**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

# «ЗАСТОСУВАННЯ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ РОЗВ'ЯЗКУ ЗАДАЧІ РОЗКЛАДУ ГУСТИНИ ДИФУЗІЙНОГО ПРОЦЕСУ ЗА ФУНКЦІЯМИ ЕРМІТА»

**Виконав:**
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Лупинос Олександр Сергійович _____

**Науковий керівник**:
кандидат фізико-математичних наук, доцент
Моторна Оксана Віталіївна _____

**Рецензент:**
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

# РЕФЕРАТ

Випускна кваліфікаційна робота за об’ємом складає 43 сторінки, містить 12 рисунків, використано 18 інформаційних джерел.

Розглянуто: паралельний та асинхронний підходи в програмуванні для підвищення продуктивності роботи додатків; інструменти, які надають обрані середовище розробки, мова програмування та веб-фреймворк для розробки програм із застосуванням паралельних обчислень.

Зроблено: реалізовано за допомогою паралельного програмування підвищення продуктивності розв’язку задачі розкладу густини дифузійного процесу за функціями Ерміта.

Ключові слова : ASP.NET Core MVC, Parallel, Concurrency, Asynchronous, C#, Multithreading

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

БД – база даних

LINQ – Langague-Intergrated Query

MVC – паттерн проєктування Model-View-Controller

API – Application Programming Interface

СДР – стохастичне диференціальне рівняння

Веб-фреймворк – набір інструментів для розробки, компіляції та запуску веб-застосунків.

Модель-View-Контролер (MVC) - це архітектурний підхід до розробки програмного забезпечення, який використовує три компоненти: модель, представлення та контролер.

# ВСТУП

На сьогодні більше 5 мільярдів людей користуються інтернетом по всьому світу[1]. Щоденно люди використовують програмне забезпечення, від соціальних мереж до банківських сервісів, яке встановлене на їх мобільні пристрої або персональні комп'ютери.

Ці мобільні пристрої та персональні комп'ютери у переважній більшості працюють на процесорах з кількома і більше ядрами для того, щоб забезпечити швидку та одночасну обробку додатків і їх інформації. Особливо коли мова йде про реалізацію програм, пов'язаних із різними науковими галузями, доволі часто доводиться працювати з даними великого об'єму або складними алгоритмами. У даній роботі мова йтиме про символьні обчислення у стохастиці, яка не є виключенням у цьому аспекті.

Стохастичні диференціальні рівняння (СДР) знайшли своє застосування в різноманітний галузях науки й техніки: фінансових процесах, океанографії, фізиці, медицині й так далі. Вони надають змогу описувати динамічні системи.

Важливою характеристикою СДР є густина перехідної ймовірності. Вона містить майже всю інформації про певну систему. Її зміст полягає в ймовірності перейти з однієї точки в іншу з усіх можливих переходів. Це поняття відрізняється від звичайної густини, де мова йде про ймовірність знаходження точки в певному стані.

У даній роботі розглядається застосування густини до дифузійного процесу, а саме апроксимація цієї густини за допомогою функцій Ерміта. Дану апроксимацію буде розглянуто оглядово, а її доведення та повний опис можна знайни у статті [10]. Для реалізації використовується готовий веб-додаток, що написаний мовою програмування C# на платформі ASP.NET Core MVC. Вона дозволяє розробляти веб-застосунки та веб-сервіси, які працюватимуть на різних операційних системах (Linux, Windows, Android тощо), та має велику бібліотеку вбудованих класів та функцій для використання.

Додаток дозволяє обрахувати розклад густини дифузійного процесу за функціями Ерміта при різних вхідних параметрах. Однак такі обчислення є доволі складними, і виникає потреба в застосуванні технологій паралельного програмування для підвищення продуктивності програми.

Дана робота присвячено пошуку шляхів підвищення продуктивності для програмного застосунку шляхом використання паралельних обчислень та відповідна реалізація.

# РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Метою роботи є застосувати паралельні обчислення при обчисленні розкладу густини перехідної імовірності дифузійного процесу за спеціальними функціями Ерміта.

Для досягнення поставленої мети необхідно:

- Проаналізувати доступні методи оптимізації програми із застосуванням паралельних та/або асинхронних обчислень
- Обрати засоби вимірювання використання комп’ютерних ресурсів програмою
- За допомогою обраних інструментів підвищити продуктивність роботи програми

# РОЗДІЛ 2. СТОХАСТИЧНІ ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ

Стохастика – розділ науки, що вивчає подання розподілу випадкової величини. Стохастичні диференціальні рівняння (СДР) – це диференціальні рівняння, у яких один або більше членів є випадковою величиною (стохастичним процесом). Хоч вони є складним математичним об'єктом, але мають широке застосування у різних галузях науки й техніки: фінансових процесах, океанографії, фізиці, медицині, а також сфері бізнесу [2]. Це пояснюється тим, що СДР надають змогу описувати поведінку динамічних систем з урахуванням випадкових факторів.

## 2.1 Задачі, що ведуть до СДР

### 2.1.1 З теорії електричних кіл

Відомо, що в момент часу $t$ у фіксованій точці електричного коло для заряду $Q(t)$ справджується диференціальне рівняння [1]:

$$LQ''(t) + RQ'(t) + \frac{1}{C}Q(t) = F(t), \qquad (1)$$

де L – індуктивність, R – опір, C – ємність, F(t) – електрорушійна сила в момент часу $t$.

Якщо допустити, що деякі коефіцієнти даного рівняння є випадковими величинами, то можна отримати наступну ситуацію:

$$F(t) = G(t) + \text{шум}, \qquad (2)$$

де точно невідомо поведінку «шуму» (його називають білим шумом), однак відомо його ймовірністний розподіл, а функція $G(t)$ вважається невипадковою.

### 2.1.2 Проста модель росту популяції

Розглянемо наступну модель росту популяції [1]:

$$\frac{dN}{dt} = a(t)N(t)\,, \;\; N(0) = N_0(\text{константа}), \qquad (3)$$

де N(t) – розмір популяції в момент часу t, $a(t)$ – швидкість її росту в момент часу $t$. Може бути така ситуація, що $a(t)$ залежить від якихось невідомих зовнішніх факторів середовища. Тоді записується:

$$a(t) = r(t) + \text{шум}, \quad (4)$$

де аналогічно попередньому прикладу $r(t)$ вважається невипадковим, а щодо «шум» відомий його ймовірнісний розподіл.

### 2.1.3 Задача оптимальної зупинки

Нехай деяка людина має активи або ресурси, які вона має намір продати. Ціна $X_t$ цих активів на відкритому ринку змінюється в часі $t$ згідно з СДР:

$$\frac{dX_t}{dt} = rX_t + \alpha X_t * \text{шум}, \quad (5)$$

Константи $\alpha, r$ відомі, процентна ставка теж. Проблема полягає в тому, коли варто виконати продаж цих активів, щоб було найвигідніше?

Через «шум» людина, знаючи поведінку ціни $X_t$, ніколи не буде впевнена, що вибір продажу в цей момент часу $t$ є найкращим. Тому, шукається *стратегія оптимальної зупинки*, яка в довгостроковій перспективі дасть найкращій результат.

## 2.2 Варіанти подання

### 2.2.1 Фізика

В фізиці СДР часто записують в формі рівняння Ланжевена. Наприклад, систему СДР першого порядку можна записати в вигляді:

$$x_i' = \frac{dx_i}{dt} = f_i(x) + \sum_{m=1}^{n} g_i^m(x)\eta_m(t), \quad (6)$$

де $x = \{x_i | 1 \le i \le k\}$ – набір невідомих, $f_i$ та $g_i$ — довільні функції, а $\eta_m$ - випадкові функції залежні від часу, їх часто називають шумовими членами.

Основним методом розв'язування СДР у фізиці є пошук розв'язку у вигляді густини ймовірності та перетворенням початкового рівняння у рівняння Фоккера-Планка. Рівняння Фоккера-Планка - диференціальне рівняння з частинними похідними без стохастичних членів. Воно визначає часову еволюцію густини ймовірності , також як рівняння Шредінгера визначає залежність хвильової функції системи від часу в квантовій механіці або рівняння дифузії задає часову еволюцію хімічної концентрації. Також розв'язки можна шукати чисельними методами, наприклад за допомогою методу Монте-Карло.

### 2.2.2 Теорія ймовірності

В теорії ймовірності (а також в її застосуваннях, наприклад фінансовій сфері) запис СДР дещо відрізняється від розглянутих вище. Цей запис робить більш наочною дещо незвичну природу випадкової функції від часу $\eta_m$ з фізичного формулювання. Також цей запис використовують в публікаціях з числових методів розв'язування стохастичних диференційних рівнянь. За строгими математичними правилами $\eta_m$ не може бути звичайною функцією, вона має бути узагальненою функцією. Таке формулювання підходить до СДР з більшою точністю і строгістю ніж фізичне.

# РОЗДІЛ 3. ПЕРЕХІДНА ГУСТИНА ЙМОВІРНОСТІ

Під густиною ймовірності (або щільністю неперервної випадкової величини) розуміють функцію, що визначає ймовірнісну міру того, що значення такої випадкової величини буде у певному діапазоні серед множини всіх можливих її значень. Вона використовується для опису різноманітних процесів. У даній роботі мова йде про обчислення густини ймовірності дифузійного процесу.

## 3.1 СДР дифузійного процесу

Явище дифузії - взаємне проникнення одна в одну дотичних речовин внаслідок руху їхніх частинок (атомів, молекул, іонів, електронів, а також квазічастинок – у конденсованому середовищі). Цей процес може відбуватися як у рідинах, так і в газах та твердих тілах.

