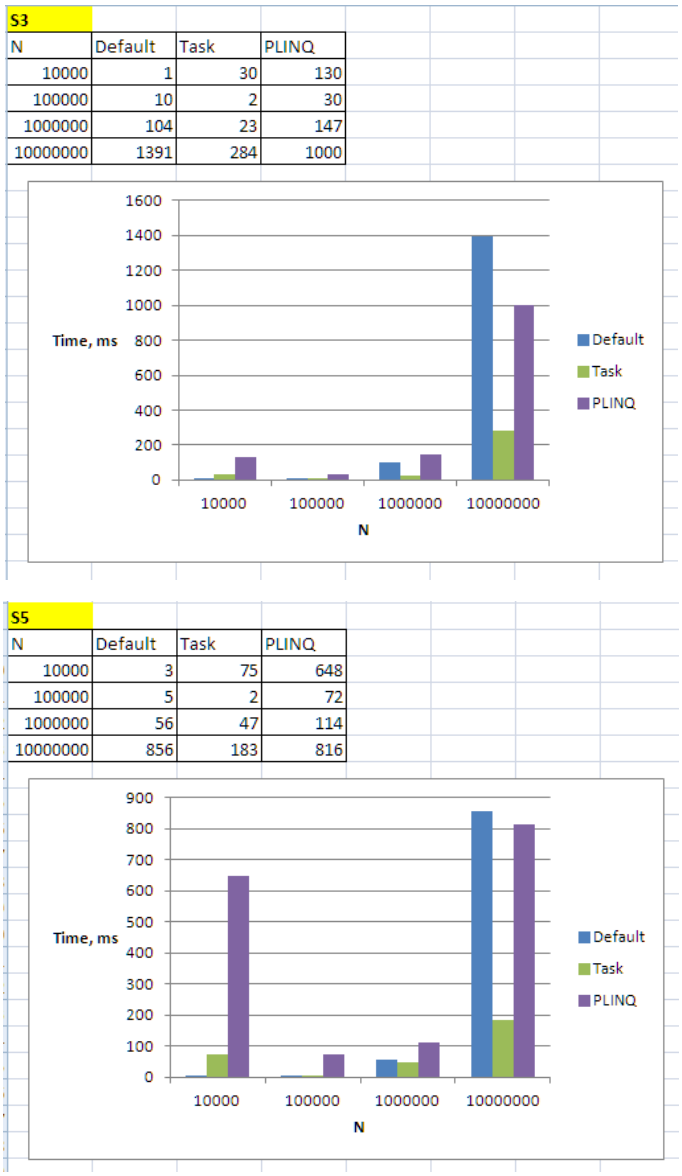



Рисунок 3.24 — Системи S3, S5 - час виконання обчислень для різних модифікацій решета Аткіна

Згідно з наведеними даними, найбільш ефективним для реалізації решета Аткіна є метод паралельного пошуку з використанням бібліотеки TPL.

Для великих N~$10^7$ найбільш ефективним підходом є TPL (рис.3.25). У той самий час, для малих N ~ $10^4$ найефективнішим є звичайний послідовний пошук без розпралелювання (рис.3.26).

| N = 10000000 | | | |
|---|---|---|---|
| S | Default | Task | PLINQ |
| S1 | 3286 | 1385 | 2063 |
| S2 | 1767 | 614 | 1057 |
| S3 | 1391 | 284 | 1000 |
| S5 | 856 | 183 | 816 |

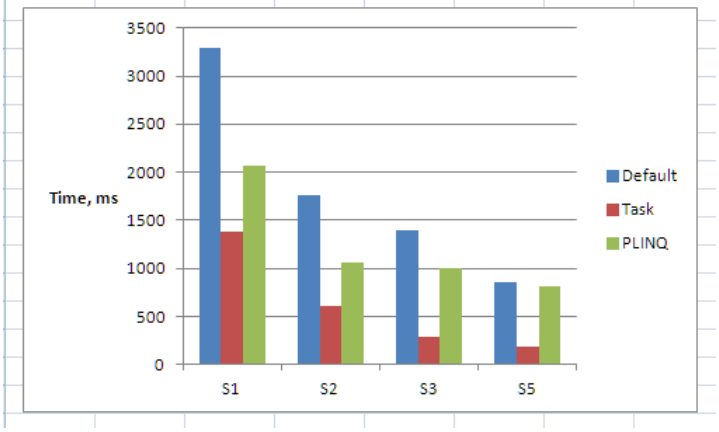



Рисунок 3.25 — Ефективність різних систем для решета Аткіна при великих N ~ $10^7$

| N = 10000 | | | |
|---|---|---|---|
| S | Default | Task | PLINQ |
| S1 | 6 | 11 | 104 |
| S2 | 3 | 26 | 223 |
| S3 | 1 | 30 | 130 |
| S5 | 3 | 75 | 648 |

