Міністерство освіти і науки України

Криворізький національний університет

Факультет інформаційних технологій

Кафедра комп’ютерних систем та мереж

Пояснювальна записка

до кваліфікаційної роботи бакалавра

за спеціальністю 123 «Комп’ютерна інженерія»

на тему: КОМП’ЮТЕРНА СИСТЕМА З ПАРАЛЕЛЬНОЮ АРХІТЕКТУРОЮ НА ОСНОВІ ПІДХОДУ СОМА

Проектував \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р. О. Шнякін

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. І. Купін

Нормоконтроль \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. І. Кузнєцов

Завідувач кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. І. Купін

Кривий Ріг

2025

Криворізький національний університет

Факультет інформаційних технологій

Кафедра комп’ютерних систем та мереж

Ступінь вищої освіти бакалавр

Спеціальність 123 «Комп’ютерна інженерія»

**ЗАТВЕРДЖУЮ**

Завідувач кафедри, голова циклової комісії

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. І. Купін

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ року

**З А В Д А Н Н Я**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “\_\_”\_\_\_\_ 20\_\_року №\_\_

2. Строк подання студентом роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно

розробити)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових

креслень)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультанти розділів роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада  консультанта | Підпис, дата | |
| завдання  видав | завдання  прийняв |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

7. Дата видачі завдання\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів роботи | Строк виконання етапів роботи | Примітка |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Студент** \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

**Керівник роботи** \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.Р

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

РЕФЕРАТ

Літера

Аркушів

ЗКІ-21

Пояснювальна записка: 44 сторінки, 20 рисунків, 3 таблиці, 4 лістингів фрагментів коду, 17 використаних джерел.

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.Р

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

РЕФЕРАТ

Літера

Аркушів

ЗКІ-21

Об’єкт аналізу – мультипроцесорна архітектура на основі COMA (Cache Only Memory Access – доступ тільки до кеш-пам’яті).

Кваліфікаційна робота складається з трьох розділів.

Перший розділ присвячений опису мультипроцесорної архітектури COMA.

У другому розділі виконується створення паралельної програми для рекурсивного обчислення подвійного факторіалу.

У третьому розділі проводиться тестування працездатності створеної програми, тестування продуктивності та аналіз показників надійності системи архітектури COMA.

МУЛЬТИПРОЦЕСОР, РОЗПОДІЛЕНА ПАМ’ЯТЬ, COMA, ПОДВІЙНИЙ ФАКТОРІАЛ, РЕКУРСІЯ.

Explanatory note: 44 pages, 20 figures, 3 tables, 4 listings of code fragments, 17 used sources.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.EN

The object of analysis is a multiprocessor architecture based on COMA (Cache Only Memory Access).

The course work consists of three sections.

The first section is devoted to the description of the COMA multiprocessor architecture.

In the second section, the creation of a parallel program for the recursive calculation of a double factorial is performed.

In the third section, the performance testing of the created program, performance testing and analysis of reliability indicators of the COMA architecture system are performed.

MULTIPROCESSOR, DISTRIBUTED MEMORY, COMA, DOUBLE FACTORIAL, RECURSION.

Зміст

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.З

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

ЗМІСТ

Літера

Аркушів

ЗКІ-21

[ПЕРЕЛІК СКОРОЧЕНЬ 7](#_Toc200021785)

[ВСТУП 8](#_Toc200021786)

[1. АРХІТЕКТУРА НА ОСНОВІ СОМА (CACHE ONLY MEMORY ACCESS – ДОСТУП ТІЛЬКИ ДО КЕШ-ПАМ’ЯТІ) 9](#_Toc200021787)

[1.2 Специфіка архітектури COMA (Cache-Only Memory Architecture) 11](#_Toc200021788)

[Висновок за розділом «АРХІТЕКТУРА НА ОСНОВІ СОМА (CACHE ONLY MEMORY ACCESS – ДОСТУП ТІЛЬКИ ДО КЕШ-ПАМ’ЯТІ) 19](#_Toc200021789)

[2 РЕКУРСИВНЕ ОБЧИСЛЕННЯ ПАРНОГО ФАКТОРІАЛУ 20](#_Toc200021790)

[2.1 Постановка задачі та розробка алгоритму 20](#_Toc200021791)

[2.2 Проблеми при реалізації паралельних обчислень 24](#_Toc200021792)

[2.3 Програмна реалізація розробленого алгоритму 26](#_Toc200021793)

[Висновок за розділом «РЕКУРСИВНЕ ОБЧИСЛЕННЯ ПАРНОГО ФАКТОРІАЛУ 32](#_Toc200021794)

[3 АНАЛІТИЧНА ЧАСТИНА 33](#_Toc200021795)

[3.1 Перевірка працездатності програми 33](#_Toc200021796)

[3.2 Виконання тестів продуктивності 34](#_Toc200021797)

[3.3 Аналіз показників надійності системи DDM COMA 36](#_Toc200021798)

[ВИСНОВКИ 38](#_Toc200021799)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 39](#_Toc200021800)

[ДОДАТКИ 41](#_Toc200021801)

[Додаток А 41](#_Toc200021802)

[Додаток Б 45](#_Toc200021803)

ПЕРЕЛІК СКОРОЧЕНЬ

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.ПС

AM – Attraction Memory (пам’ять притягання).

ccNUMA – Cache Coherent Non-Uniform Memory (неоднорідний доступ до пам'яті із забезпеченням когерентності кешів).

COMA – Cache-only Memory Access (доступ тільки до кеш-пам’яті).

DDM – Data Diffusion Machine (машина розповсюдження даних)

KSR – Kendall Square Research (назва компанії).

MIMD – Multiple Instruction, Multiple Data (множинний потік команд, множинний потік даних).

nccNUMA – Non-Cache Coherent Non-Uniform Memory (неоднорідний доступ до пам'яті без забезпечення когерентності кешів).

NUMA – Non-uniform Memory Access (неоднорідний доступ до пам'яті).

UMA – Uniform Memory Access (однорідний доступ до пам'яті).

ЕОМ – електронно-обчислювальна машина.

ОС – обчислювальна система.

ВСТУП

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.ВС

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

ВСТУП

Літера

Аркушів

ЗКІ-21

Тема паралельних обчислень є критично важливим аспектом сучасної сфери інформаційних технологій. Зі збільшенням обсягів даних і складністю проблем, які необхідно вирішити, паралельні обчислення стають все більш важливими в останні десятиліття. За своєю метою вони мають забезпечити більш швидкий час обробки задач та більш ефективне використання ресурсів, маючи перспективу до отримання максимально точних результатів за найменший використаний час.

Паралельні обчислення особливо актуальні в таких сферах, як наукові дослідження, розробка штучного інтелекту, аналітика великих даних і фінансове моделювання, де задіяні великі обсяги даних і використовуються складні алгоритми. У цих сферах паралельні обчислення мають потенціал для прискорення доcліджень і розробок, покращення процесу прийняття рішень і підвищення продуктивності.

Коли йдеться про підвищення продуктивності, масштабованості та надійності комп'ютерних cиcтем, збільшення кількості процесорів у системі може бути ефективним рішенням. Однак, при збільшенні кількості процеcорів виникають проблеми, пов'язані з доступом до спільної пам'яті. Якщо кілька процесорів намагаються отримати доступ до одних і тих же даних у спільній пам'яті одночасно, може виникнути конфлікт доступу, що призведе до уповільнення системи або її некоректної роботи.

Поділ пам'яті між процесорами може вирішити цю проблему, але може призвести до інших проблем. Наприклад, в архітектурах UMA (Uniform Memory Access) усі процесори мають рівний доступ до загальної пам'яті, що може призвести до витрат на пересилання великої кількості даних через шину даних при великій кількості процесорів. А в архітектурах NUMA (Non-Uniform Memory Access) пам'ять розбивається на окремі блоки, кожен із яких оброблюється лише певномим процеcором чи групою процесорів, що дозволяє збільшити ефективність пересилання даних, але водночас збільшує складність програмування та управління системою.

Вибір архітектури обчислювальної системи є важливим етапом при проектуванні будь-якої комп'ютерної cиcтеми, і він повинен бути заснований на специфічних вимогах і потребах конретної системи. Кожна архітектура має свої переваги та недоліки, і вибір правильної архітектури може суттєво вплинути на продуктивність, масштабованість та надійність системи. Тому важливо ретельно аналізувати специфіку системи та вимоги до неї, щоб обрати найбільш підходящу архітектуру, яка задовольнить усім потребам системи.

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.01.АНОС

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

АРХІТЕКТУРА НА ОСНОВІ СОМА (CACHE ONLY MEMORY ACCESS – ДОСТУП ТІЛЬКИ ДО КЕШ-ПАМ’ЯТІ)

Літера

Аркушів

ЗКІ-21

1. АРХІТЕКТУРА НА ОСНОВІ СОМА (CACHE ONLY MEMORY ACCESS – ДОСТУП ТІЛЬКИ ДО КЕШ-ПАМ’ЯТІ)

1.1.1 Теоретичне тло для сучасних великих процесорів обчислювальних систем

Прямі багатопроцесорні системи можуть обійтися без друкування за наявності потужностей, що володіють теоретичною силою, необхідною для підготовки спеціалізованих процесорів-схем у їхніх засобах, що можуть запропонувати: НЕПОПОСЛІДНІСТЬ БІСЛІННОСТІ.