Одним з видів даного процесу являється броунівський рух, що є хаотичним рухом молекул у рідинах, та розчинах.

У статті [10] наводиться СДР для дифузійного процесу $X$:

$$\mathrm{d}X_t = a(X_t)dt + \sigma(X_t)\mathrm{d}W_t, \qquad (6)$$

де W – Броунівський рух ($W \in R^m$), та $a: R^d \to R^d$ та $\sigma: R^m \to R^{d\times m}$ – коефіцієнти.

Також у даній статті за допомогою функцій Ерміта наводиться апроксимація (Hermit Function Approximation – HFA) закону розподілу $X$ рекурсивним чином. Точність такого способу складає $O\left(t^{(N+1)/2}\right), N = 0, 1, 2, \ldots$ - «ітерації» апроксимації. Таким чином, рівняння отримає форму:

$$X_t^{N,x} = X_t^x + U_t^N(x, X_t^x), \quad (7)$$

де $U_t^N \in R^m$ .

## 3.2 Функції та поліноми Ерміта

Як було сказано вище, у процесі апроксимації застосовувалася побудова поліномів Ерміта.

Для коефіцієнтів $a \in R^d$ та $b \in R^{d \times d}$, де $b = \sigma^2$ та $a$ – коефіцієнти СДР дифузійного процесу (6), функція Ерміта нульового порядку має наступний вигляд:

$$\Upsilon_t(a, b; x, y) = (2\pi t)^{-d/2} (\det b)^{1/2} \exp\left( -\frac{1}{2t} (b^{-1}(y - x - at), y - x - at) \right), \quad (8)$$

Для функцій вищого порядку вираз ускладнюється: для будь-якого вектора $(i_1, \dots, i_n) \subset \{1, \dots, d\}$,

$$\Upsilon_t^{(i_1,\dots,i_n)}(a, b; x, y) = \partial_{xi_1} \dots \partial_{xi_n} \Upsilon_t(a, b; x, y), \quad (9)$$

Відповідні поліноми Ерміта визначаються співвідношенням

$$\Upsilon_t^{(i_1,\dots,i_n)}(a, b; x, y) = H_t^{(i_1,\dots,i_n)}(a, b; x, y)\, \Upsilon_t(a, b; x, y), \quad (10)$$

Якщо коефіцієнти СДР – $a$ та $\sigma$, при чому $b(x) = \sigma(x)\sigma(x)^*$ та $b(x) > 0$, то отримується наступний вигляд функцій

$$\Upsilon_t(x, y) = \Upsilon(a(x), b(x); x, y),$$
$$\Upsilon_t^{(i_1,\dots,i_n)}(x, y) = \Upsilon^{(i_1,\dots,i_n)}(a(x), b(x); x, y),$$
$$H_t^{(i_1,\dots,i_n)}(x, y) = H_t^{(i_1,\dots,i_n)}(a(x), b(x); x, y). \quad (11)$$

## 3.3 Формула Ньютона-Жирарда

У процесі апроксимації розподілу $X$ також використовується дана формула Ньютона-Жирарда [11], яка надає можливість записати будь-який елементарний симетричний поліном як суму добутків експоненційних сум симетричних поліномів та елементарних симетричних поліномів меншого порядку. Вона дозволяє записати один складний многочлен у вигляді суми інших.

## 3.4 Апроксимація СДР дифузійного процесу

З використанням наведеної вище інформації у теоремі 4.1 статті [10] наводиться наступна апроксимація для процесу $X$.

Нехай при коефіцієнтах $a$ та $b = \sigma^2$, виконується $N \geq 1$ та наступні умови:

- $a \in C^N$ та $b \in C^{N+1}$;
- $a$ і $b$ мають всі свої похідні обмеженими (англ. Bounded);
- $b$ також обмежена та рівномірно еліптична.

Тоді існує унікальна функція Ерміта HFA $p_t^N(x,y)$ порядку $N/2$, що дозволяє апроксимувати дифузійний процес $X$ з точністю $O\left(t^{(N+1)/2}\right)$.

Таким чином, для N = 1 маємо наступні вирази

$$p_t^1(x,y) = \left(1 + t^2 \sum_{i,j,k=1}^{d} c_{ijk}(x)\, H_t^{(i,j,k)}(x,y)\right) \Upsilon_t(x,y),$$

$$c_{ijk}(x) = \frac{1}{4}\sum_{l=1}^{d} \left(b_{ij}(x)\right)' b_{kl}(x), \qquad (12)$$

Якщо $N = 2$ або $d = 1$, тоді такі вирази:

$$p_t^2(x,y) = \big(1 + c(x)t^2 H_t^3(x,y) + d_2(x)t^2 H_t^2(x,y) + d_3(x)t^3 H_t^4(x,y) + d_4(x)t^4 H_t^6(x,y)\big)\, \Upsilon_t(x,y), \qquad (13)$$

де

$$c(x) = \frac{1}{4} b'(x) b(x)$$

$$d_2(x) = \frac{1}{2} a'(x) b(x) + \frac{1}{4} b'(x) a(x) + \frac{1}{8} b''(x) b(x),$$

$$d_3(x) = \frac{1}{12} b''(x) b(x)^2 + \frac{1}{8} \left(b'(x)\right)^2 b(x)$$

$$d_4(x) = \frac{1}{32}\left(b'(x)\right)^2 b(x)^2$$

Можна побачити, що при збільшенні N та/або d, складність рівнянь та відповідно кількість обчислень, які програмі буде потрібно зробити, збільшується помітно.

# РОЗДІЛ 4. ПРОГРАМНИЙ ЗАСТОСУНОК

## 4.1 Опис програми

Для обчислення перехідної густини дифузійного процесу, описаної вище, наявний веб-додаток. Окрім цього застосунок має декілька сторінок, що пов’язані з іншими розрахунками понять у стохастиці. У майбутньому розлядається застосування технологій паралельного програмування та практики, отриманої унаслідок виконання даної роботи, для підвищення продуктивності їх виконання.

Інтерфейс додатку зображено на Рис.4.1.

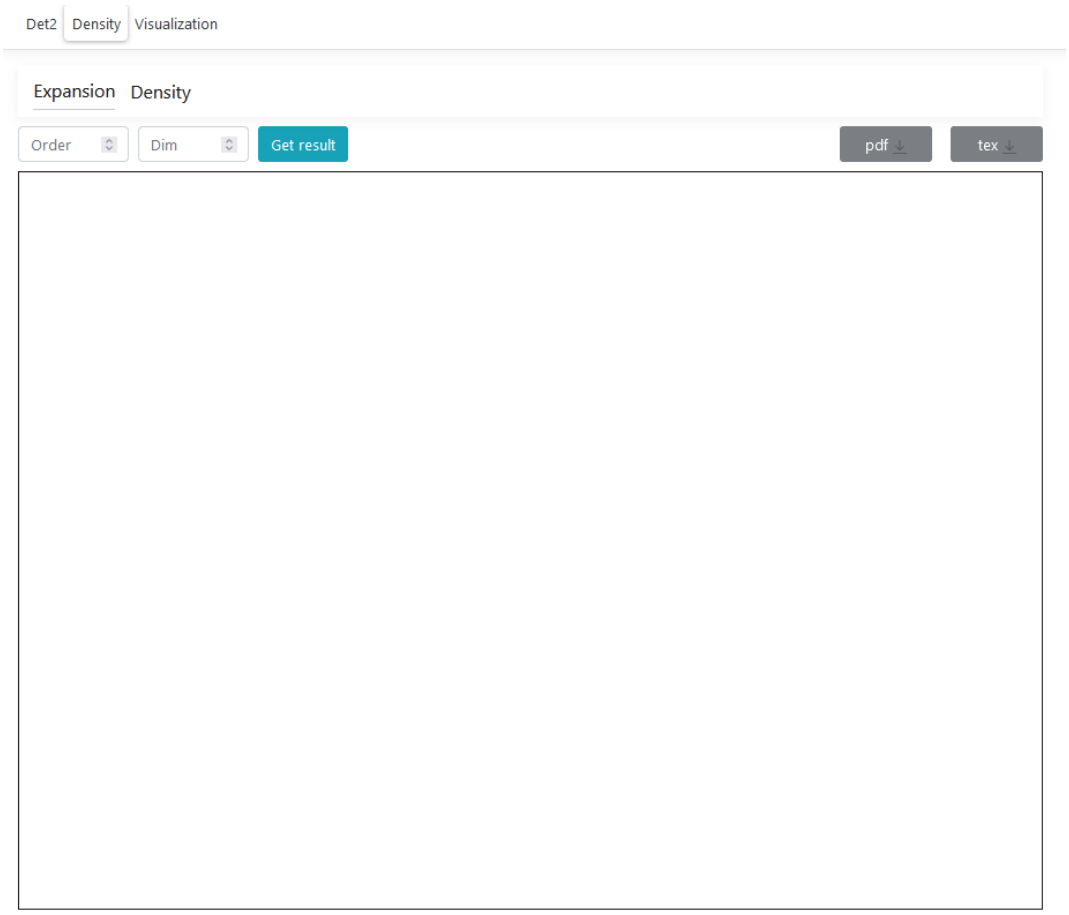


Рис. 4.1 Інтерфейс додатку

Наявний програмний застосунок дозволяє обчислити апроксимацію дифузійного процесу $X$ при різному порядку N (на рисунку - Order) та розмірності d (на рисунку - Dim) і показати вихідну функцію. Після цього

результат можна завантажити у форматі pdf або txt для зберігання на своєму пристрої та подальшого використання.