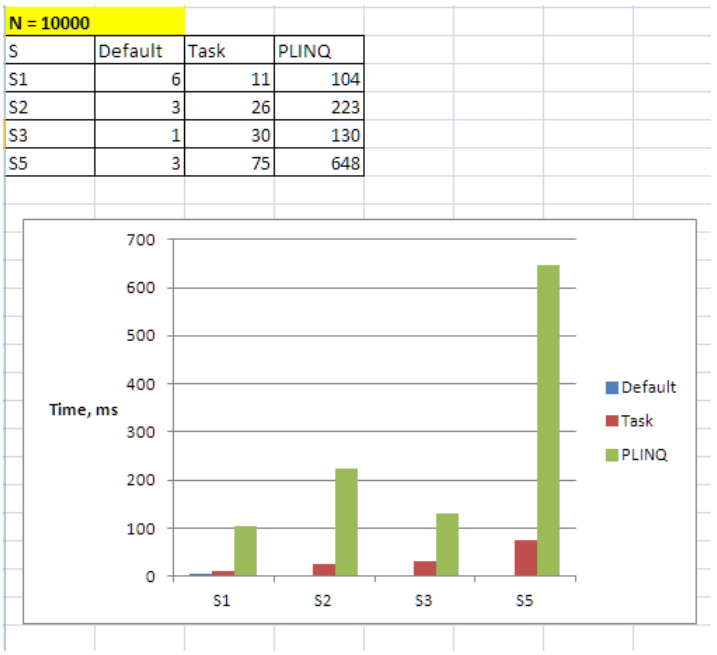



Рисунок 3.26 — Ефективність різних систем для решета Аткіна при малих N ~ $10^4$

Таким чином, вибір технології розпаралелювання при пошуку за решетом Аткіна також залежить від порядку N. Але, на відміну від решета

Ератосфена, для решета Аткіна можна вказати універсальний підхід, що дає відносно добрі результати на всіх конфігураціях ПК – це розпаралелювання з використанням TPL.

*Гра «Життя» Конвея*

Тестування програми проводилося на тих самих п'яти системах ПК і збірок, що використовувалися при пошуку простих чисел (див. тбал.3.1).

Вивчалося три режими роботи:

N = 10, тобто. 10 «років» на полі 10 x 10;

N = 100, тобто. 100 «років» на полі 100 x 100;

N = 1000, тобто. 1000 «років» на полі 1000 x 1000;

Заміри часу, виконані щодо різних систем, представлені на рис.3.27 – 3.28.

Як бачимо, ефективність підходів до розв’язку задачі помітно залежить від розмірності N та кількості ядер процесора (рис.3.29).

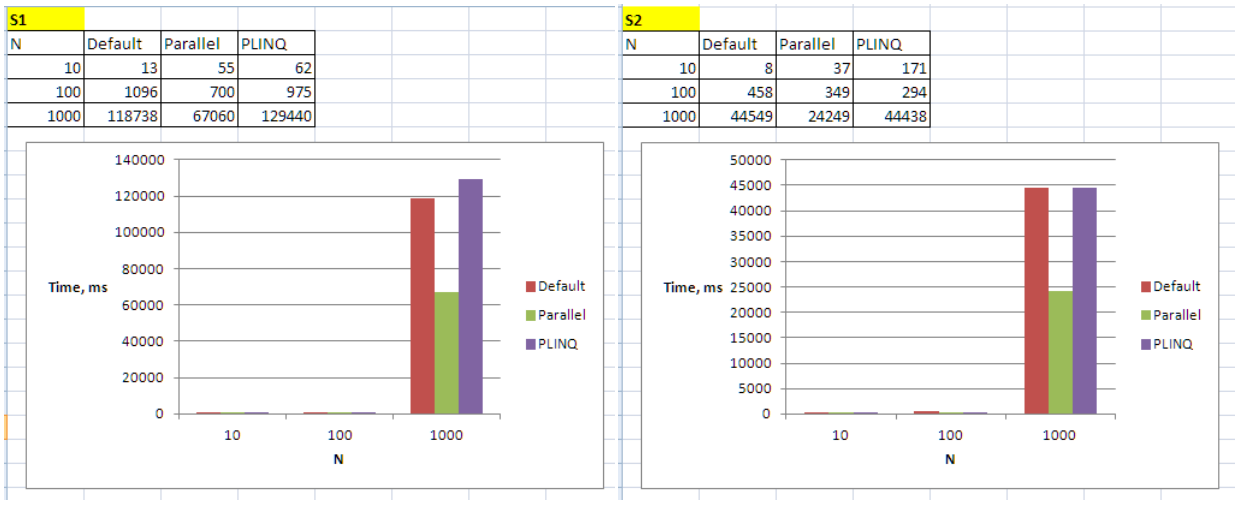
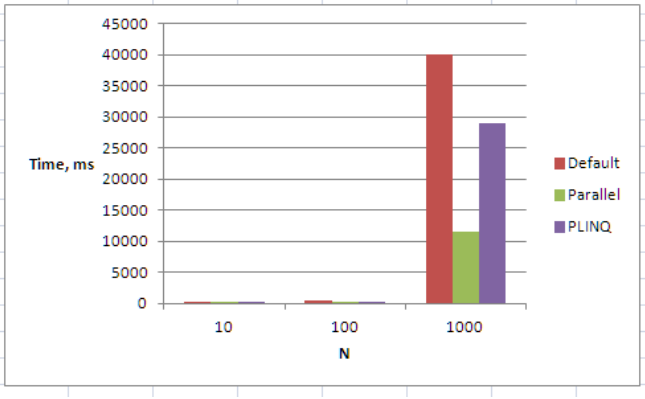



Рисунок 3.27 — Системи S1, S2 - час виконання обчислень для різних розмірностей світу Конвея