Багатопроцесорні системи забезпечують можливість одноразово створювати кілька незалежних операцій; ПАРАЛЕЛЬНЕ ОБЧИСЛЕННЯ: це відбувається одночасно з кількома процесами. Паралельне обчислення є ключовою владою обчислювальних систем, що забезпечують швидкий розвиток технологій потужних обчислень.

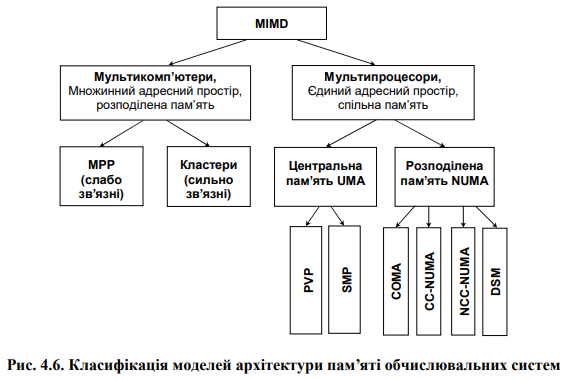
Використання багатопроцесорних архітектур, що створюють кілька обчислювальних процесорів (ЦП) та/або обробних вузлів одночасно, є однією з найважливіших тенденцій підвищення обчислювальної потужності. У цьому аспекті моделі організації, доступної для збору даних, але в деталях, значення, які вона має, значно впливають на високу продуктивність систем, масштабованість і стійкість до роботи.

Це два окремі шляхи організації всіх банківських організацій багатопроцесорних систем, тобто систем із розподіленою пам'яттю та систем із розвиненою пам'яттю. Виберіть одну з цих моделей, що базується на різних масштабах та особливостях обчислювального процесу [13].

Насамперед продуктивність може бути високою, усі процесори працюють на одній глобальній адресній пам'яті; таким чином, ви можете ефективно синхронізувати свою роботу. Такі системи називають жорстко пов'язаними живучими. Спеціалізовані контролери та шина даних гарантують, що саме в цих регіонах оперативна пам'ять доступна [13].

В іншому боці, у зручному місці з розподіленою пам'яттю, процесор (або вузол) обладнаний своєю пам'яттю, і ця мова між вузлами відводиться шляхом зі зручностями. Такі системи називаються слабо пов'язаними ланцюгами [13].

В архітектурі системи перший варіант характеризується мегапроцесорами, а інші - багатокомп'ютерами. Усі можливі типи взаємодій можуть бути реалізовані в MIMD (Багатоінструкція - Багатодані) і зручні для всіх типів пам'яті. Рисунок 1.1 являє собою повну категоризацію моделей архітектури будівель, які найчастіше використовуються моделями пам'яті в комп'ютерних категоріях.



арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Рисунок 1.1 – Класифікація моделей архітектури пам’яті обчислювальних систем класу MIMD [13]

В інших категоріях таких типів використовується архітектура UMA (Uniform Memory Access - уніфікований доступ до пам'яті), у якій час доступу до пам'яті завжди доступний для всіх процесорів. Така організація доступу до даних об'єктів здійснюється на центральному контролері пам'яті та повністю системних шинах, які використовують наші процесори для доступу до пам'яті.

Одна з проблем архітектури UMA - це масштабність. Зі збільшенням кількості процесорів ми можемо збільшити пропускну здатність шин, доступність і зниження продуктивності систем. Більша кількість таких архітектур визнана для систем із числом процесорів до 8, від 32 до 64, але в таких порядках [13], [16] це значно менший ефект.

Найважливішим аспектом для систем UMA є уважність до неправильності модулів пам'яті. Опитування всіх процесів дають змогу вибрати один з адресних простирадл, використовуючи модуль, який може призвести до дестабілізації всієї системи.

Сектор галузових застопорних систем UMA - це сектор заправних систем, а також систем з великою проблемою і розкидом у будь-який час [4].

З використанням раніше заданих нестандартних моделей UMA, наукове об'єднання більш уважно ставиться до архітектури з відповідним доступом до пам'яті, що містить NUMA (Non-Uniform Memory Access - нерівномірний доступ до пам'яті), про що доведеться йти далі в майбутньому підрозділі

1.2 Специфіка архітектури COMA (Cache-Only Memory Architecture)

COMA (технологія кеш-пам'яті як єдина архітектурна пам'ять) - це особливий вид архітектури NUMA-системи, у якій основна частина архітектури складається з лише кеш-пам'яті.

Основна концепція COMA полягає в чіткому розмежуванні між локальним і глобальним знанням. У таких системах цей модуль використовує привід, який дає змогу адаптуватися до простору даних у просторі, що динамічно підтверджує характеристики загального процесу.

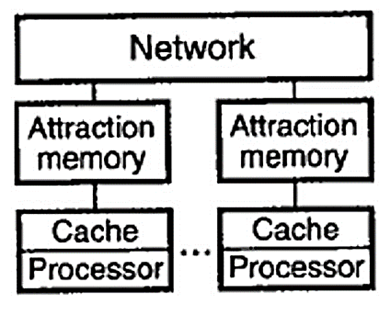


Рисунок 1.2 – Узагальнена схема архітектури COMA [3]

На відміну від традиційних систем NUMA, під час використання локальних вузлів можна фіксувати повідомлення в окремій спільноті в глобальному адресному просторі, у середньостатистичному COMA можна перебиратися між кешами процесорів швидко, тож, це означає, що потрібно забрати на зберігання конкретні дані.

У результаті в архітектурі COMA фізичне розташування будівлі не є фіксованим, а дані не пов'язані з одним модулем пам'яті. Така динаміка підвищує однорідність, якщо знати необхідні дані в локальних умовах, тому самим мінімізуємо завантаження пересилювання цих даних і трафіку в західних країнах [2], [13].

У локальних системах COMA є кожен процесор, що обслуговує великий кеш іншого рівня (L2), який заповнений складами глобальних систем. Середній доступ до нього можна отримати завдяки тому, що він може отримати більше задоволення. Яка інформація розширюється найбільше в одному локальному кеші одночасно, протокол без утримання узгогенності кешу обирається, щоб переконатися, що підтримуються узгоджені копії, так що це найбільша інформація зберігається в продовж усієї системи.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Однією з ключових концепцій COMA є **Attraction Memory (AM)** — проміжна форма між кешем та спільною пам’яттю. Вона використовується для тимчасового зберігання даних, які були «притягнуті» до кешу в результаті звернення процесора. Саме завдяки механізму **тегування елементів** і спеціалізованим контролерам, дані автоматично переміщуються між кешами в залежності від обчислювальних потреб [2].

Архітектура COMA також передбачає використання **каталогів кешів (cache directories)** — спеціалізованих структур керування, що зберігають інформацію про розташування даних у системі. Кожен каталог містить **теги кеш-ліній**, а також біти стану, які сигналізують про актуальність, модифікованість або застарілість відповідних елементів даних.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Важливою перевагою COMA є **гнучкість у розміщенні даних**, що сприяє оптимальному використанню обчислювальних ресурсів та підвищенню продуктивності системи при високих навантаженнях. Завдяки динамічному переміщенню даних, COMA-архітектури демонструють добру масштабованість, дозволяючи ефективно працювати навіть при наявності сотень або тисяч процесорів.

Водночас COMA має і низку суттєвих недоліків. По-перше, складність апаратного забезпечення, необхідного для підтримки когерентності, зумовлює зростання вартості системи. По-друге, у випадках інтенсивного паралельного доступу до одних і тих самих змінних з боку кількох процесорів можливі часті переміщення кеш-ліній між вузлами. Це може спричинити **ефект «гарячої точки»**, при якому кеш-пам’ять постійно передається від одного процесора до іншого, що веде до зниження загальної ефективності обчислень.

Попри зазначені труднощі, COMA залишається перспективною альтернативою іншим підходам до побудови паралельних систем, зокрема в умовах, коли пріоритетом є **мінімізація затримок доступу до пам’яті** та **максимізація локалізації даних**.

1.3 Прикладні реалізації архітектури COMA у мультипроцесорних системах

Попри те, що архітектура COMA (Cache-Only Memory Architecture) демонструє вагомі теоретичні переваги, практичних реалізацій даної концепції порівняно небагато. Це пов’язано з високою складністю апаратної підтримки, необхідністю впровадження багаторівневих протоколів когерентності, а також зростанням витрат на розробку й експлуатацію таких систем.