На Рис. 4.2 зображено приклад результату роботи програми при N = 2 та d = 2. Можна побачити, що вихідна формула займає 11 сторінок. Це свідчиться про те, що обчислення доволі складні. Якщо узяти N = 2 та d = 3, то функція займатиме 150 сторінок. Отже, зрозуміло, що при збільшенні параметрів, розмір результату буде збільшуватися.

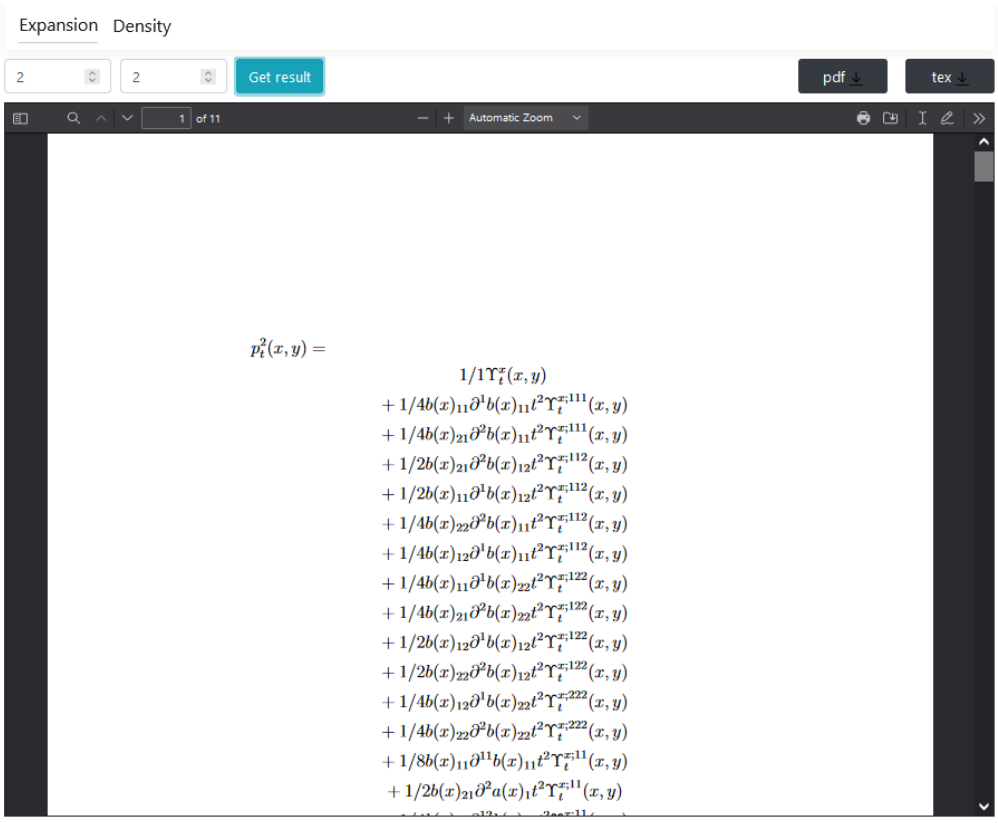


Рис. 4.2 Результат обчислення при N = 2 та d = 2

## 4.2 C# та платформа .NET

Додаток написаний мовою програмування C# на платформі ASP.NET Core MVC - високорівневою мовою, що була розроблена компанією Microsoft для розробки додатків на платформи .NET. Це включає настільні, веб-, ігрові та мобільні додатки.

Основні переваги мови C# наступні:

- Платформонезалежність: програми, написані на C#, можуть працювати на будь-якій платформі, що підтримує .NET Framework або .NET Core.
- Об'єктно-орієнтована: C# є повністю об'єктно-орієнтованою мовою програмування, що дозволяє розробникам легко створювати, розширювати та підтримувати код.
- Висока продуктивність: C# має компілюється під час виконання коду, що дозволяє досягнути високої продуктивності та швидкодії.
- Широкі можливості: C# підтримує багато різних типів даних та функцій, що дозволяє створювати різноманітні програми та додатки.
- Розширюваність: C# має вбудовану підтримку багатьох різних інтерфейсів та бібліотек, що дозволяє розширювати можливості мови та використовувати сторонні розширення.

Недоліки:

- Високий поріг входження: C# є високорівневою мовою програмування, що може виявитися складною для початківців.
- Строга типізація: C# має строгу типізацію, що означає, що розробник повинен дуже точно визначити тип даних для кожної змінної в програмі.

Зважаючи на наведені переваги та недоліки, не дивно, що згідно Coursera[12], C# досі лишається однією з найпопулярніших мов програмування, що свідчить про її значну популярність та широке використання в розробці програмного забезпечення.

Щодо ASP.NET Core MVC - це веб-фреймворк, що є частиною платформи .NET Core. Під «Core» розуміють .NET версії 5 або більше. Цей фреймворк дозволяє розробникам створювати веб-додатки, які використовують архітектуру MVC (Model-View-Controller). Серед них є блоги, соціальні мережі, інтернет-магазини та інші.

Деякі з основних переваг фреймворку ASP.NET Core MVC включають:

- Підтримка кросплатформенності: фреймворк може працювати на Windows, macOS та Linux.
- Підтримка моделювання: фреймворк дозволяє створювати моделі, що представляють дані додатка.
- Підтримка роутінгу: фреймворк надає можливість налаштування маршрутів, які використовуються для відображення сторінок додатка.
- Підтримка розширень: фреймворк дозволяє розробникам додавати розширення до додатку, що дозволяє розширювати його функціональність.
- Відкритий код: фреймворк має відкритий вихідний код, тому його можна вільно розширювати, щоб адаптувати додаток до окремого бізнес-процесу.

Крім того, фреймворк має широкий спектр інструментів для тестування та відладки веб-додатків.

Недоліками ASP.NET Core MVC можуть бути складність налаштування деяких компонентів, потреба в додаткових знаннях та навичках для розробки веб-додатків з використанням фреймворку.

Особливості ASP.NET Core MVC в версії 6 включають покращення швидкодії, зменшення обсягу коду, збільшення продуктивності та покращення розширюваності додатків. Крім того, в ASP.NET Core MVC 6 вперше з'явився новий фреймворк для створення інтерактивних веб-додатків, що базується на SignalR. Цей фреймворк дозволяє забезпечувати миттєву взаємодію з користувачами, включаючи чати, онлайн-ігри та інші типи додатків, що потребують миттєвої взаємодії.

Загалом, ASP.NET Core MVC - це потужний та гнучкий веб-фреймворк, який дозволяє розробникам створювати високоякісні веб-додатки з використанням сучасних технологій.

4.3 Середовище розробки

У ході виконання даної роботи було використано інтегроване середовище розробки (скор. IDE) Visual Studio 2022 для реалізації паралельних обчислень та вимірювання часу роботи програми. Це одне з найпопулярніших середовищ розробки програмного забезпечення, яке має багатий функціонал та засоби, які допомагають підвищити продуктивність розробника.

Згідно з github.io [13], Visual Studio на травень 2023-го року посідає одне з перших місць за кількістю пошуків у системі Google сторінки для завантаження IDE (див. Рис. 4.3).

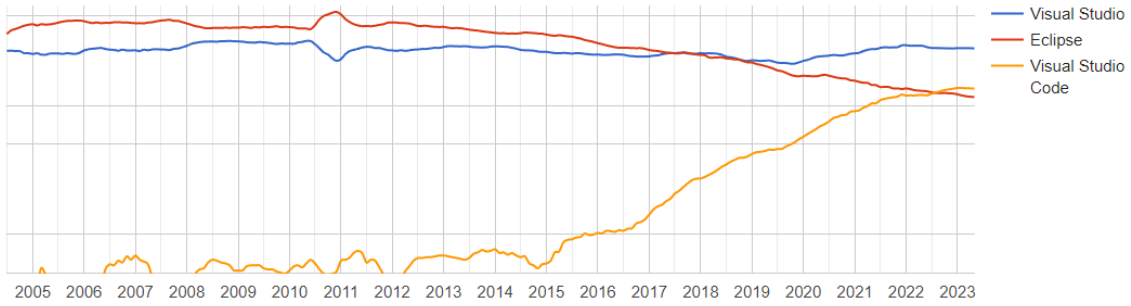


Рис. 4.3 Відносний графік популярності IDE

Основні можливості Visual Studio включають:

- Редактор коду з підсвічуванням синтаксису та автодоповненням.
- Вбудовані інструменти для розробки графічних інтерфейсів, включаючи редактор XAML для розробки програм на платформі .NET.
- Вбудовані інструменти для тестування, включаючи юніт-тестування, функціональне тестування та тестування продуктивності.
- Вбудовані інструменти для тестування, включаючи юніт-тестування, функціональне тестування та тестування продуктивності.
- Засоби для налагодження коду, включаючи можливість встановлення точок зупину та крокування під час налагодження в режимі реального часу.

- Інтегрований засіб для контролю версій (Git), що дозволяє розробникам спільно працювати над проектами.

Visual Studio також має багато додаткових функцій та інструментів, які можуть бути встановлені як додаткові компоненти (англ. назва NuGet packages). Наприклад, можна встановити засоби для розробки веб-додатків, розширення для роботи з базами даних, інструменти для розробки мобільних додатків та інше.

У загальному, Visual Studio є потужним та гнучким середовищем розробки, яке може бути використане для розробки різноманітних програмних продуктів на різних платформах.