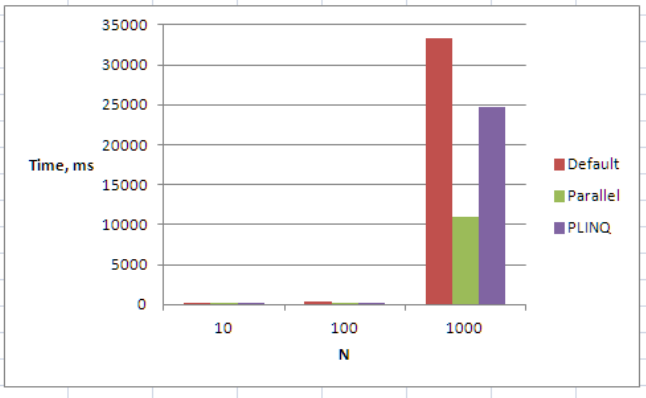
S3

| N | Default | Parallel | PLINQ |
|---|---|---|---|
| 10 | 6 | 31 | 153 |
| 100 | 407 | 141 | 166 |
| 1000 | 40113 | 11478 | 28977 |




S4

| N | Default | Parallel | PLINQ |
|---|---|---|---|
| 10 | 7 | 70 | 185 |
| 100 | 337 | 129 | 192 |
| 1000 | 33327 | 11048 | 24772 |




S5

| N | Default | Parallel | PLINQ |
|---|---|---|---|
| 10 | 3 | 636 | 522 |
| 100 | 83 | 101 | 118 |
| 1000 | 8847 | 3078 | 18467 |

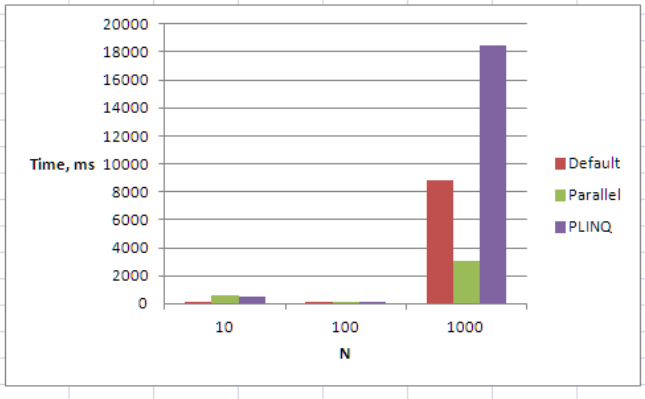
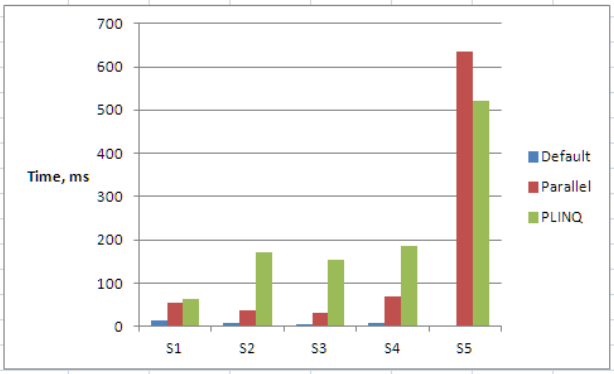



Рисунок 3.28 — Системи S3- S5 - час виконання обчислень для різних розмірностей світу Конвея

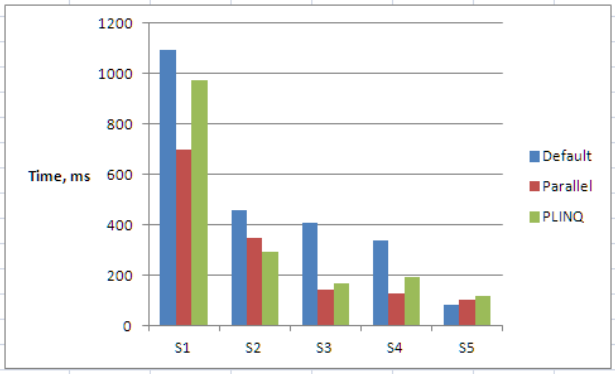
N = 10

| S | Default | Parallel | PLINQ |
|---|---|---|---|
| S1 | 13 | 55 | 62 |
| S2 | 8 | 37 | 171 |
| S3 | 6 | 31 | 153 |
| S4 | 7 | 70 | 185 |
| S5 | 3 | 636 | 522 |




N = 100

| S | Default | Parallel | PLINQ |
|---|---|---|---|
| S1 | 1096 | 700 | 975 |
| S2 | 458 | 349 | 294 |
| S3 | 407 | 141 | 166 |
| S4 | 337 | 129 | 192 |
| S5 | 83 | 101 | 118 |




N = 1000

| S | Default | Parallel | PLINQ |
|---|---|---|---|
| S1 | 118738 | 67060 | 129440 |
| S2 | 44549 | 24249 | 44438 |
| S3 | 40113 | 11478 | 28977 |
| S4 | 33327 | 11048 | 24772 |
| S5 | 8847 | 3078 | 18467 |