Проте було десять складських проєктів у галереях паралельних зібрань, які допоможуть реалізувати ідею COMA у фізичних навчальних центрах. Найпоширеніші застосунки в галузі архітектури включають у себе машини для поширення даних (DDM) і Kendall Square Research-1 (KSR-1).

Машина розповсюдження даних (DDM)

Data Diffusion Machine (DDM) - один із класичних застосунків ієрархічної реалізації COMA, у якій кожна пам'ять системи проглядається як розподілена кеш-пам'ять. Крім того, фундаментальним елементом пристрою є обрахована сума «процесор-пам'ять», отримана через системну шину, що є додатковим внутрішнім спілкуванням системи.

Погляньте на нижні рівні дерева з вузлами на інших рівнях для додаткового каталогу або поєднання в схемах DDM. Це те, що унеможливлює маршрутизацію даних про завантаження та вилучення файлів. Таким чином, систему можна розширити до великої кількості вузлів без витрат.

.

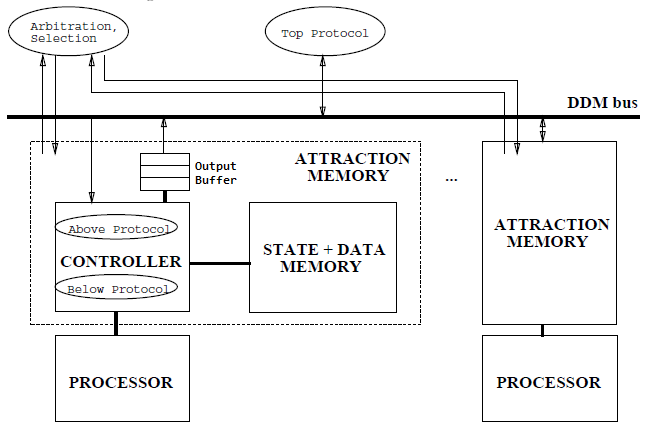


Рисунок 1.3 – Архітектура базової одношинної DDM [3]

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Пам'ять привабливості підключається до процесора для додаткового протоколу «Звуку», а до шини DDM - для додаткового протоколу «Зверху».

На верхньому рівні структури (на найменшому) вводиться протокол «Вгору», який приднати основний DDM до заговорного дерева.

У цьому випадку протокол передачі даних слайдів за алгоритмом рис. 1.4.

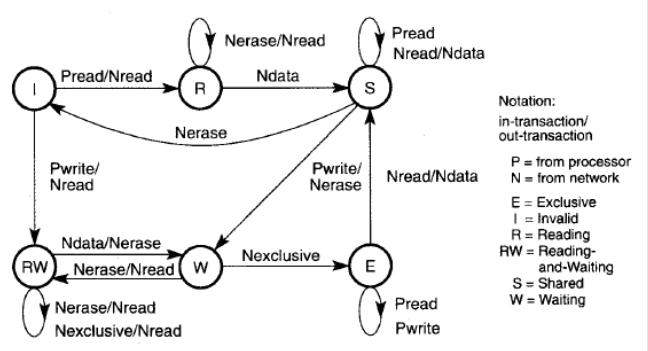


Рисунок 1.4 – Робота протоколу пам’яті притягання [3]

ДДМ має механізм, зумовлений змінами, заснованими на моделях асинхронної взаємомодії та використовості для розв'язування транзакцій запитки та підтвердження з часом.

Шина DDM забута в побудованій схемі арбітражу, щоб полегшити конфлікти і спробувати обслужити запитки без можливості усунення тупикової ситуації. Це можна скоротити час очищення під час роботи з високою інтенсивністю завантаження та підвищення продуктивності того, що залишилося.

Шина DDM, порівняно з іншими шинами, має механізм вибору, який гарантує, що не прикро, якщо ви обслужите один і той самий запит. При цьому виключається, що кожна транзакція на шинах не породжує нижчу одну нову транзакцію для цих шин, що є необразливо умово для унікальності тупикових ситуацій [2]

Робота протоколу шини DDM розроблена на рисунку 1.5..

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

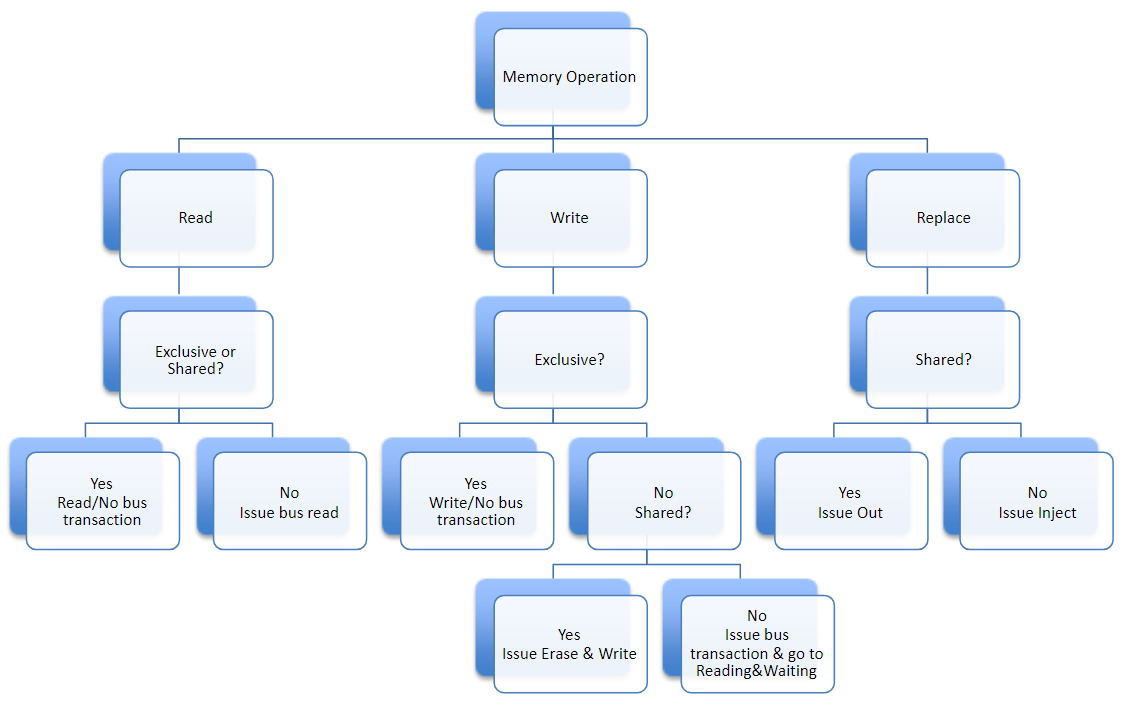
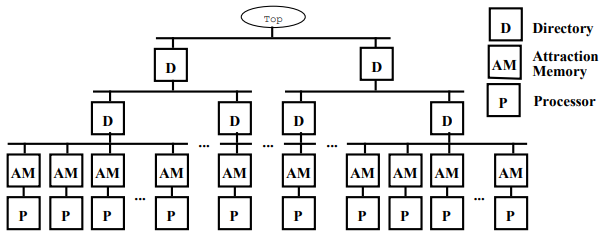


Рисунок 1.5 – Алгоритм роботи протоколу шини DDM

Простий, єдиний вузол DDM, розмови вище, можливо, будуть підкомпонентами великого, арархічного DDM, і це буде очевидно на мал. 1.6, ця архархічна структура показана у двох містах, на вершинах розміщений новий каталог, що має назву мови між шиною і більш високим рівнем шин того ж типу.



арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Рисунок 1.6 – Ієрархічна DDM (трирівнева) [2]

Важлива влада DDM для мене полягає в тому, що він перебуває в нещодавньому пост-копіюванні глобальних даних - зараз цей період часу вимірюється в кешах і управляється меншою мірою у світі необхідності за рахунок кількості обювальних процесів. Це ключова відмінність між COMA і традиційними архітектурами NUMA.

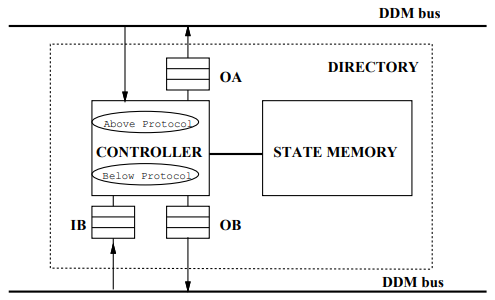


Рисунок 1.7 – Архітектура каталогу (директорії) [2]

Каталог, на малюнку 1.7, також має невеликий «верхній» вихидний буфер (ОА: вихидність звіра) для тимчасового виконання транзакцій, який спрямовується на верхню шину. Транзакції, які лежать на нижній шині, знімаються в нижньому вихідному буфері (OB), тоді як транзити з нижньої шини знімаються в нижньому вихідному буфері (IB). Коли IB має достатньо часу для слотів, щоб просканувати свій верстат пам'яті, щоб прочитати дані в каталозі [2].