### 4.4 Інструменти для вимірювання часу роботи програми

У Visual Studio існує кілька способів вимірювання часу виконання програми у самому коді. Один з найпростіших способів - скористатися стандартною бібліотекою Systems.Diagnostics у C#, яка має клас System.Diagnostics.Stopwatch. За допомогою методів *Start* та *Stop* цього класу запускається та зупиняється таймер. Результат зберігається у властивості *ElapsedMilliseconds* екземпляра класу. Наприклад:

```
using System.Diagnostics;

var stopwatch = new Stopwatch();
stopwatch.Start();

// Код, який потрібно виміряти

stopwatch.Stop();
Console.WriteLine($"Час виконання: {stopwatch.ElapsedMilliseconds} мс");
```

Також Visual Studio також надає можливість вимірювати час виконання програми за допомогою Diagnostics Tool Window. Для цього потрібно відкрити вікно Debug -> Windows -> Show Diagnostic Tools і запустити програму в режимі Debug. У вікні Diagnostics Tool Window буде відображено час виконання програми, а також інші корисні метрики (див. Рис. 4.4).

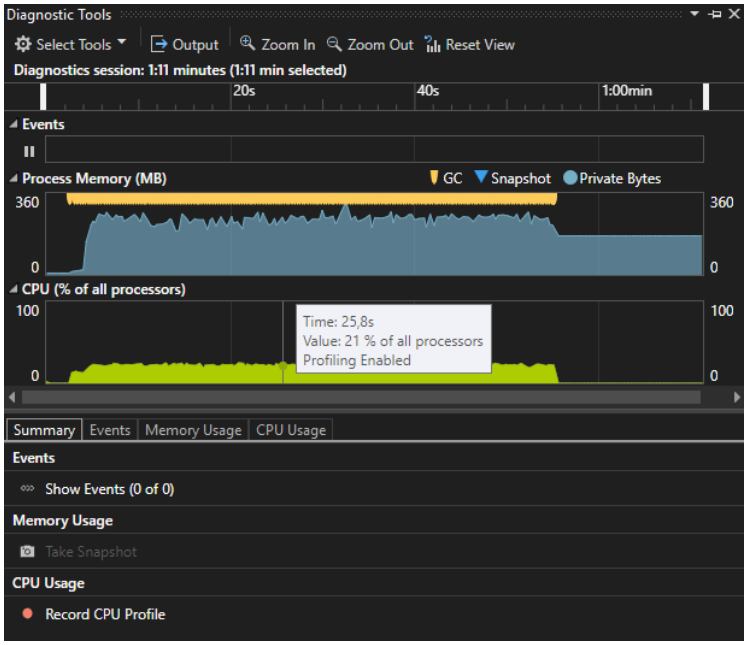


Рис. 4.4 Приклад відображення даних за допомогою Diagnostic Tools

Перевагою даного способу є наявність графічної візуалізації різних метрик та структуризація виміряних даних.

## 4.5 Дані вимірювання програми

За допомогою Stopwatch було зібрано дані про час обчислення перехідної густини ймовірності програмою при різних порядках N та розмірності d. Виміри було зроблено на комп'ютері з процесором Intel Core i7-4700HQ та організовано у Таблиці 1. Варто розуміти, що ці дані приблизні і мають певну незначну похибку. Це пов'язано із завантаженістю процесора у момент виконання програми, різними фізичними чинниками.

*Таблиця 1 – Дані вимірювання виконання програми*

| N | d | Приблизний час обчислення |
|---|---|---|
| 1 | 1 | 3 мс |
| 2 | 1 | 15 мс |
| 2 | 2 | 17 мс |
| 2 | 3 | 1,3 с |
| 2 | 4 | 52,5 с |

| | | |
|---|---|---|
| 3 | 2 | 3,7 с |
| 3 | 3 | 133 хв |

У таблиці можна побачити, що час обчислення помітно збільшується починаючи з d = 4 або N = 3. Це можна також побачити на Рис. 4.5, де зображено вивід функції в додатку при N = 2 та d = 4. За таких параметрів функція займає більше тисячі сторінок при перегляді її у вікні.

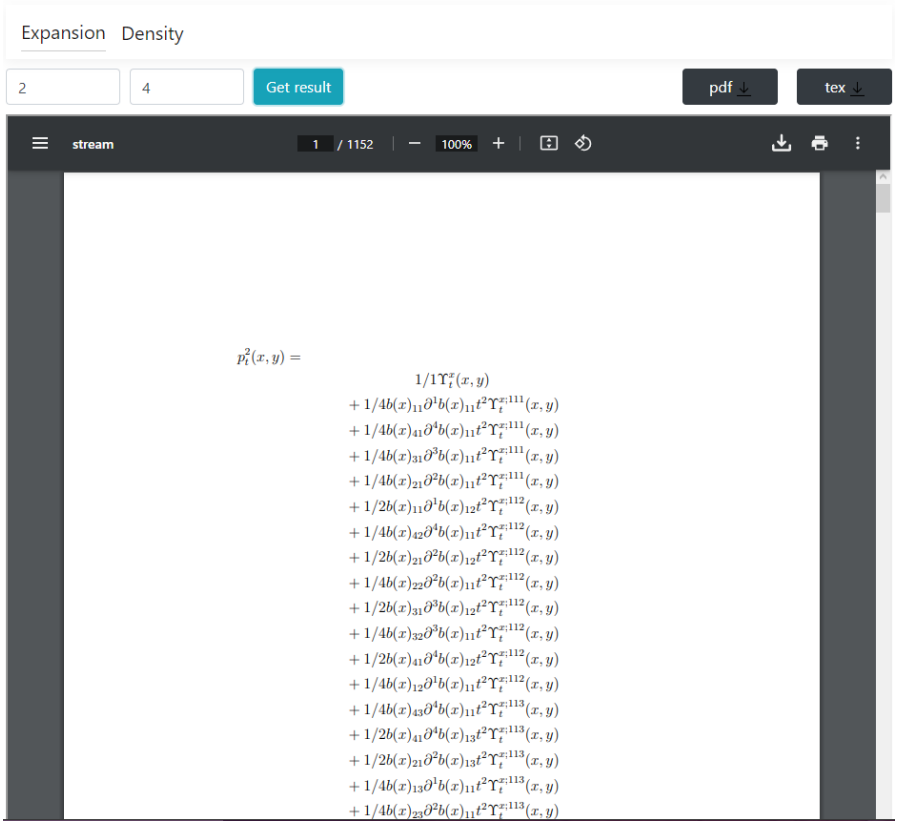


Рис. 4.5 Приклад виводу функції при N = 2 та d = 4

Зрозуміло, що при подальшому збільшенні параметрів, обчислення будуть займати великі проміжки часу. Тому варто розглянути варіанти підвищення продуктивності обчислення даної програми.

Бажаним результатом реалізації даної мети є зменшення часу обчислення в кілька разів при обчисленні порядків N = 2 та d = 3 і більше.

# РОЗДІЛ 5. Паралельне програмування у .NET

Наявний програмний продукт потребує реалізації певних технологій мови C# та платформи .NET для підвищення продуктивності його роботи. Тому наступним буде розглянуто й порівняно варіанти оптимізації програмних застосунків.

Оскільки при обчисленні густини додаток не використовує поняття паралельності чи асинхронності, то код у ній виконується послідовно. Це означає, що він виконується крок за кроком, один метод виконується після іншого. Тому для того, щоб пришвидшити його, потрібно виконувати певні фрагменти паралельно. Платформа .NET надає застосунки для реалізації цього, які буде озглянуто пізніше.

## 5.1 Асинхронність та паралелізм

Поряд із паралельністю розглядають також асинхронність. Асинхронність та паралелізм - це два різні механізми, які дозволяють оптимізувати виконання програм на платформі .NET.

Асинхронність в .NET дозволяє програмам продовжувати виконання певних задач, не чекаючи завершення інших задач, які можуть зайняти багато часу. Це дозволяє програмі займатися іншими корисними діями, поки виконується довгий процес. Класи, які дозволяють реалізувати асинхронність в .NET, включають в себе Async та Await. Асинхронність створює ілюзію паралельності, оскільки відбувається чергування між виконанням процесів, як це показано на рис. 5.1.

Прикладом може слугувати офіціант у ресторані, який подає страви кільком клієнтам за раз.

Якщо перенести даний приклад у програмування, то це означає, що програма може відправити запит до БД, і поки вона чекає на відповідь, почати зчитувати файл, або обчислити певний вираз/функцію. Після того, як дані з БД

прийдуть, програма їх оброблює й відправляє користувачеві, таким чином не витрачаючи час на очікування відповіді, а виконуючи інші операції в цей час.

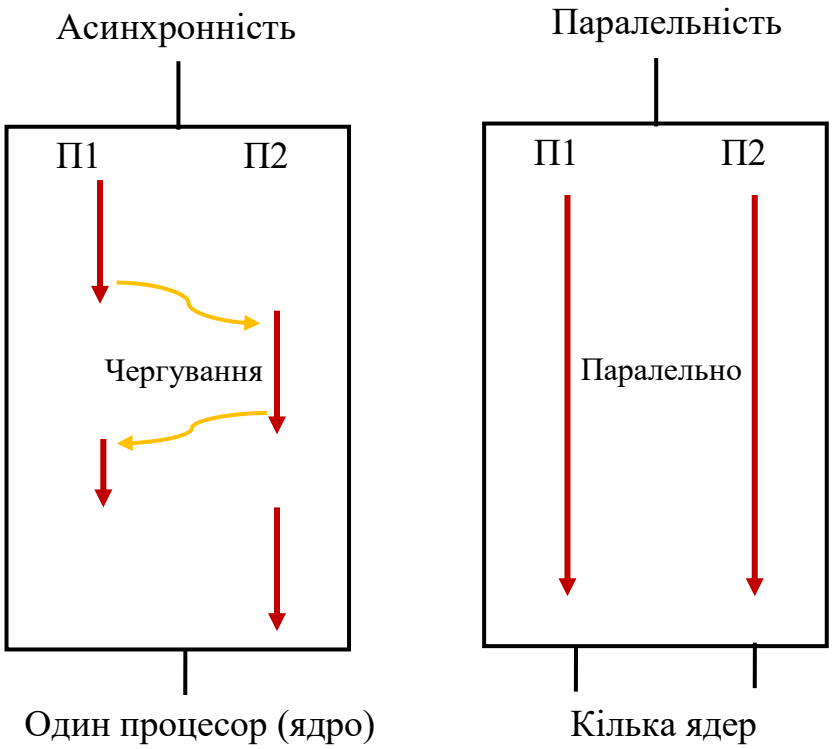


Рис. 5.1 Обробка процесів асинхронним та паралельним чином

Таким чином асинхронність забезпечує неблокуючу роботу програми, тобто поки один процес очікує на відповідь з БД, або запит до API, виконання програми не блокується, а дозволяється почати виконання нового процесу.