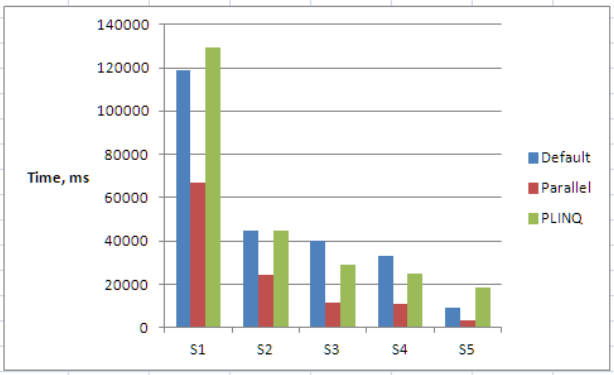



Рисунок 3.29 — Ефективність систем при грі в «Життя» з різними N

При малих N~10 найбільш ефективним підходом є послідовна обробка комірок без розпаралелювання. Цікаво, що розпаралелювання при малих N дає тим гірші результати, чим краще машина, тобто. є контрпродуктивним.

Для середніх N~100 розпаралелювання стає вже ефективним. TPL і PLINQ дають приблизно однакові результати, з невеликою перевагою TPL. Тепер кількість ядер відіграє позитивну роль: що більше, то краще.

Для великих N~1000 розпаралелювання PLINQ втрачає ефективність. Очевидну перевагу має бібліотека TPL. Відзначимо також суттєву різницю у швидкості для збірок, зроблених для 4-ядерного ПК у VS2015 та VS2017– у 3,8 разів для послідовного процесу; 3,6 рази для TPL та 1,3 рази для PLINQ.

Таким чином, при вирішенні цього завдання необхідно враховувати порядок розрахунків N, а також прагнути використовувати якомога новіші із доступних інструментів програмування.

**Висновки до розділу 3**

Враховуючи основне призначення даного програмного компексу – стати зручним підручним інструментом для вивчення основ паралельних обчислень, - ми вибрали для демонстрації задачу елементарного рівня – пошук простих чисел (реалізувавши два алгоритми: решето Ератосфена та решето Аткіна), та завдання середнього рівня – клітинний автомат «Життя» Конвея.

Для кожного з цих завдань представлено кілька способів розв'язання: «звичайні» послідовні обчислення, розпаралелювання за допомогою інструментів бібліотеки TPL та за допомогою PLINQ. Здійснюються виміри часу виконання залежно від максимальної кількості при пошуку простих чисел або від розмірності ігрового поля в клітинному автоматі. Тим самим студент отримує можливість провести емпіричні дослідження ефективності представлених рішень для різних вхідних даних та різних конфігурацій ПК та зробити відповідні висновки.

Відповідні виміри, проведені в даному дослідженні показали наступне.

Для пошуку простих чисел за допомогою решета Ератосфена при великих $N \sim 10^7$ найбільш ефективним підходом є низькорівнева синхронізація. Трохи гірший час обчислень із використанням PLINQ. У той самий час, для малих $N \sim 10^4$ найбільш ефективним використання бібліотеки TPL. Таким чином, вибір технології розпаралелювання для цього завдання залежить від порядку N.

Вибір технології розпаралелювання при пошуку за решетом Аткіна також залежить від порядку N. Але, на відміну від решета Ератосфена, для решета Аткіна можна вказати універсальний підхід, що дає відносно добрі результати на всіх конфігураціях ПК – це розпаралелювання з використанням TPL.

Для гри «Життя » при малих N~10 найбільш ефективним підходом є послідовна обробка комірок без розпаралелювання; причому, розпаралелювання при малих N дає тим гірші результати, чим краще машина, тобто. є контрпродуктивним. Для середніх N~100 розпаралелювання стає вже ефективним; TPL і PLINQ дають приблизно однакові результати, з невеликою перевагою TPL; кількість ядер відіграє позитивну роль: що більше, то краще. Для великих N~1000 розпаралелювання PLINQ втрачає ефективність, очевидну перевагу має бібліотека TPL. Відзначимо також суттєву різницю у швидкості для збірок, зроблених для 4-ядерного ПК у VS2015 та VS2017– у 3,8 разів для послідовного процесу; 3,6 рази для TPL та 1,3 рази для PLINQ. Таким чином, для гри «Життя» необхідно враховувати порядок розрахунків N, а також прагнути використовувати якомога новіші із доступних інструментів програмування.

# ВИСНОВКИ

На основі проведеного дослідження можна зробити наступні висновки.

При класифікації обчислювальних систем за М.Фліном основним визначальним архітектурним параметром є взаємодія потоку команд та потоку даних (операндів та результатів). Відповідно, виділяють наступні чотири класи :

- SISD (Single Instruction, Single Data)
- SIMD (Single Instruction, Multiple Data)
- MISD (Multiple Instructions, Single Data)
- MIMD (Multiple Instructions, Multiple Data) .

З розвитком багато поточних систем виникла необхідність більш детальної класифікації, яка була запропонована Джонсоном. Класифікація Джонсона заснована на структурі пам'яті (global - глобальна або distributed - розподілена) і механізмі комунікацій та синхронізації (shared variables - змінні, що розділяються, або message passing - передача повідомлень). Системи GMSV (global-memory-shared-variables) часто називаються також мультипроцесорами з пам'яттю (shared-memory multiprocessors). Системи DMMP (distributed-memory-message-passing) також називають мультикомп'ютерами з розподіленою пам'яттю (distributed-memory multicomputers).