Архівний DDM і його протокол можуть бути дуже поширені з архітектурою. Один DDM відштовхується від них тим, що в протоколі використовуються перехідні стани, які не включають фізичну повну пам'ять і не вилучення хороших даних (високоякісного кешу) в меншій мірі [2].

Kendall Square Research-1 (KSR-1)

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Один із перших комп'ютерів Midi. Система KSR-1 - це перша реалізація продукту COMA, розроблена компанією Kendall Square Research. Архітектура машин внутрішньої концепції ALLCACHE - це функції, що дозволяють реалізувати зміну логічного єдиного адресного простору в шляхах організації великих локальних кешів.

Частини KSR-1 обрано в сегменті, кожна з яких має склад максимум із 32 процесорних вузлів і двох каталогів ALLCACHE. Розміщення кеш-стору в різних каталогах матриць, що дає змогу визначити статус кожної підсторінки в локальному кеші, наприклад, «недійсний», “копія”, «ексклюзивний власник» або «ексклюзивний власник».

KSR-1 побудований за допомогою масиву агентів ALLCACHE, створених у єдиному деревопідпорядкованому середовищі. Іноді можна знайти до 32 процесорних вузлів і в двох каталогах ALLCACHE

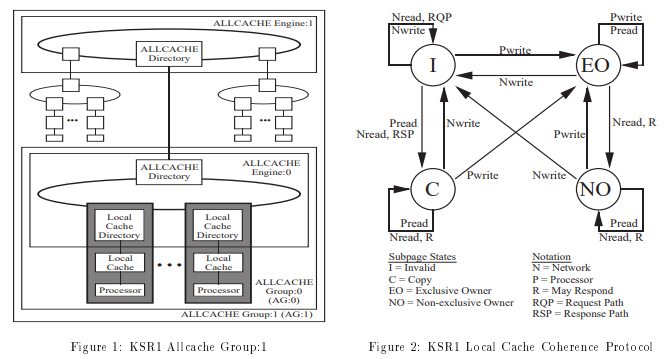
.

Рисунок 1.8 – Приклад архітектури мультипроцесора KSR-1 [6]

ALLCACHE - це система каталогів, яка забезпечує послідовну узгодженість для своєї політики узгодженості кешу. Сторінки та підсторінки зберігаються в ALLCACHE. Розподіл пам'яті в локальних кешах здійснюється лише на рівні сторінки (використовуючи сторінки розміром 16 КБ). Підсторінка має розмір 128 байт і є одиницею для переміщення даних між локальними кешами.

Розподілений каталог — це матриця, в якій є рядок для кожної підсторінки, стовпець для кожної підсторінки, і стовпець для кожного локального кешу. Запис у матриці порожній, якщо жодної копії підсторінки не існує у відповідному локальному кеші [7].

Незаповнені елементи можуть перебувати в одному з чотирьох основних станів:

* Недійсний (I): локальний кеш не має дійсної копії цієї підсторінки.
* Копія (C): існує локальна копія підсторінки лише для читання.
* Ексклюзивний власник (EO): локальний кеш має підсторінку, і жодної іншої копії цієї підсторінки немає в інших кешах.
* Неексклюзивний власник (NO): локальний кеш володіє підсторінкою, проте може існувати інша копія цієї підсторінки в інших кешах [6].

Фактичний протокол KSR-1 має понад 200 станів через одночасні/незадоволені запити та розбиті транзакції. Проте, протокол узгодженості для підсторінки може бути спрощено до лише чотирьох станів, як показано на діаграмі переходів станів на рисунку 1.9.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

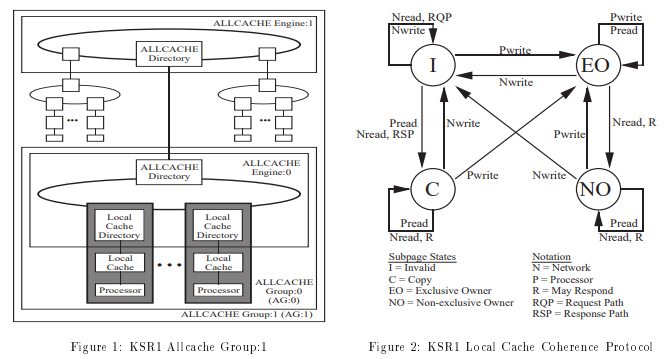


Рисунок 1.9 – Алгоритм роботи протоколу узгодженості KSR [6]

Локальний блок керування кешем змінює стан підсторінки або після отримання запиту від процесора (Pread/Pwrite), або після відстеження відповідного запиту в кільцевій мережі (Nread/Nwrite) [6].

Протокол узгодженості KSR забезпечує функцію автоматичного оновлення запитів на читання мережі. Перший кеш у кільці, який містить підсторінку в не-інвалідному стані, відповідає даними на цей запит. Усі локальні кеш-пам’яті в кільці (або кільцях) між відповідачем і запитуючою коміркою (тобто вздовж шляху відповіді), які мають підсторінку в інвалідному стані, автоматично оновлюють свої копії, поки локальний кеш не звільниться [6].

Унікальною особливістю KSR-1 є **підтримка автоматичного оновлення даних** під час проходження запитів по кільцю, що дозволяє локальним кешам уздовж шляху відповіді актуалізувати свої копії. Також реалізовано спеціалізовані інструкції попереднього завантаження (prefetch) і післязбереження (poststore), які додатково зменшують затримки доступу до даних і покращують загальну ефективність системи.

Хоча KSR-1 та DDM залишаються прикладами одиничних реалізацій, вони відкрили нові підходи до побудови масштабованих паралельних систем з гнучкою організацією пам’яті. Досвід, здобутий під час розробки COMA-систем, активно використовується в сучасних дослідженнях над гібридними архітектурами, включаючи сучасні рішення на базі когерентного NUMA, багатоадресних кешів та адаптивної маршрутизації пам’яті.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.01.АНОС

Висновок за розділом «АРХІТЕКТУРА НА ОСНОВІ СОМА (CACHE ONLY MEMORY ACCESS – ДОСТУП ТІЛЬКИ ДО КЕШ-ПАМ’ЯТІ)

У розділі було розглянуто архітектуру мультипроцесорних систем на основі COMA (Cache Only Memory Access), що дозволяє оптимізувати доступ до пам'яті шляхом використання лише кешу. Було проаналізовано переваги та недоліки такої архітектури, а також методи приховування затримок, що виникають під час доступу до віддаленої пам'яті, зокрема за допомогою попередньої вибірки та післязбереження. Архітектура COMA сприяє зменшенню трафіку зв’язку та підвищенню продуктивності у багатопроцесорних система

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.01.ВРАНОС

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

Висновок за розділом «АРХІТЕКТУРА НА ОСНОВІ СОМА (CACHE ONLY MEMORY ACCESS – ДОСТУП ТІЛЬКИ ДО КЕШ-ПАМ’ЯТІ)

Літера

Аркушів

ЗКІ-21

# 2 РЕКУРСИВНЕ ОБЧИСЛЕННЯ ПАРНОГО ФАКТОРІАЛУ

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.02.РОПФ

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

РЕКУРСИВНЕ ОБЧИСЛЕННЯ ПАРНОГО ФАКТОРІАЛУ

Літера

Аркушів

ЗКІ-21

## 2.1 Постановка задачі та розробка алгоритму

У рамках даної кваліфікаційної роботи об'єктом дослідження є процес реалізації паралельних обчислень подвійного факторіалу натурального числа за допомогою методів рекурсії. Ми здійснюємо обчислення у вигляді функції мовою програмування C# в середовищі .NET Framework.

Подвійний факторіал — це бінарна операція, яка узагальнює факторіал. Він позначається символом n!! з добутком по всіх додатних цілих числах, парність яких збігається з n (тобто всі непарні або всі парні), що не перевищують n.·

для **парного n**:  
n!!=n⋅(n−2)⋅(n−4)⋅…⋅2;

· · для **непарного n**:  
n!!=n⋅(n−2)⋅(n−4)⋅…⋅1;

· · за домовленістю:  
0!!=1.

Ця операція часто зустрічається у комбінаториці, теорії графів та різних областях прикладної математики. Зважаючи на те, що подвійні факторіали зростають дуже швидко, навіть для малих n, обчислення цієї функції може бути затратним.

#### Вибір інструментарію

Обчислення були реалізовані за допомогою мови C# з наступних причин:

* підтримка **багатопотокового програмування** через прості в реалізації конструкції, як-от Thread, Task, Parallel, async/await;
* уся «бібліотечна благодать», що походить з великої кількості доступних бібліотек для паралельного коду;
* можливість візуального перегляду процесу на полотні розробки як у Windows Forms для проектування інтерфейсу користувача;
* хороша читабельність коду та легкість у відлагодженні.

#### Формалізована постановка задачі

Метою є розробка програмного модуля, що реалізує **паралельне рекурсивне обчислення** подвійного факторіалу з урахуванням розподілу навантаження між k потоками. Вихідними параметрами алгоритму є:

* n — вхідне число, для якого обчислюється подвійний факторіал;
* recursive — логічний параметр, що визначає режим обчислення (рекурсивний або звичайний).