*Паралелізм* дозволяє програмі виконувати багато завдань одночасно, здійснюючи розподіл виконання між доступними процесорними ядрами (див. Рис. 5.1). Це дозволяє прискорити виконання програми і підвищити її продуктивність.

Порівняльна характеристика цих двох механізмів підвищення продуктивності програм описана в таблиці 2.

*Таблиця 2 – Порівняння асинхронного програмування із паралельним*

| Асинхронність | Паралелізм |
|---|---|
| Мета підходу в ефективному використанні потоків, запобігаючи їм | У той час, як паралелізм направлений на виконання декількох завдань, |

| | |
|---|---|
| бути заблокованими | розділених між кількома процесорами або ядрами одного процесора |
| Забезпечується чергуванням між потоками в одному ядрі ЦП | Забезпечується одночасним обчисленням на декількох ядер ЦП |
| Підвищує кількість роботи виконаної за проміжок часу | Підвищує обчислювальну спроможність та пропускну здатність програми |
| Налагоджування асинхронних програм є помітно складнішим | У той час, як налагоджування паралельні програми є менш складним |

Отже, оскільки задача полягає в підвищення продуктивності саме в обчисленні, а не в оптимізації виконання мережевих запитів або операцій введення/виведення в додатку, то доцільним є використання паралельного програмування.

Однак, важливо зазначити, що в багатьох випадках асинхронне та паралельне програмування можуть бути використані разом для досягнення максимальної ефективності. Наприклад, використання асинхронних запитів для мережевих операцій може допомогти знизити час очікування на відповідь від сервера, тоді як паралельне виконання обчислювально-інтенсивних задач на кількох ядрах процесора може допомогти знизити час виконання програми.

## 5.2 Потоки та задачі

При реалізації паралельного програмування на платформі .NET так чи інакше створюються потоки, які виконуються паралельно на різних ядрах процесору. Для цього в .NET існують два основних підходи до роботи з багатопоточністю: використання класу Thread і використання класу Task. Task - це більш сучасний підхід, який має кілька переваг над класом Thread:

- Ефективність ресурсів: При використанні класу Thread кожен новий потік вимагає певних ресурсів від операційної системи, що може призвести до перевантаження системи. Task використовує пул потоків

(Thread pool), який створює певну кількість потоків під час запуску додатка, які можуть бути перевикористані для виконання різних задач. Це зменшує кількість потоків, створюваних операційною системою.

- Простота використання: Клас Task дозволяє простіше виконувати асинхронні операції, оскільки він надає ряд методів для роботи з ними, таких як ContinueWith, Wait, WhenAll, WhenAny і т. д. Крім того, при використанні Task можна легко використовувати async/await, що спрощує написання асинхронного коду.
- Інтеграція з планувальником завдань: Використання Task дозволяє інтегруватись з планувальником завдань (.NET Task Scheduler), що дозволяє керувати плануванням та виконанням завдань в контексті певного додатка. Це дозволяє краще керувати потоками та зменшує вплив на ресурси системи.

Узагальнюючи, використання Task є більш ефективним та простим в реалізації способом для виконання асинхронних операцій у .NET, і він рекомендується для використання в більшості сучасних додатків.

## 5.3 Типи паралелізму

У C# розрізняють два типи паралелізму:

1) Паралелізм даних (Data parallelism) - це техніка паралельної обробки даних, коли велику кількість даних розділяють на менші частини та обробляють одночасно на різних процесорах або ядрах. Data parallelism можна виконати в .NET за допомогою Parallel LINQ (PLINQ) та клас Parallel, які надають розробникам можливість створювати паралельні запити до баз даних, паралельно обробляти великі масиви даних, а також паралельно виконувати будь-які інші завдання, що можуть бути розбиті на менші частини. Прикладом даного типу може слугувати фільтрація елементів масиву.
2) Паралелізм завдань (Task parallelism) – це техніка паралельної обробки завдань, коли одночасно виконуються різні завдання в межах одного

додатку. Task parallelism можна виконати в .NET за допомогою вище згаданих завдань (Tasks) із TPL (Task Parallel Library). Прикладом може слугувати відправлення користувачеві одночасно листа на електронну адресу та SMS – це дві різні задачі, які можна виконати паралельно.

Таким чином, використовуючи Data parallelism та Task parallelism в .NET, розробники можуть ефективно використовувати ресурси комп'ютера та забезпечувати високу продуктивність додатків.

## 5.4 Інструменти для імплементації паралельного програмування

Платформа .NET та середовище розробки Visual Studio надають різні застосунки для ефективної розробки паралельних програм (див. офіційну документацію [3]).

### *5.4.1 Task Parallel Library*

Бібліотека TPL (Task Parallel Library) - надає високорівневий інтерфейс для паралельного програмування в .NET. TPL дозволяє створювати та запускати задачі (Tasks), які автоматично розподіляються між доступними ядрами процесора. Для того, щоб мати змогу використовувати функціонал, пов’язаний із задачами, який надає дана бібліотека, потрібно підключити простір імен System.Threading.Tasks. Повний опис методів та класів для керування задачами та їхніми результатами можна знайти за офіційною документацією Microsoft [3].

Основні компоненти TPL:

1) Клас Task та клас планувальника задач (TaskScheduler), про які було згадано вище.
2) TaskCompletionSource - це компонент, який дозволяє створювати задачі, які повертають значення. TaskCompletionSource можна використовувати для створення задач, які можуть бути запущені та відслідковуватися за допомогою методів, таких як Task.WhenAll () та Task.WhenAny ().
3) Parallel - це клас, який імплементує паралелізм даних (згаданий у п. 5.3). Він містить паралельну реалізацію для циклів For() та ForEach(), наявних у C#. Під час виконання даних методів, TPL автоматично розподіляє

колекцію даних на частини, які паралельно оброблятимуться. Цей розподіл виконує планувальник задач, зважаючи на використання системних ресурсів та навантаженість.

### *5.4.2 Parallel LINQ (PLINQ)*

Розширення мови LINQ, яке дозволяє використовувати паралельну обробку даних в LINQ запитах. PLINQ має ті ж самі функції, що і звичайний LINQ, такі як Select, Where, OrderBy, і GroupBy, але виконує їх паралельно. PLINQ використовує TPL для автоматичного розподілу операцій LINQ між доступними процесорами або ядрами.

Одна з основних переваг PLINQ полягає в тому, що можна досить легко перетворити існуючий код на LINQ на паралельний за допомогою методу AsParallel().

### *5.4.3 Concurrent Collections[14]*

Спеціальні колекції, що дозволяють безпечно взаємодіяти зі спільними даними з декількох потоків. Це дозволяють механізми синхронізації, які реалізовані у цих колекціях. Крім того, Concurrent Collections дозволяють отримати доступ до даних у потокобезпечному режимі, що дозволяє уникнути помилок, пов'язаних зі зміною даних з декількох потоків одночасно.

Для того, щоб використовувати дані колекції, потрібно підключити простір імен System.Collections.Concurrent. У ньому наявні наступні класи:

- ConcurrentBag – аналог класу List з простору System.Collections.Generic, який дозволяє багатьом потокам одночасно додавати та вилучати елементи з «мішка» без блокування всього контейнера. Колекція зберігає елементи невпорядковано.
- ConcurrentQueue - це розширення класу Queue, що працює за принципом First-In-First-Out (FIFO). ConcurrentQueue дозволяє багатьом потокам одночасно додавати елементи в чергу та вилучати їх

з неї, також без блокування всього контейнера. Це робить його корисним для обміну даними між потоками.

- ConcurrentStack - це розширення класу Stack, який працює на основі Last-In-First-Out (LIFO) принципу. Дозволяє багатьом потокам одночасно додавати та вилучати елементи зі стеку без блокування всього контейнера.
- BlockingCollection - Надає можливості блокування та обмеження для безпечних для потоків колекцій, які реалізують інтерфейс System.Collections.Concurrent.IProducerConsumerCollection<T>. Відповідно дана колекція імплементує паттерн Виробник-Споживач (див. офіційну документацію Microsoft[15]). BlockingCollection<T> можна використовувати як базовий клас або резервне сховище для забезпечення блокування та обмеження для будь-якого класу колекції, що підтримує IEnumerable<T>.
- ConcurrentDictionary - реалізація класу Dictionary, яка дозволяє багатьом потокам одночасно читати та записувати дані у словник без блокування всього словника. Це дозволяє підтримувати високу продуктивність при використанні словника в багатопотоковому середовищі.

Використання Concurrent Collections дозволяє уникнути затримок та блокувань при роботі зі спільними даними, що покращує продуктивність та швидкість виконання додатку.