Для визначення ефективних способів організації паралельних обчислень було запропоновано розбивати завдання на такі етапи: аналіз, дизайн, реалізація, налагодження, тестування та налаштування.

Збільшення ступеня ефективності паралелізму (зменшення тимчасових витрат на накладні витрати) досягається такими способами:

- укрупненням одиниць розпаралелювання;
- зменшенням складності алгоритмів генерації паралельних процедур (підпрограм);

- початковою підготовкою пакету різних варіантів вихідних даних;
- розпаралелювання алгоритмів генерації паралельних процедур (підпрограм).

Отже, оцінювати рівень ефективності при виконанні паралельного додатку можна за чотирма напрямами: виділення незалежних частин, визначення інформаційної взаємодії між частинами, масштабування завдань та їх розподілу між процесорами.

Windows – це багатопотокова операційна системою з витісняючою багатозадачністю, тому що кожен потік може бути зупинений у довільний момент часу і замість нього обраний для виконання інший.

Огляд інструментів .NET в цілому та мови C# зокрема показує, що для організації багатопоточності та синхронізації в Windows створені досить потужні бібліотеки, які дозволяють ефективно вирішувати завдання із розробки паралельних додатків.

Актуальність паралельних обчислень зумовила появу безлічі навчальних та інформаційних ресурсів у цій галузі. На наш погляд, використання переваг інтерактивних систем навчання суттєво підвищує ефективність процесу підготовки розробників, а також сприяє їх подальшій освіті та самоосвіті.

Комплекс програм для навчальної системи з методів паралельного програмування, що розробляється, має наступні призначення:

- навчального посібника;
- засобу для демонстрації прикладів паралельних обчислень;
- засобу для тестування студентів і перевірки засвоєння учбового матеріалу.

Користувачами програмного продукту є:

- викладач, котрий може розробляти, доповнювати та редагувати навчальний посібник, тести і відповіді до них;

– студент, котрий вивчає посібник, запускає на виконання демонстраційні приклади, виконує тестові завдання і отримує оцінку за результатами виконання.

Для роботи з програмою користувачу необхідно скопіювати папку дистрибутиву ParallelOptimization на жорсткий диск або будь-який інший носій. Програма працює в ОС Windows, починаючи від 7 та вище.

Із вимог до ПК, обов’язковим для дослідження паралельних обчислень є наявність мінімум 2 ядер у процесорі та відеокарти.

Проект *Parallel Optimization* містить

— 1 клас, що є точкою входу для IDE (Program.cs);

— 5 класів користувача, що є формами і наслідують Windows.Form;

— 7 класів типів користувача.

Всі представлені класи з певними змінними та властивостями разом з розробленими методами дають можливість реалізувати функціональні можливості навчального комплексу згідно із технічним завданням.

Враховуючи основне призначення даного програмного комплексу – стати зручним підручним інструментом для вивчення основ паралельних обчислень, - ми вибрали для демонстрації задачу елементарного рівня – пошук простих чисел (реалізувавши два алгоритми: решето Ератосфена та решето Аткіна), та завдання середнього рівня – клітинний автомат «Життя» Конвея.

Для кожного з цих завдань представлено кілька способів розв'язання: «звичайні» послідовні обчислення, розпаралелювання за допомогою інструментів бібліотеки TPL та за допомогою PLINQ. Здійснюються виміри часу виконання залежно від максимальної кількості при пошуку простих чисел або від розмірності ігрового поля в клітинному автоматі. Тим самим студент отримує можливість провести емпіричні дослідження ефективності представлених рішень для різних вхідних даних та різних конфігурацій ПК та зробити відповідні висновки.

Відповідні виміри, проведені в даному дослідженні показали наступне.

Для пошуку простих чисел за допомогою решета Ератосфена при великих N~$10^7$ найбільш ефективним підходом є низькорівнева синхронізація. Трохи гірший час обчислень із використанням PLINQ. У той самий час, для малих N ~ $10^4$ найбільш ефективним використання бібліотеки TPL. Таким чином, вибір технології розпаралелювання для цього завдання залежить від порядку N.

Вибір технології розпаралелювання при пошуку за решетом Аткіна також залежить від порядку N. Але, на відміну від решета Ератосфена, для решета Аткіна можна вказати універсальний підхід, що дає відносно добрі результати на всіх конфігураціях ПК – це розпаралелювання з використанням TPL.

Для гри «Життя » при малих N~10 найбільш ефективним підходом є послідовна обробка комірок без розпаралелювання; причому, розпаралелювання при малих N дає тим гірші результати, чим краще машина, тобто. є контрпродуктивним. Для середніх N~100 розпаралелювання стає вже ефективним; TPL і PLINQ дають приблизно однакові результати, з невеликою перевагою TPL; кількість ядер відіграє позитивну роль: що більше, то краще. Для великих N~1000 розпаралелювання PLINQ втрачає ефективність, очевидну перевагу має бібліотека TPL. Відзначимо також суттєву різницю у швидкості для збірок, зроблених для 4-ядерного ПК у VS2015 та VS2017– у 3,8 разів для послідовного процесу; 3,6 рази для TPL та 1,3 рази для PLINQ. Таким чином, для гри «Життя» необхідно враховувати порядок розрахунків N, а також прагнути використовувати якомога новіші із доступних інструментів програмування.