#### Алгоритмічний підхід

Основна ідея полягає в розподілі рішень факторизації на k груп і призначенні кожної групи окремому потоку. Чинники створюються на основі парності числа n, і тоді кожен добуток підмножини обчислюється окремо

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

Залежно від прапора recursive, для множення використовується або:

* **рекурсивна функція** MultiplyRecursive, яка ділить масив на підмасиви до базового випадку (1 або 0 елементів), або
* **ітеративна функція** MultiplySimple, що послідовно перемножує елементи.

Результати, отримані кожним потоком, згодом **агрегуються в єдиний підсумковий добуток**, що й є значенням n!!.

Цей підхід дозволяє зменшити складність обчислень шляхом:

* рівномірного розподілу обчислювального навантаження;
* зниження ризику **конфліктів доступу до спільних ресурсів** (shared state);
* забезпечення **високого рівня масштабованості** при збільшенні значення n або кількості потоків.

Проблеми реалізації паралельного коду розглядаються в подальших підрозділах, а саме: розподіл ресурсів, методи синхронізації та приклади програмного коду.

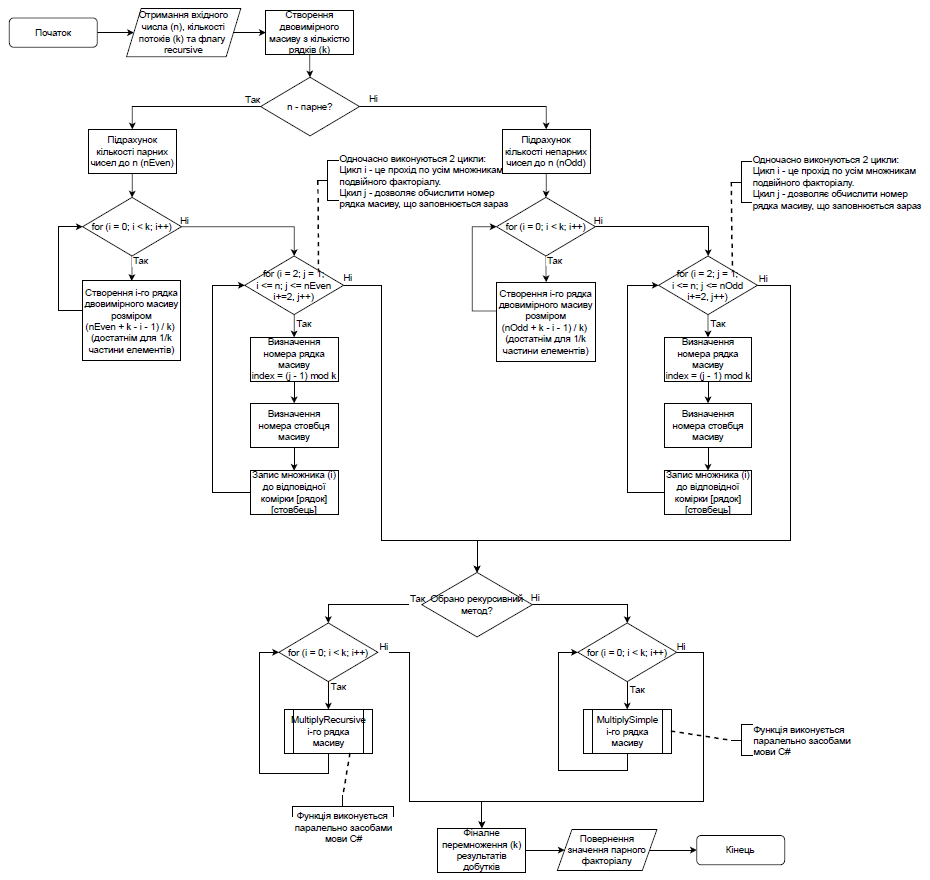
У даній кваліфікаційній роботі запропоновано наступний алгоритм обчислення подвійного факторіалу за допомогою рекурсії (рисунок 2.1).

Він приймає на вхід n, кількість потоків k і прапорець рекурсії (якщо ви бажаєте, щоб він виконувався з рекурсією або без неї). Функція тоді обчислює кількість парних або непарних чисел залежно від того, чи n парне або непарне.

Далі вона створює k наборів масивів для розподілу чисел, заповнює ці масиви значеннями (кожен k-й індекс з масиву чисел, але із застосуванням паритетного фільтру) і запускає k завдань для множення значень у відповідних масивах.

Якщо прапорець рекурсії істинний, це буде обчислено рекурсивно за допомогою функції MultiplyRecursive, в іншому випадку використовується функція MultiplySimple для простого перемноження елементів масиву.

Після завершення всіх завдань отриманий добуток повертається як кінцевий результат — подвійний факторіал числа n.

Рисунок 2.1 – Блок-схема алгоритму роботи функції обчислення подвійного факторіалу

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

Алгоритм рекурсивної функції MultiplyRecursive представлений на рисунку 2.2. Ця функція приймає одновимірний масив цілих чисел, позначений як arr, і рекурсивно перемножує кожен елемент цього масиву.

Спочатку перевіряється, чи дорівнює довжина масиву нулю; у такому разі функція повертає 1. Якщо довжина масиву дорівнює одиниці, вона повертає значення єдиного елемента в масиві.

Якщо довжина масиву більше одиниці, тоді спочатку знаходиться середній елемент масиву arr, потім створюються два нових масиви: лівий масив (left) (з першими mid елементами оригінального масиву arr) і правий масив (right) (з рештою елементів оригінального масиву arr).

Потім функція викликає сама себе для лівого і правого масивів, щоб отримати результати, і множить два отримані результати. Нарешті, функція повертає добуток всіх елементів масиву arr, як показано нижче.

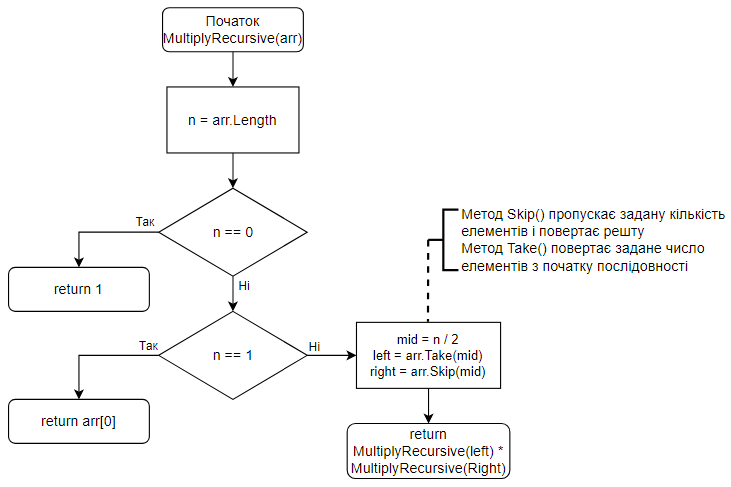


Рисунок 2.2 – Алгоритм рекурсивної функції MultiplyRecursive

Алгоритм функції MultiplySimply є звичайним перемноженням елементів одновимірного масиву (рисунок 2.3).

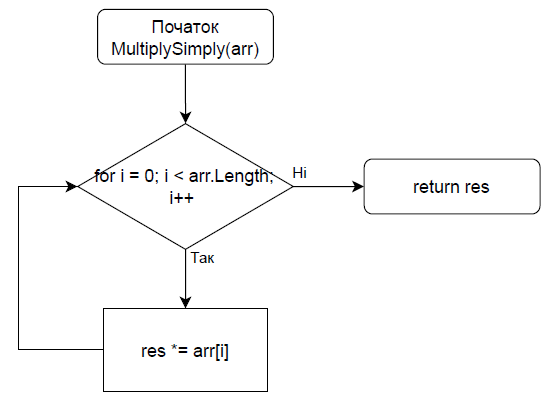


Рисунок 2.3 – Алгоритм функції MultiplySimply

Програма для обчислення парного факторіалу буде володіти графічним інтерфейсом, який буде виконаний за допомогою .NET технології Windows Forms.

Windows Forms – це платформа інтерфейсу користувача для створення класичних програм Windows. Вона забезпечує один з найефективніших способів створення програм за допомогою візуального конструктора в Visual Studio [14].

Використання Windows Forms дозволяє душе швидко редагувати зовнішній вигляд та функції елементів через візуальний конструктор, надає можливість розширяти функціонал.

Таким чином, для написання паралельної програми рекурсивного обчислення подвійного факторіалу за створеним алгоритмом буде використано мову програмування C# та технологію Windows Forms.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

2.2 Проблеми при реалізації паралельних обчислень

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

Windows Forms — це фреймворк користувацького інтерфейсу для створення настільних додатків Windows. Це один з найкращих способів розробки додатків в Visual Studio IDE, оскільки він використовує візуальний дизайнер. У Windows Forms дуже швидко відбувається розміщення елементів керування, а зовнішній вигляд та функціональність елементів легко редагуються у візуальному дизайнері та розширюються.