Наприклад, якщо потрібно створити багатопотоковий додаток, що має зберігати дані у словнику та читати їх з декількох потоків одночасно, можна скористатися ConcurrentDictionary для забезпечення безпеки при взаємодії зі словником. Кожен потік може одночасно додавати та читати дані зі словника без блокування інших потоків.

Однак, важливо пам'ятати, що використання Concurrent Collections не гарантує правильної роботи багатопотокового додатку без додаткової уваги до

деталей реалізації. Потрібно уникати ситуацій, коли декілька потоків одночасно змінюють або зчитують один і той же елемент колекції, тому що це може призвести до неправильної роботи додатку. Цю ситуацію називають гонкою даних (race conditions).

## 5.5 Інструменти для діагностики та аналізу роботи додатку

Інструменти для вимірювання продуктивності додатку є вкрай важливими для розробників, які хочуть оптимізувати свій код та покращити продуктивність додатків. Visual Studio пропонує ряд інструментів для профілювання (англ. profiling) та діагностики. Вони дозволяють проаналізувати використання пам’яті та процесора, а також інші проблеми на рівні додатку.

Для вимірювання продуктивності під час налагодження додатку, Visual Studio має Diagnostic Tools, описані у п. 4.4. Окрім способу Debug > Windows > Show Diagnostic Tools, даний інструмент можна відкрити натисканням клавіш Ctrl + Alt + F2 на клавіатурі, при чому використовувати його слід у режимі Debug.

Для аналізу продуктивності збірок у конфігурації Release Visual Studio надає інструменти, які знаходяться у Performance Profiler (профілювач продуктивності). Даний застосунок дозволяє не тільки збирати різні метрики, а й формує звіт, який можна зберегти із розширенням .diagsession для повторного використання даних. Відкрити Performance Profiler можна, вибравши Debug > Performance Profiler (або натисканням Alt + F2).

Основні інструменти профілювача продуктивності описані у таблиці 3.

*Таблиця 3. Інструменти Performance Profiler у .NET*

| Назва інструменту | Опис | Напрямок застосування |
|---|---|---|
| CPU Usage | Дозволяє переглянути, які функції викликають найбільше навантаження на процесорі та скільки | Загальні проблеми з продуктивністю, оптимізація використання процесора, затримки у викликах API тощо. |

| | | |
|---|---|---|
| | часу займає їх виконання. | |
| Memory Usage | Показує використання оперативної пам'яті додатком | Використання пам'яті додатком, проблеми із зависання інтерфейсу та можливі витоки пам'яті |
| .NET Object Allocation Tracking | Показує, де розміщуються об'єкти .NET та інформацію про роботу Garbage Collector (скор. GC), вбудованого у .NET | Алокацію та чистка пам'яті, що виконує GC |
| .NET Async | Показує використання async/await у додатку .NET | Дослідження ймовірних проблем з продуктивністю асинхронного коду |
| Database | Дозволяє переглянути продуктивність запитів до бази даних. Доступний лише для ASP.NET Core додатків | Знаходження запитів, які виконуються повільно, та ймовірних шляхів вирішення проблеми |
| Events Viewer | Показує HTTP-запити, лог-повідомлення та винятки при подіях (англ. events) | Затримки у API запитах, дослідження повільної роботи додатку на віддаленому веб-сервері |

Повний список інструментів у Performance Profiler та їх огляд можна знайти у документації [16].

На Рис. 5.2 можна побачити доступні для наявного додатку інструменти Performance Profiler. У даній роботі переважно використовувався застосунок CPU Usage для перегляду використаних ресурсів процесору під час виконання програми, а також ресурсів оперативної пам’яті за допомогою Memory Usage.

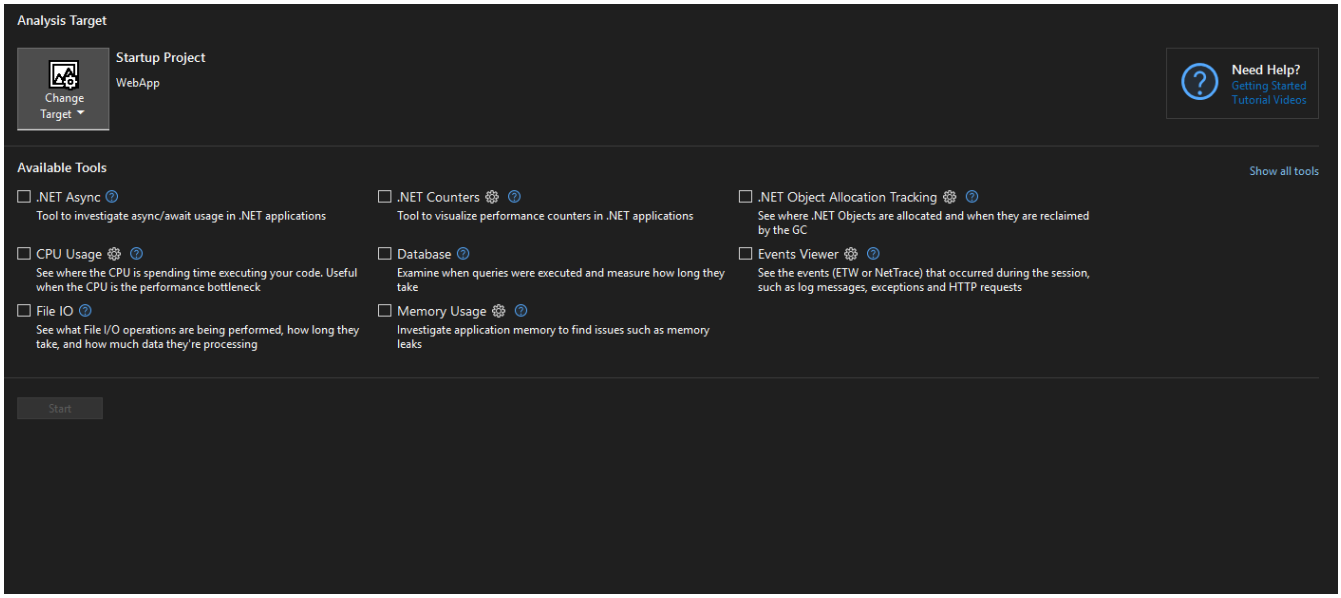


Рис. 5.2 Інструменти Performance Profiler для додатку ASP.NET Core MVC

Приклад використання CPU Usage та Memory Usage зображено на Рис. 5.3. Параметри узято N = 3 та d = 2. Для більш точного вимірювання було зроблено консольну програму, яка виконує тільки обрахунки перехідної густини ймовірності. Це потрібно для того, щоб побачити використання ресурсів саме обчисленням функції. Інакше, вимірювання даних інструментів включали би ресурси процесору та оперативної пам'яті, які споживають інші процеси веб-додатку.

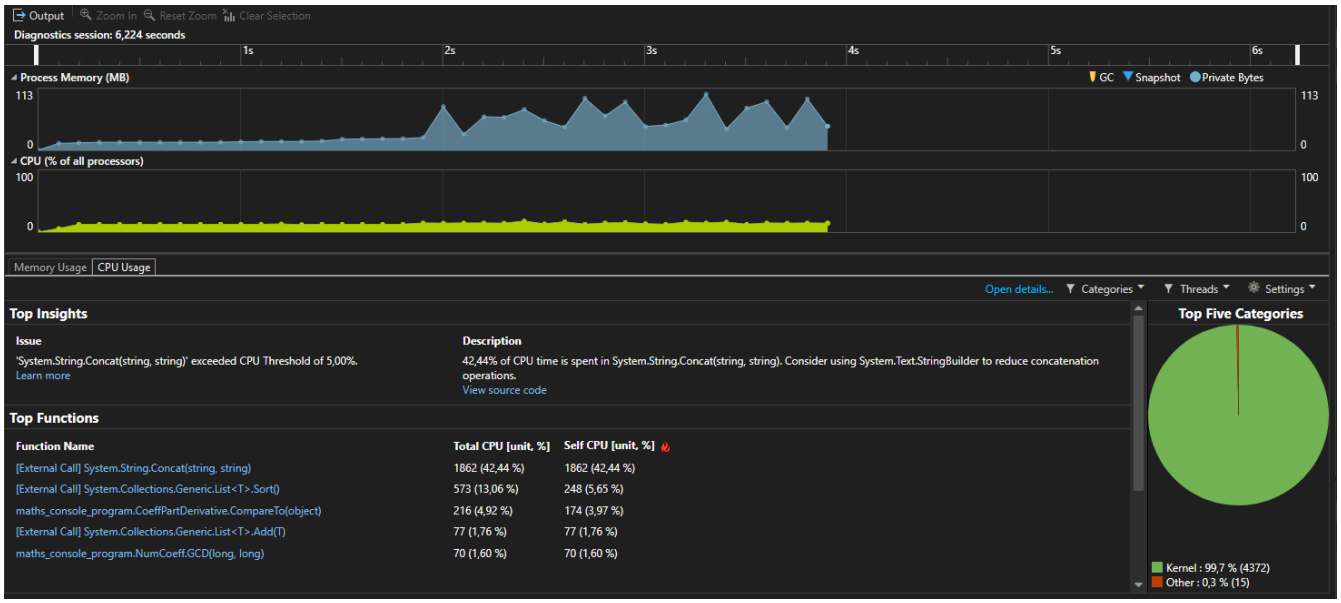


Рис. 5.3 Виміри CPU та Memory Usage при N = 3 та d = 2

Отже, на рисунку можна побачити, як програма при обчисленні використовувала лише одне ядро процесору, а також в основному від 100 до 220 МБ пам’яті.