Ми вважаємо, що узагальнення отриманого практичного досвіду буде корисно для більш глибокого розуміння сутності паралельних обчислень, їх потреб та можливостей, а також полегшить перехід до вирішення більш складних завдань.

У роботі докладно розглянуто предметну область, функціональне призначення, вимоги до технічного та програмного забезпечення, проаналізовано структуру програмного продукту, протестовано можливості застосування додатку для проведення дослідів із ефективності розпаралелювання процесів. Усі завдання, поставлені перед програмою, реалізовані.

На наш погляд, проведене дослідження буде корисним для науковців, розробників, аспірантів та студентів, які спеціалізуються або цікавляться проблематикою паралельних обчислень.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Албахари Д. C# 5.0. Справочник. Полное описание языка. - М.: ООО "И.Д. Вильямс", 2014. - 1008 с.

2. Богачёв К.Ю. Основы параллельного программирования. - Бином. Лаборатория знаний, 2010 г. - 344 с.

3. Гергель В.П. Высокопроизводительные вычисления для многоядерных многопроцессорных систем. Учебное пособие – Нижний Новгород; Изд-во ННГУ им. Н.И.Лобачевского, 2010. – 421 с.

4. Голдштейн С., Зурбалев Д., Флатов И. и др. Оптимизация приложений на платформе .NET. - М.: ДМК Пресс, 2014. — 522 с.

5. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.0 на языке C#. 3-е изд. - СПб.: Питер, 2012.- 928 с.:

6. Сысоев А.В., Мееров И.Б., Сиднев А.А. Средства разработки параллельных программ для систем с общей памятью. Библиотека Intel Threading Building Blocks. Учеб. метод. пособие. — Н. Новгород: Изд-во ННГУ, 2007. — 86 с.

7. Уильямс, Энтони. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. – М.: ДМК Пресс, 2012. – 672 с.

8. Adamatzky A. (ed.) Game of Life Cellular Automata. - Springer, 2010. -184 pp.

9. Berman, Kenneth A.; Paul, Jerome L. Algorithms: Sequential, Parallel, and Distributed. – Thomson Course Technology, Boston, Massachusetts, 2005. – 992 pages

10. Blewett R., Clymer A. Pro Asynchronous Programming with .NET. - Apress, 2013. — 352 p.

11. Campbell Colin, Miller Ade. Parallel Programming with Microsoft Visual C++. – Microsoft Corporation, 2011. – 195 pages

12. Cleary S. Concurrency in C# Cookbook. - O'Reilly Media, 2014. — 208 p.

13. Duffy, Joe. Concurrent Programming on Windows. – Addison-Wesley, Boston, 2009. – 990 pages

14. Freeman, Adam. Pro .NET 4 Parallel Programming in C#. – Apress, New York, 2010. – 329 pages

15. Freeman, Bryan. .NET 4.5 Parallel Extensions Cookbook. - Published by Packt Publishing Ltd., UK. – 320 p.

16. Gebali F. Algorithms and parallel computing. - Wiley series on parallel and distributed computing. Wiley & Sons, Inc., 2011. - 365 pages

17. Herlihy Maurice, Kozlov Dmitry, Rajsbaum Sergio. Distributed computing through combinatorial topology. - Elsevier Inc., Waltham, 2014. – 310 pages

18. Hillar, Gastón C. Professional Parallel Programming with C#. - Wiley Publishing, Inc., Indianapolis, Indiana, 2011. – 547 pages

19. Marshall, Donis. Parallel Programming with Microsoft® Visual Studio® 2010 Step by Step. - O'Reilly Media, Inc., 2011. - 249 p.

20. McCool M., Robison A.D., Reinders J. Structured Parallel Programming: Patterns for Efficient Computation. - Morgan Kaufmann, 2012.- 433 pages.

21. Midkiff S.P. Automatic Parallelization. An Overview of Fundamental Compiler Techniques. - Morgan & Claypool, 2012. -170 pp.

22. Miller R., Boxer L. Algorithms Sequential & Parallel: A Unified Approach. - Cengage Learning, 2012. – 448 p.

23. Padua, David (Ed.) Encyclopedia of Parallel Computing. – Springer, New York, Dordrecht, Heidelberg, London, 2011. – 2196 pages

24. Rahman M. C# Deconstructed. - Apress, 2014. — 172 p.

25. Rauber T., Rünger G. Parallel Programming: for Multicore and Cluster Systems. - 2nd Edition. — Springer, 2013. — 522 p.