Отже, щоб написати паралельну програму для рекурсивного обчислення подвійного факторіалу на основі розробленого алгоритму, ми будемо використовувати мову програмування C# та технологію Windows Forms.

Під час збільшення потужності паралельних обчислень обчислювальні вчені стикаються з проблемою перегонів даних, що також називається умовами перегонів. Це концепція, коли два або більше потоки намагаються отримати доступ до спільного ресурсу. Це може стосуватися будь-чого: оперативної пам’яті (найчастіше), інших видів пам’яті (наприклад, роботи з одним і тим же файлом), пристрою, підключеного до комп'ютера.

Якщо кілька потоків одночасно намагаються отримати доступ до спільної області, і принаймні один з них щось записує, результати часто будуть безглуздими.

Наприклад, два потоки можуть спробувати прочитати поточне значення змінної безпосередньо з її пам'яті та зберегти його у своєму регістрі в той самий час. Після цього вони можуть виконати операцію зі змінною в пам'яті регістру одночасно. Однак вам доведеться записати отриманий результат у комірку результатів вашої процедури окремо. Потік, який першим запише в свою комірку пам'яті, «виграє», оскільки вміст, записаний у першу комірку пам'яті, буде перезаписаний, коли другий потік запише у спільну комірку пам'яті. Звичайно, якщо ми затримаємо виконання цих потоків, то, щоб потім виконати другий набір команд, сума має відображати роботу всіх потоків. Але щоб отримати цей результат, потрібно знайти спосіб координації роботи потоків, що працюють паралельно.

У якості реального прикладу, нижче наведено фрагмент коду з паралельної програми обчислення подвійного факторіалу шляхом використання одного єдиного циклу з перемноженням елементів (лістинг 2.1).

Лістинг 2.1 – Фрагмент коду, в якому присутня проблема перегонів даних

Parallel.For(2, n + 1, new ParallelOptions { MaxDegreeOfParallelism = threads }, i =>

{

if (i % 2 == 0)

{

var value = new BigInteger(i);

result \*= value;

}

});

У цьому фрагменті коду є потенційні перегони даних. Вони можуть ставатися при одночасному зверненні та редагуванні спільної змінної result  різними потоками, що створює метод Parallel.For. Це призводить до непередбачуваних результатів – при однакових вхідних даних видаються різні відповіді (рисунок 2.4).

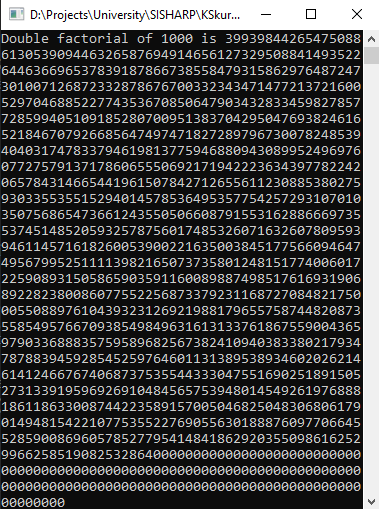
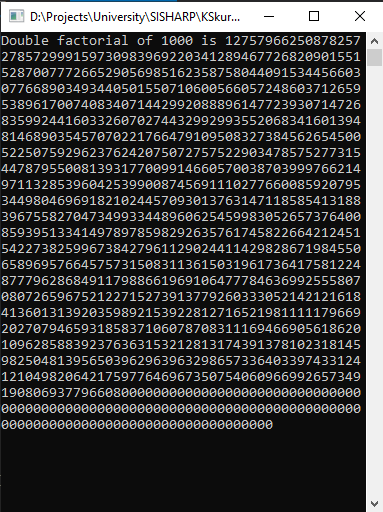


Рисунок 2.4 – Демонстрація перегонів даних в паралельній програмі (при однакових вхідних значеннях видаються різні непередбачувані відповіді)

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

У C# для багатопотокових програм існує кілька механізмів для запобігання проблемам змагання даних (хоча це не є повним списком), зокрема:

* Застосування блокувань (наприклад, оператор lock) лише тоді, коли це необхідно: тільки на тих частинах коду, де може фактично виникнути проблема змагання даних, оскільки отримання та звільнення блокувань може викликати додаткові затримки і, отже, знижувати продуктивність.
* «М'ютекси» та «семафори» забороняють доступ до ресурсів, поки вони не звільняться, якщо ними користувався попередній потік.
* Використання конструкцій синхронізації, таких як AutoResetEvent і ManualResetEvent, які дозволяють потоку чекати, доки певна умова не буде виконана, перш ніж розпочати.
* Інструменти синхронізації, такі як AutoResetEvent та ManualResetEvent, які дозволяють потокам чекати певного стану перед продовженням.
* Застосування паралельних процедур, як, наприклад, вбудований Parallel.ForEach та Parallel у методах, які автоматично розподіляють роботу між потоками.
* Усі колекції з безпекою потоків, Task, Thread тощо

Незалежно від того, який синхронний метод ви виберете, особливо важливим є оптимізувати алгоритм та програмний код, щоб досягти максимальної продуктивності та уникнути змагання даних під час паралелізації програми.

У наведеному вище алгоритмі (розділ 2.1) він повинен бути захищеним від змагань даних на теоретичному рівні, оскільки дані, які будуть оброблені його потоками, рівномірно розподіляються на початку серед і в кількості масивів, що дорівнюють кількості потоків, і потік обробляє свій власний масив даних. Кінцеве множення всіх значень, обчислених потоками, виконується лише після того, як усі потоки повернули налаштувальник результату. Тому в будь-який момент часу, протягом виконання програми, конкуруючі потоки не працюють з загальними даними, тобто вони не страждають від змагання даниx.

2.3 Програмна реалізація розробленого алгоритму

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

Оскільки я збираюся написати програму, яка виконуватиме цей алгоритм, у мене є проблема, що подвійний факторіал числа може стати величезним, якщо число велике. Це може перевищити діапазон типу long, діапазон типу int тощо. Ви можете скористатися структурою BigInteger, щоб вирішити цю проблему.

BigInteger — це клас у просторі імен System.Numerics системи .NET, тип, що містить ціле число довільного розміру або порядку, теоретично, яке не обмежене певним інтервалом і називається цілим числом із довільною точністю [1].

BigInteger, по суті, є числом, що зберігається у формі масиву байтів, у якому кожен байт масиву насправді просто містить частину двійкового подання цього числа. Це те, що дозволяє BigInteger зберігати та виконувати арифметичні операції з числами будь-якого розміру. Коли для типу BigInteger виконується арифметична операція, двійкове подання цих цілих чисел перетворюється на байтові подання, які обробляються окремо. Це повільніший процес, але він дозволяє програмі працювати з числами довільної довжини з повною точністю [1].

Отже, коли ви не обмежуєте свої числа в програмі та виводите їх у стандартний висновок, ви повинні використовувати BigInteger, оскільки n!! може бути дуже великим.

На основі алгоритмів на рисунках 2.1-2.3 розроблено код для паралельної програми обчислення подвійного факторіалу на C#. Центральна функція, яка обчислює значення подвійного факторіалу, визначена в окремій функції CalculateDoubleFactorial (листинг 2.2).

Лістинг 2.2 (а) – Програмний код функції CalculateDoubleFactorial

static BigInteger CalculateDoubleFactorial(BigInteger n, int k, bool recursive)

{

int[] counters = Enumerable.Repeat(-1, k).ToArray(); // масив лічильників для заповнення масивів

int[][] arrayOfArrays = new int[k][]; // масив масивів

BigInteger nEven = 0;

BigInteger nOdd = 0;

if (n % 2 == 0)

{

// підрахунок парних чисел

for (int i = 2; i <= n; i += 2)

{

if (i % 2 == 0)

{

nEven++;

Лістинг 2.2 (б) – Програмний код функції CalculateDoubleFactorial (продовження)

}

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

}

// Створення порожніх масивив

for (int i = 0; i < k; i++)

{

arrayOfArrays[i] = new int[(int)((nEven + k - i - 1) / k)]; // створення масивів з розміром, достатнім для всіх чисел

}

// Заповнення масивів значениями

for (int i = 2, j = 1; (i <= (int)n) && (j <= (int)nEven); i += 2, j++)

{

int index = (j - 1) % k; // визначення номеру масива

counters[index]++; // стовбці

arrayOfArrays[index][counters[index]] = i; // запис i у відповідний масив

}

}

else

{

// підрахунок непарних чисел

for (int i = 1; i <= n; i += 2)

{

if (i % 2 != 0)

{

nOdd++;

}

}

// Створення порожніх масивив

for (int i = 0; i < k; i++)

{

arrayOfArrays[i] = new int[(int)((nOdd + k - i - 1) / k)]; // створення масивів з розміром, достатнім для всіх чисел