# РОЗДІЛ 6. Реалізація та її аспекти

## 6.1 Хід роботи

На Рис. 6.1 зображено список класів, які використовуються програмою при обчисленні перехідної густини ймовірності.

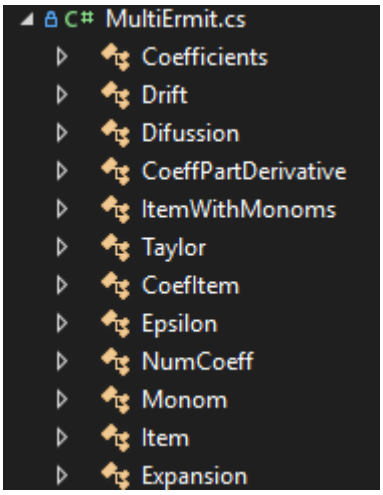


Рис. 6.1 Список класів програми

Серед даних класів основним є Expansion, який використовує усі інші в ході виконання програми.

Програма працює таким чином, що кожен доданок функції обраховується окремо. Відповідно створюється великий масив елементів, який обробляється методами класів. Підтвердженням цьому є вихідні функції, які показані у прикладах роботи додатку.

За допомогою інструменту CPU Usage було визначено методи, які найдовше виконуються. Це конструктор класу Expansion та метод ToString для перетворення масиву даних у формат string та виводу його на екран. На Рис. 6.2 можна побачити, скільки ресурсів споживають дані функції при обчисленні густини з параметрами N = 2 та d = 3.

**Hot Path**

| Function Name | Total CPU [unit, %] | Self CPU [unit, %] |
|---|---|---|
| maths_console_program (PID: 3508) | 137 (100,00 %) | 27 (19,71 %) |
| maths_console_program.Program.Main(System.String[]) | 110 (80,29 %) | 0 (0,00 %) |
| maths_console_program.Expansion.ToString() | 82 (59,85 %) | 0 (0,00 %) |
| maths_console_program.Expansion.ctor(int, int) | 28 (20,44 %) | 1 (0,73 %) |

Рис. 6.2 Пункт Hot Path інструменту CPU Usage

Варто зауважити, що обробка великих масивів даних (у даному випадку дані зберігаються в колекції List) підпадає під категорію паралелізму даних. Застосовувати паралелізм завдань не є доцільним, оскільки дані методи залажеть одне від одного. Тому якщо оброблювати їх паралельно, то одні задачі очікувати результату виконання інших.

За допомогою CPU Usage tool було визначено, що основний час, який приходиться на обчислення розглянутих методів, займає виконання циклів For та ForEach, що обробляють масив даних. Для застосування паралельних обчислень під час виконання цих циклів було використано методи Parallel.For та Parallel.ForEach класу Parallel з бібліотеки TPL (див. п. 5.4.1), який розподіляє колекцію на частини та виконує ітерації циклу паралельно, ітеруючи частини колекції.

Для уникнення проблеми із перегоном даних було застосовано колекції з простору імен System.Collections.Concurrent (п. 5.4.3). Це забезпечує безпечне використання колекції різними задачами одночасно.

Однак паралельна обробка призводить до того, що елементи записуються у результуючу колекцію в іншому порядку. Це є важливим, щоб результат обчислення із застосуванням паралельності не відрізнявся від результату послідовного обчислення. Тож для зберігання правильного порядку елементів після обробки колеції методами було використано потокобезпечну колекцію ConcurrentDictionary – аналог словника Dictionary із невпорядкованими елементами, що є парами ключ-значення. За допомогою ключів даного словника зберігається номер елементу із початкової колекції, а значення зберігає сам елемент. Таким чином, під час паралельної обробки зберігається ключ, який записується разом із елементом у результат, після чого можна відсортувати вихідний словник за ключами і відновити порядок.

У свою чергу це може призвести до додаткових обчислень. Однак дані витрати є незначними, що можна побачити по вимірах результату.

Отже, за допомогою бібліотеки TPL, а саме класу Parallel, із використанням потокобезпечних колекцій Concurrent Collections було реалізовано паралельне обчислення методів класу Expansion.

## 6.2 Результати та вимірювання

З використанням ConcurrentDictionary та циклів Parallel.ForEach та Parallel.For було досягнуто помітне підвищення продуктивності виконання програми. Для перевірки роботи програми оберемо N = 2 та d = 4. Звіт інструментів CPU та Memory Usage можна побачити на Рис. 6.3.

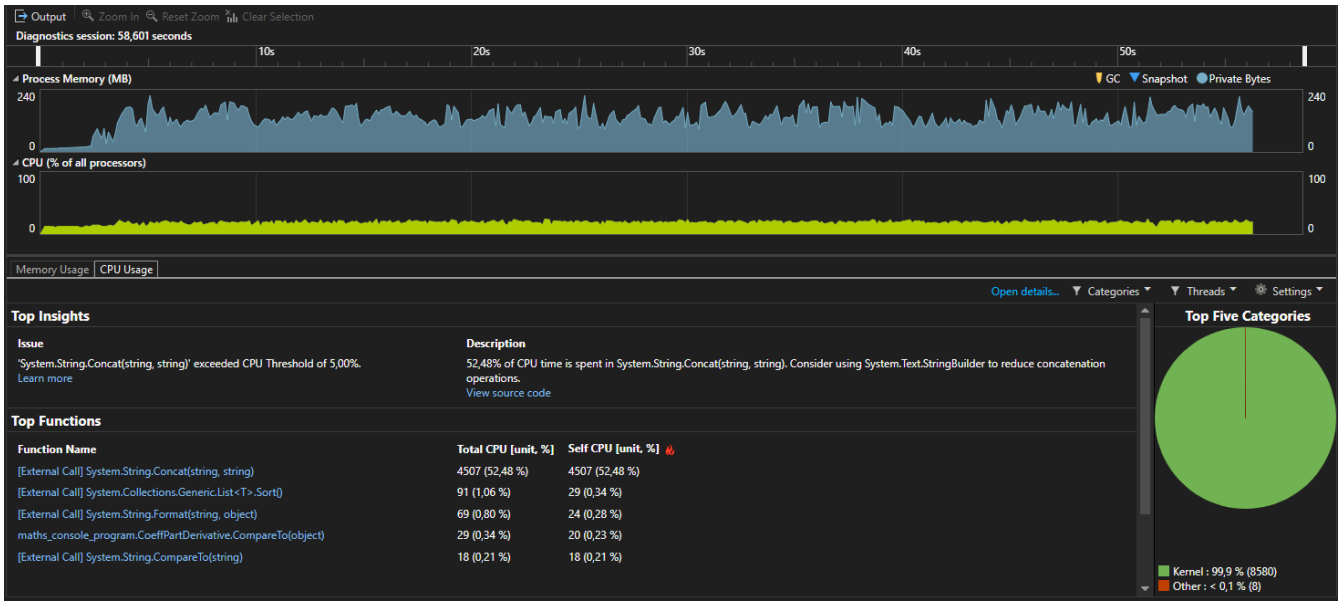


Рис. 6.3 Виміри CPU та Memory Usage при N = 3 та d = 2

Після реалізації маємо наступні виміри (у даному випадку обчислення функції розподілялися між 4-ма ядрами процесора Intel Core i7-4700HQ):