26. Ringler R. C# Multithreaded and Parallel Programming. - PACKT Published, 2015. - 344p.

27. C# Programming Guide. - [Электронный ресурс] https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx

# ДОДАТКИ

Додаток А

Клас для пошуку простих чисел за допомогою решета Ератосфена

```
class Eratosthenes
    {
        public List<long> Basic(long n)
        {
            var basic = new List<long>();

            for (var i = 2; i <= n; ++i)
            {
                var prime = true;
                for (var j = 0; j < basic.Count && prime; ++j)
                    prime = i % basic.ElementAt(j) != 0;

                if (prime)
                    basic.Add(i);
            }
            return basic;
        }

        public List<long> Modified(long n)
        {
            var sqrt = (int)Math.Sqrt(n);
            var basic = Basic(sqrt);

            var primes = new List<long>(basic);

            for (int i = sqrt + 1; i <= n; ++i)
            {
                var prime = true;
                for (int j = 0; j < basic.Count() && prime; ++j)
                    prime = i % basic.ElementAt(j) != 0;

                if (prime)
                    primes.Add(i);
            }

            return primes;
        }

        public List<long> ModifiedParallel(long n)
        {
            var sqrt = (long)Math.Round(Math.Sqrt(n));
            var basic = Basic(sqrt);

            var primes = new List<long>(basic);
            var spinlock = new SpinLock();
            Parallel.For(sqrt + 1, n, (x, loopState) =>
            {
                var prime = true;
                for (int j = 0; j < basic.Count() && prime; j++)
                    prime = x % basic.ElementAt(j) != 0;

                if (prime)
                {
                    var lockTaken = false;
                    try
                    {
                        spinlock.Enter(ref lockTaken);
                        primes.Add(x);
```

```
            }
            finally
            {
                if (lockTaken) spinlock.Exit(false);
            }
        }
    });
    return primes;
}

public List<long> ModifiedTask(long n)
{
    var sqrt = (long)Math.Round(Math.Sqrt(n));
    var basic = Basic(sqrt);

    var primes = new List<long>(basic);
    var tasks = new List<Task>();
    var spinlock = new SpinLock();
    for (long x = sqrt + 1; x <= n; x++)
    {
        var y = x;
        var task = Task.Factory.StartNew(() =>
        {
            var prime = true;
            for (int j = 0; j < basic.Count() && prime; j++)
                prime = y % basic.ElementAt(j) != 0;

            if (prime)
            {
                var lockTaken = false;
                try
                {
                    spinlock.Enter(ref lockTaken);
                    primes.Add(y);
                }
                finally
                {
                    if (lockTaken) spinlock.Exit(false);
                }
            }
        });
        tasks.Add(task);
    }

    Task.WaitAll(tasks.ToArray());
    return primes;
}

public List<long> ModifiedPLinq(long n)
{
    var sqrt = (long)Math.Round(Math.Sqrt(n));
    var basic = Basic(sqrt);

    var primes = new List<long>(basic);
    var aftersqrt = new List<long>();

    for (var i = sqrt + 1; i <= n; i++)
    {
        aftersqrt.Add(i);
    }

    var other = aftersqrt.AsParallel().Where(x =>
    {
        var prime = true;
```

```
            for (int j = 0; j < basic.Count() && prime; j++)
                prime = x % basic.ElementAt(j) != 0;
            return prime;
        });
        primes.AddRange(other);
        return primes;
    }
```

Додаток Б

Клас AtkinAlg для реалізації алгоритму решета Аткінса пошуку простих чисел

```
public class AtkinAlg : AAtkin
    {
        public AtkinAlg(int x)
            : base(x)
        {

        }

        public override void FindPrimes()
        {
            Primes = new bool[_limit + 1];
            double sqrt = Math.Sqrt(_limit);
            var limit = (ulong)_limit;
            for (ulong x = 1; x <= sqrt; x++)
                for (ulong y = 1; y <= sqrt; y++)
                {
                    ulong x2 = x * x;
                    ulong y2 = y * y;
                    ulong n = 4 * x2 + y2;
                    if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                        Primes[n] ^= true;

                    n -= x2;
                    if (n <= limit && n % 12 == 7)
                        Primes[n] ^= true;

                    n -= 2 * y2;
                    if (x > y && n <= limit && n % 12 == 11)
                        Primes[n] ^= true;
                }

            for (ulong n = 5; n <= sqrt; n += 2)
                if (Primes[n])
                {
                    ulong s = n * n;
                    for (ulong k = s; k <= limit; k += s)
                        Primes[k] = false;
                }
            Primes[2] = true;
            Primes[3] = true;
        }


        public void FindPrimesParallel()
        {
            Primes = new bool[_limit + 1];
            var sqrt = Math.Ceiling(Math.Sqrt(_limit));
            var limit = (ulong)_limit;
            Parallel.For(1, (long)sqrt,
            (x) =>
            {
                Parallel.For(1, (long)sqrt, (y) =>
                {
                    ulong x2 = (ulong)(x * x);
                    ulong y2 = (ulong)(y * y);
                    ulong n = 4 * x2 + y2;
                    if (n <= limit && (n % 12 == 1 || n % 12 == 5))
```

```
                    Primes[n] ^= true;

                n -= x2;
                if (n <= limit && n % 12 == 7)
                    Primes[n] ^= true;

                n -= 2 * y2;
                if (x > y && n <= limit && n % 12 == 11)
                    Primes[n] ^= true;
            });
        });
        for (ulong n = 5; n <= sqrt; n += 2)
            if (Primes[n])
            {
                ulong s = n * n;
                for (ulong k = s; k <= limit; k += s)
                    Primes[k] = false;
            }
        Primes[2] = true;
        Primes[3] = true;
    }

    public void FindPrimesPlinq()
    {
        Primes = new bool[_limit + 1];
        double sqrt = Math.Sqrt(_limit);
        var limit = (ulong)_limit;
        var a = new List<ulong>((int)Math.Ceiling(sqrt));
        for (ulong i = 1; i <= sqrt; i++)
        {
            a.Add(i);
        }

        a.AsParallel().ForAll(x =>
        {
            a.AsParallel().ForAll(y =>
            {
                ulong x2 = x * x;
                ulong y2 = y * y;
                ulong n = 4 * x2 + y2;
                if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                    Primes[n] ^= true;

                n -= x2;
                if (n <= limit && n % 12 == 7)
                    Primes[n] ^= true;

                n -= 2 * y2;
                if (x > y && n <= limit && n % 12 == 11)
                    Primes[n] ^= true;
            });
        });

        for (ulong n = 5; n <= sqrt; n += 2)
            if (Primes[n])
            {
                ulong s = n * n;
                for (ulong k = s; k <= limit; k += s)
                    Primes[k] = false;
            }
        Primes[2] = true;
        Primes[3] = true;
    }
}
```