}

// Заповнення масивів значениями

for (int i = 1, j = 1; (i <= (int)n) && (j <= (int)nOdd); i += 2, j++)

{

int index = (j - 1) % k; // визначення номеру масива

Лістинг 2.2 (в) – Програмний код функції CalculateDoubleFactorial (продовження)

counters[index]++;

arrayOfArrays[index][counters[index]] = i; // запис i у відповідний масив

}

}

var tasks = new Task<BigInteger>[k]; // масив задач для потоків

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

BigInteger[] ansarr = new BigInteger[k];

// Створення завдання кожного рядка масиву, обчислення або рекурсивно, або ні

if (recursive)

{

for (int i = 0; i < k; i++)

{

int temp = i;

tasks[i] = Task.Run(() => ansarr[temp] = MultiplyRecursive(arrayOfArrays[temp])); // паралельне виконання функції

}

} else

{

for (int i = 0; i < k; i++)

{

int temp = i;

tasks[i] = Task.Run(() => ansarr[temp] = MultiplySimple(arrayOfArrays[temp])); паралельне виконання функції

}

}

Task.WaitAll(tasks); // очікування завершення потоків

BigInteger dFact = 1;

for (int i = 0; i < k; i++)

{

dFact \*= ansarr[i];

}

return dFact;

}

Програмний код функції рекурсивного множення елементів масиву MultiplyRecursive наведена у лістингу 2.3.

Лістинг 2.3 – Програмний код функції MultiplyRecursive

static BigInteger MultiplyRecursive(int[] arr)

{

int n = arr.Length;

if (n == 0)

{

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

return 1;

}

else if (n == 1)

{

return arr[0];

}

else

{

int mid = n / 2;

int[] left = arr.Take(mid).ToArray();

int[] right = arr.Skip(mid).ToArray();

return MultiplyRecursive(left) \* MultiplyRecursive(right);

}

}

Програмний код функції простого поелементного множення елементів масиву MultiplySimple наведена у лістингу 2.4.

Лістинг 2.4 – Програмний код функції MultiplySimple

static BigInteger MultiplySimple(int[] arr)

{

BigInteger res = 1;

for (int i = 0; i < arr.Length; i++)

{

res \*= arr[i];

}

return res;

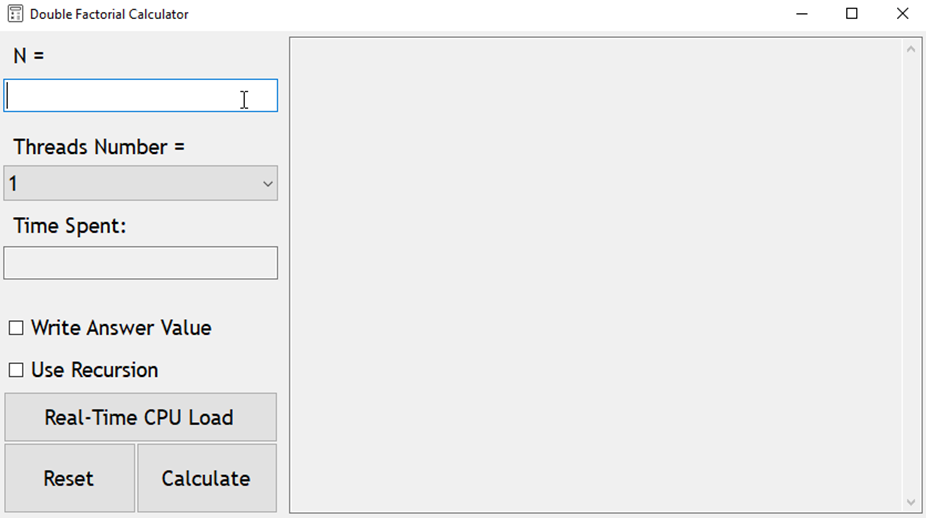
}

Повний код програми наведено у додатку А.

Для програми було розроблено графічний інтерфейс користувача в Windows Forms (див. рисунок 2.5). Ця форма має:

* поле введення для значення n;
* випадаючий список для вибору кількості потоків (це обмежено кількістю логічних ядер вашого процесора);
* перемикач для "використання рекурсії";
* перемикач для запису результату подвійного факторіалу.

Якщо другий перемикач увімкнено, то значення подвійного факторіалу буде збережено в полі відповіді. Також є таймер на C#, щоб побачити, скільки часу займе виконання цієї функції.



арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

Рисунок 2.5 – Графічний інтерфейс головного вікна програми

Засобами мови C# було також реалізовано відстеження навантаження на логічні ядра процесора в реальному часі (рисунок 2.6)

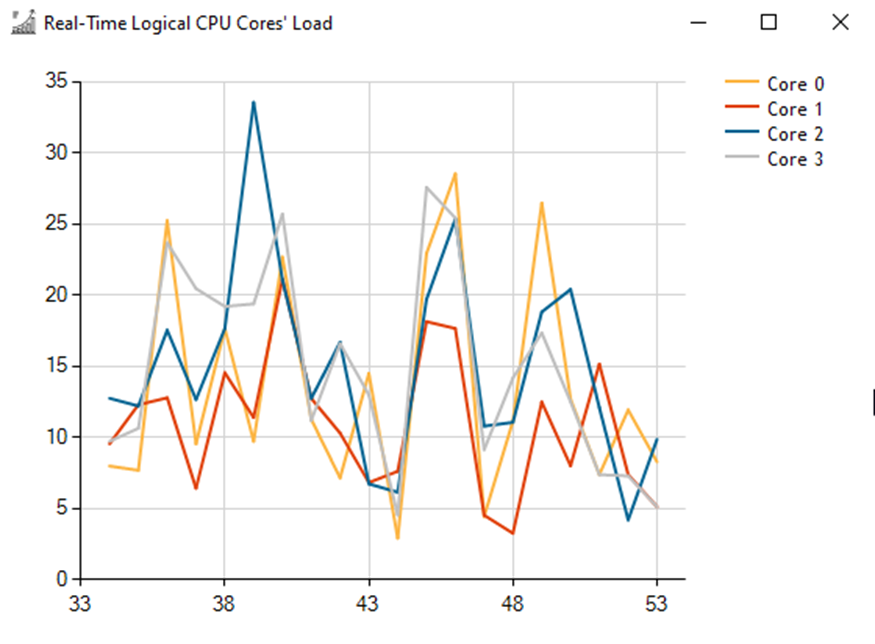


Рисунок 2.6 – Додаткова форма для відображення навантаження на логічні ядра процесора в реальному часі

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.02.РОПФ

Висновок за розділом «РЕКУРСИВНЕ ОБЧИСЛЕННЯ ПАРНОГО ФАКТОРІАЛУ

Під час проекту було здійснено реалізацію рекурсивного обчислення парного факторіалу в C#. Для прискорення процесу використовується багатопоточність, що дає можливість зменшити час обчислення факторіалу великих чисел. Рекурсивна природа гарантує просту та чітко структуровану реалізацію, яка підходить для обробки великих чисел без втрати точності. Завдяки технології Windows Forms ми змогли забезпечити дружній графічний інтерфейс користувача, що покращує використання програми.

Під час реалізації були розглянуті питання паралельних обчислень, такі як умови гонки, і здійснено відповідну синхронізацію потоків. Отримані експериментальні результати довели, що обраний підхід є одним з ефективних методів, завдяки якому можна дуже швидко і точно обчислити парний факторіал навіть з більшими значеннями вхідних даних.

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.02.ВРРОПФ

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

Висновок за розділом РЕКУРСИВНЕ ОБЧИСЛЕННЯ ПАРНОГО ФАКТОРІАЛУ

Літера

Аркушів

ЗКІ-21

3 АНАЛІТИЧНА ЧАСТИНА

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.03.АЧ

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

АНАЛІТИЧНА ЧАСТИНА

Літера

Аркушів

ЗКІ-21

3.1 Перевірка працездатності програми

Починаючи з перевірки правильності розробленого обчислення перед оцінкою продуктивності новоствореної програми, перевірка обчислення подвійного факторіалу на прикладі не надто великих чисел буде виконана за допомогою онлайн-калькулятора подвійного факторіалу [5]. Як ілюстрацію, було оцінено подвійний факторіал числа 500, оскільки його можна записати на 1 сторінці, і повний отриманий результат можна побачити на одній фігурі.

Правильність програми було протестовано в режимі одного потоку (Рисунок 3.1) та багатопотоковому режимі (Рисунок 3.2), щоб підтвердити, що немає проблем з гонкою даних.

.