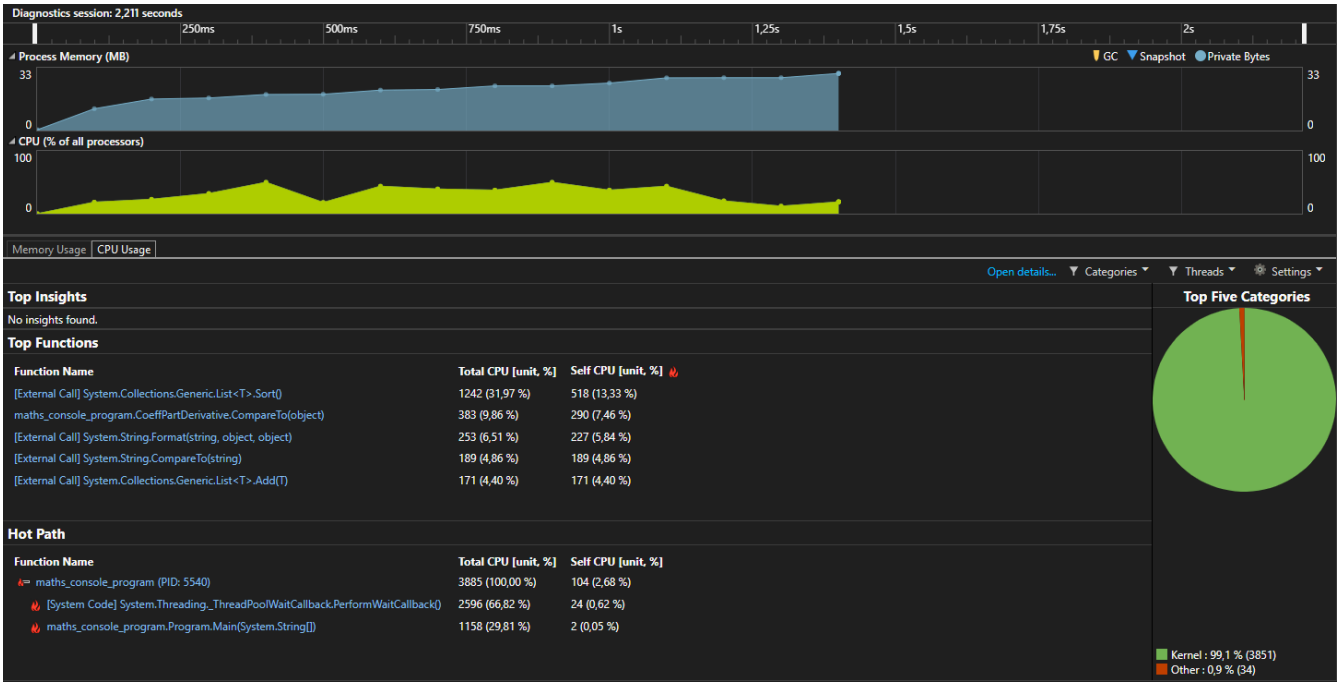


Рис. 6.4 Вимірювання роботи програмою із застосуванням паралельних обчислень

Використання інших колекцій разом із паралельними обчисленнями призвело до зміни часу обчислення програми з 55,5-56,5 с до 1,3-1,5 с. Якщо розглянути виконання конструктору класу Expansion, то час його обчислення змінився з 2,5 - 2,7 с до 1,1 – 1,25 с, задіюючи 4 потоки при цьому. Щодо методу ToString, то під час його виконання основну частину затримки складає багаторазові операції конкатенації (поєднання кількох об’єктів типу String): кожен доданок функції, яка обрахувалася конструктором, перетворюється у формат String та додається до результату таким чином, що на виході повертається об’єкт типу String, який виводиться на сторінці додатку. Оскільки використання паралельних обчислень потребує зберігання порядку елементів, які додаються, то у даному випадку використання ConcurrentDictionary для цього дозволило також уникнути багаторазового додавання до результату. Елементи переводяться у формат String і зберігаються у словнику, після чого вони поєднуються вбудованим у string методом Join, що використовує клас StringBuilder. На відміну від методів String, StringBuilder представляє змінний

рядок символів, що дозволяє уникати створенню нового рядка при конкатенації. Це у свою чергу зменшує використання Garbage Collector, який під час очистки пам’яті призупиняє роботу програми. Таким чином, час обчислення ToString при N = 2 та d = 4 змінився з 53 с до 0,2 с.

Результати вимірювання часу обчислення програми за допомогою Stopwatch при різних N та d наведено у таблиці 4 (виміри даних виконувалися при паралельному обчисленні програми на чотирьох ядрах процесора). Дані є більш точними, ніж CPU та Memory Usage, оскільки останні задіюють інструменти, що затримують виконання програми для збору метрик.

*Таблиця 4. Порівняння часу роботи за допомогою Stopwatch*

| N | d | До реалізації | | Після | |
|---|---|---|---|---|---|
| | | Конструктор | ToString | Конструктор | ToString |
| 1 | 1 | 1 мс | 1 мс | 1 мс | 1 мс |
| 2 | 1 | 1 мс | 1 мс | 1 мс | 1 мс |
| 2 | 2 | 20 мс | 10 мс | 10 мс | 4 мс |
| 2 | 3 | 165 мс | 990 мс | 104 мс | 18 мс |
| 2 | 4 | 1,5 с | 51 с | 0,76 с | 0,18 с |
| 3 | 2 | 0,8 с | 2,5 с | 0,43 с | 40 мс |
| 3 | 3 | 114 с | 128 хв | 48 с | 2,2 с |

Parallel також дозволяє обмежити максимальну кількість задач, які паралельно обробляються Parallel.For та Parallel.ForEach, шляхом вказання властивості MaxDegreeOfParallelism класа ParallelOptions. За допомогою цього можна порівняти часи виконання програми з розподіленням її на різну кількість задач, а відповідно й ядер, при паралельному обчисленні.

*Таблиця 5. Виміри часу виконання програми при різної кількості паралельних задач*

| К-сть | N | 2 | 2 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|
| | d | 2 | 3 | 4 | 2 | 3 |
| 1 задача | Загальний час | 31 мс | 213 мс | 1,87 с | 0,92 с | 109 с |
| | Конструктор | 21 мс | 170 мс | 1,5 с | 0,82 с | 104 с |
| | ToString | 10 мс | 43 мс | 372 мс | 0,1 с | 5 с |
| 2 задачі | Загальний час | 20 мс | 156 мс | 1,23 с | 0,6 с | 64,6 с |
| | Конструктор | 13 мс | 127 мс | 0,98 с | 0,55 с | 61,5 с |
| | ToString | 7 мс | 28 мс | 250 мс | 55 мс | 3,1 с |
| 3 задачі | Загальний час | 16 мс | 137 мс | 1,05 с | 0,5 с | 55,6 с |
| | Конструктор | 11 мс | 115 мс | 0,85 с | 0,45 с | 53 с |
| | ToString | 5 мс | 22 мс | 200 мс | 49 мс | 2,6 с |

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 задачі | Загальний час | 15 мс | 122 мс | 0,94 с | 0,47 с | 50,2 с |
| | Конструктор | 10 мс | 104 мс | 0,76 с | 0,43 с | 48 с |
| | ToString | 4 мс | 18 мс | 0,18 с | 40 мс | 2,2 с |

З даних наведених у таблиці 5 можна зробити висновок, що при використанні чотирьох ядер досягається зменшення часу обчислення удвічі.

Також можна побачити, що найбільше зменшення часу обчислення припадає на варіант із використанням двох задач, що обробляються на двох ядрах відповідно, у порівнянні з послідовним варіантом (тобто 1 задача). Наступні збільшення кількості задіяних ядер процесора дають менший приріст у продуктивності роботи програми, оскільки накладаються витрати на створення, розподіл між ядрами та синхронізацію задач.

Отже, за допомогою паралельних циклів можна дійсно підвищити продуктивність обчислення програми. Однак слід розуміти, що під час їх використання додаються витрати, пов'язані з розділенням колекції джерел та синхронізацією робочих потоків, і які можуть призвести до надмірного розпаралелення циклів. У будь-якому випадку, найкращий спосіб визначити оптимальну форму запиту - це тестування та вимірювання.

# ВИСНОВКИ

У даній роботі було підвищено продуктивність обчислення програми для розв’язку задачі перехідної густини ймовірності дифузійного процесу шляхом використання паралельних обчислень. Реалізація дозволила виконувати обчислення з більшими параметрами та меншими витратами часу. Відповідно дана програма може знайти застосування у багатьох сферах науки та життя, оскільки поняття дифузійного процесу є абстрактним і може застосовуватися нескінченну кількість разів.

Було розглянуто застосування стохастичних диференціальних рівнянь та подання СДР дифузійного процесу.

Було розглянуто платформу .NET, середовище розробки, на якому виконувалися реалізації паралельних обчислень, а також інструменти для вимірювання часу роботи програми.

Було порівняно два підходи до програмування з використанням багатопоточності: асинхронність та паралельність. Визначено, що для реалізації підвищення продуктивності обчислень перевага надається паралельному програмуванню.

Було реалізовано застосування паралельних обчислень та проведено тестування роботи програми при розпаралеленні між різною кількістю ядер процесора. Отримані результати демонструють зменшення часу обчислення вдвічі при використання чотирьох ядер.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дослідження Dataportal щодо цифровізації світу [Електронний ресурс] - https://datareportal.com/global-digital-overview

2. Oksendal, Bernt. Stochastic differential equations: an introduction with applications. Springer Science & Business Media, 2013. //

3. Офіційна документація .NET [Електронний ресурс] — Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/

4. Вступ до паралельного та асинхронного програмування в C# [Електронний ресурс] - Режим доступу: https://dotnettutorials.net/lesson/introduction-to-parallel-and-asynchronous-programming-in-csharp/

5. Різниця між одночасністю та паралельністю [Електронний ресурс] - https://www.geeksforgeeks.org/difference-between-concurrency-and-parallelism/

6. Visual Studio 2022 – Code faster, Work smarter [Електронний ресурс] – Режим доступу: https://visualstudio.microsoft.com/vs/

7. Аналіз продуктивності асинхронного коду .NET [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/visualstudio/profiling/analyze-async?view=vs-2022

8. Інструмент Visual Studio для аналізу використання ресурсів процесора [Електронний ресурс] - Режим доступу: https://learn.microsoft.com/en-us/visualstudio/profiling/cpu-usage?view=vs-2022

9. Gastón C. Hillar. Professional Parallel Programming with C# / Wiley Publishing, Inc. / 10475 Crosspoint Boulevard, Indianapolis // IN 46256

10. Ivanenko D.O, Higher order approximations in law of a multidimensional diffusion, May 7, 2019, Abstract //

11. Samuel Chamberlin. Azadeh Rafizadeh. "A generalized Newton–Girard formula for monomial symmetric polynomials." Rocky Mountain J. Math. 50 (3) 941 - 946, June 2020.//

12. Найбільш популярні мови програмування у 2023 році [Електронний ресурс] - https://www.coursera.org/articles/popular-programming-languages

13. Найпопулярніші IDE за пошуками сторінки для їх завантаження [Електронний ресурс] - https://pypl.github.io/IDE.html

14. Concurrent Collections у .NET [Електронний ресурс] - https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/data-structures-for-parallel-programming

15. Огляд BlockingCollection у .NET [Електронний ресурс] - https://learn.microsoft.com/en-us/dotnet/standard/collections/thread-safe/blockingcollection-overview

16. Документація Microsoft по інструментах Performance Profiler [Електронний ресурс] - https://learn.microsoft.com/en-us/visualstudio/profiling/profiling-feature-tour?view=vs-2022

17. Andrew Troelsen, Phil Japikse. Pro C# 10 with .NET 6 - Foundational Principles and Practices in Programming, 30 July 2022

18. Øksendal, Bernt. "Stochastic differential equations." Stochastic differential equations. Springer, Berlin, Heidelberg, 2003. 65-84. //