Додаток В

## Клас Game (“Життя” Конвея)

```
public class Game
    {
        private bool[,] world;
        private bool[,] updated;
        int size;

        public Game(int size)
        {
            if (size < 0) throw new ArgumentOutOfRangeException("Size must be greater than
0");
            this.size = size;
            world = new bool[size, size];
            updated = new bool[size, size];
        }

        public bool this[int x, int y]
        {
            get { return world[x, y]; }
            set { world[x, y] = value; }
        }

        public void UpdateParallel()
        {
            Parallel.For(0, size, x =>
            {
                Parallel.For(0, size, y =>
                {
                    UpdateCell(x, y);
                });
            });
            UpdateWorld();
        }

        public void UpdateTask()
        {
            var tasks = new List<Task>();
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    var cell = new Tuple<int, int>(i, j);
                    var task = Task.Factory.StartNew(() =>
                    {
                        UpdateCell(cell.Item1, cell.Item2);
                    });
                    tasks.Add(task);
                }
            }
            UpdateWorld();
        }

        public void UpdatePLinq()
        {
            var cells = new List<Tuple<int, int>>();
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    cells.Add(new Tuple<int, int>(i, j));
                }
            }
            cells.AsParallel().ForAll(x => UpdateCell(x.Item1, x.Item2));
            UpdateWorld();
        }
```

```
        void UpdateWorld()
        {
            for (int x = 0; x < size; x++)
            {
                for (int y = 0; y < size; y++)
                {
                    world[x, y] = updated[x, y];
                }
            }
        }

        public void Update()
        {
            for (int x = 0; x < size; x++)
            {
                for (int y = 0; y < size; y++)
                {
                    UpdateCell(x, y);
                }
            }
            UpdateWorld();
        }

        void UpdateCell(int x, int y)
        {
            var neighbors = HaveNeighbor(x, y, -1, 0)
                          + HaveNeighbor(x, y, -1, 1)
                          + HaveNeighbor(x, y, 0, 1)
                          + HaveNeighbor(x, y, 1, 1)
                          + HaveNeighbor(x, y, 1, 0)
                          + HaveNeighbor(x, y, 1, -1)
                          + HaveNeighbor(x, y, 0, -1)
                          + HaveNeighbor(x, y, -1, -1);

            var shouldLive = false;

            var isAlive = world[x, y];

            if (isAlive && (neighbors == 2 || neighbors == 3))
            {
                shouldLive = true;
            }
            else if (!isAlive && neighbors == 3)
            {
                shouldLive = true;
            }
            updated[x, y] = shouldLive;
        }

        private int HaveNeighbor(int x, int y, int offsetx, int offsety)
        {
            var alive = 0;

            var proposedOffsetX = x + offsetx;
            var proposedOffsetY = y + offsety;
            var outOfBounds = proposedOffsetX < 0 || proposedOffsetX >= size | proposedOffsetY
< 0 || proposedOffsetY >= size;
            if (!outOfBounds)
            {
                alive = world[x + offsetx, y + offsety] ? 1 : 0;
            }
            return alive;
        }
```

Додаток Г

## Шифрування та дешифрування за шифром Цезаря

```
public class Ceasar
    {
        private const string alphabet =

"01abcdefghijklmnopqrstuvwxyz23ABCDEFGHIJKLMNOPQRSTUVWXYZ45абвгдеёжзийклмнопрстуфхцчшщъыь
эюя67АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ89";
        private const int crypto = 17;

        private static int GetPos(char c)
        {
            for (int i = 0; i < alphabet.Length; i++)
            {
                if (alphabet[i] == c)
                    return i;
            }
            return -1;
        }

        public static string[] Encrypt(string[] content)
        {
            string[] encoded = new string[content.Length];

            for (int i = 0; i < content.Length; i++)
            {
                encoded[i] = string.Empty;
                for (int j = 0; j < content[i].Length; j++)
                {
                    int pos = GetPos(content[i][j]);
                    if (pos != -1)
                        encoded[i] += alphabet[(pos + crypto) % alphabet.Length];
                    else
                        encoded[i] += content[i][j];
                }
            }
            return encoded;
        }

        public static string[] Decrypt(string[] content)
        {
            string[] decoded = new string[content.Length];

            for (int i = 0; i < content.Length; i++)
            {
                decoded[i] = string.Empty;
                for (int j = 0; j < content[i].Length; j++)
                {
                    int pos = GetPos(content[i][j]);
                    if (pos != -1)
                        decoded[i] += alphabet[(alphabet.Length + pos - crypto) %
alphabet.Length];
                    else
                        decoded[i] += content[i][j];
                }
            }
            return decoded;
        }
    }
```