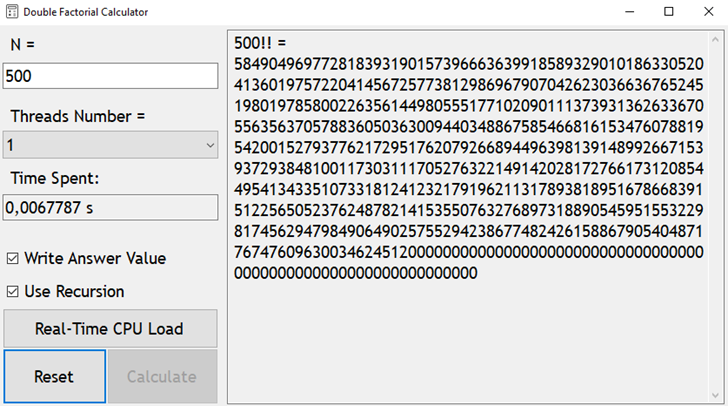


Рисунок 3.1 – Результат виконання програми обчислення 500!! в однопоточному режимі

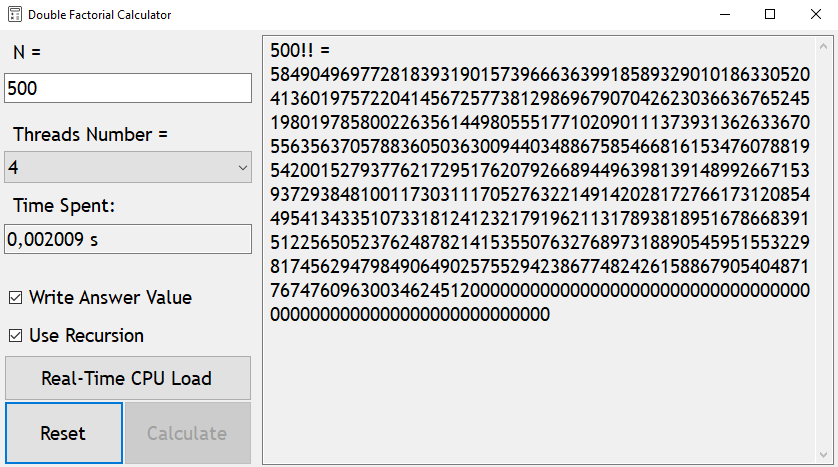
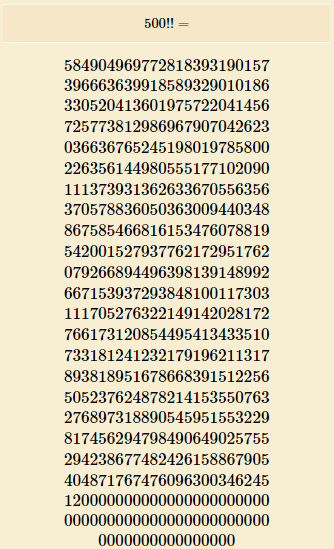


Рисунок 3.2 – Результат виконання програми обчислення 500!! з використанням 4 потоків



арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.03.АЧ

Рисунок 3.3 – Результат, отриманий від онлайн-калькулятора

Враховуючи результати реалізованої програми (Рисунки 3.1 – 3.2) та результати онлайн-калькулятора (Рисунок 3.3), можна з упевненістю сказати, що створена паралельна програма для рекурсивного обчислення подвійного факторіалу працює належним чином.

3.2 Виконання тестів продуктивності

Ми також оцінили продуктивність програмного забезпечення для обчислення подвійного факторіалу шляхом експериментування в режимі однопотокових і багатопотокових операцій. Це дозволяє обґрунтовано порівняти продуктивність та оцінити реальний виграш від паралелізму.

Для того, щоб надати більш надійні результати тестування, кожен тест повторювався у різні часи при тих самих вхідних параметрах. Повертаючись до керівних принципів, зауважте, що середній час виконання було взято з вихідних результатів, аби зменшити вплив випадкових відхилень, пов’язаних із навантаженням системи або змінами обчислювальної потужності.

Також продуктивність програми з використанням і без використання рекурсії обговорювалася окремо. Це дало можливість проаналізувати ефект покращення прибутків обраного обчислювального підходу з перспективою.

Результати експерименту наведено в таблиці 3.1, де порівнюються показники часу обчислення подвійного факторіалу в режимах одного потоку та 4 потоків з використанням рекурсії та без неї. Таблиця 3.2 надає інші вихідні дані для іншого значення вхідного параметра n.

Таблиця 3.1 – Результати тестування продуктивності при обчисленні

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.03.АЧ

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер досліду | 1 | 2 | 3 | 4 | 5 | Середній  час |
| Час виконання без рекурсії (сек) | | | | | | |
| 1 потік | 0,0572765 | 0,025391 | 0,0241853 | 0,0229166 | 0,0228424 | 0,0305224 |
| 4 потоки | 0,0146588 | 0,0182725 | 0,0142 | 0,0125498 | 0,0121324 | 0,0143627 |
| Час виконання з рекурсією (сек) | | | | | | |
| 1 потік | 0,0181912 | 0,01409 | 0,0143157 | 0,016004 | 0,0160102 | 0,0157222 |
| 4 потоки | 0,0188706 | 0,009888 | 0,0166038 | 0,0174323 | 0,0136066 | 0,0152803 |

Таблиця 3.2 – Результати тестування продуктивності при обчисленні

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер досліду | 1 | 2 | 3 | 4 | 5 | Середній  час |
| Час виконання без рекурсії (сек) | | | | | | |
| 1 потік | 2,5055269 | 2,6227649 | 2,5014206 | 2,6124272 | 2,5103138 | 2,5504907 |
| 4 потоки | 1,2866294 | 1,4932379 | 1,2319356 | 1,2345647 | 1,254009 | 1,3000753 |
| Час виконання з рекурсією (сек) | | | | | | |
| 1 потік | 1,2671184 | 1,4551464 | 1,2561768 | 1,2218416 | 1,2825761 | 1,2965719 |
| 4 потоки | 1,0595763 | 1,1312235 | 1,0564003 | 1,0659191 | 1,0871546 | 1,0800548 |

За результатами, наведеними у таблицях 3.1 – 3.2 можна визначити показники прискорення. Прискорення визначається за формулою , де  – час розв’язання задачі на одному потоці, , – час розв’язання тієї ж задачі на потоках.

Прискорення при обчисленні подвійного факторіала на 4 потоках без використання рекурсії дорівнює . При обчисленні прискорення . Аналогічним чином рахуються показники прискорення по іншим отриманим даним, результати заносяться в таблицю 3.3.

Таблиця 3.3 – Показники прискорення

|  |  |  |  |
| --- | --- | --- | --- |
| Обчислення подвійного факторіалу | Прискорення при обчисленні | Прискорення при обчисленні | Середнє прискорення |
| Без рекурсії |  |  | 2,04346 |
| З рекурсією |  |  | 1,1147 |

Маючи прискорення, логічним висновком є те, що паралелізація подвійного факторіала нерекурсивним алгоритмом з чотирма потоками досягає приблизно подвійної швидкості, як і у випадку з послідовною версією.

Для продуктивності рекурсивного розрахунку подвійного факторіала це невелике покращення, r≈1.1, проте паралелізація програми все ж таки була ефективною для збільшення швидкості.

Також з результатів можна зробити висновок, чи доцільно викликати вашу програму обчислення подвійного факторіала у рекурсивному вигляді при заданому n/2.

3.3 Аналіз показників надійності системи DDM COMA

Обговорюється приєднання резервних блоків пам'яті прив'язування до елементарної одношинної системи DDM (див. рис. 3.4).

Реплікація блоків кешу в системі з архітектурою COMA є дуже дорогою не лише у реалізації апаратного забезпечення, але й у реалізації підтримувального програмного забезпечення для відповідних протоколів. Проте це потенційно могло б підвищити відмовостійкість, надійність багатопроцесорної системи.

На рисунку 3.4 показана конфігурація паралельного з'єднання блоків кешу в базовій одношинній DDM.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.03.АЧ

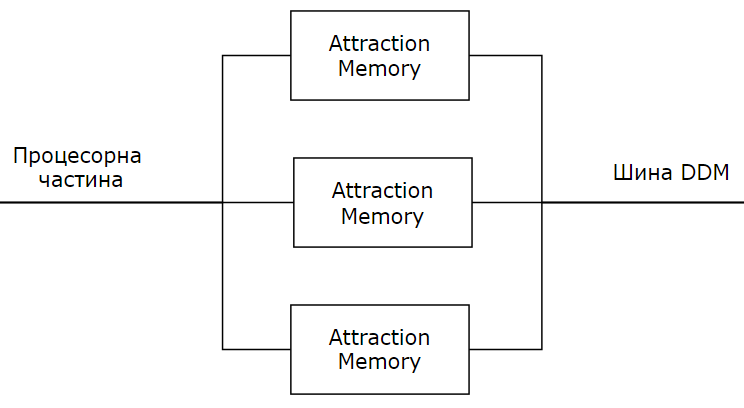


Рисунок 3.4 – Схема паралельного з’єднання блоків пам’яті

Припустимо, що кожен блок пам’яті притягання має однакову імовірність безвідмовної роботи .

Обчислимо імовірність безвідмовної роботи для схеми з трьох паралельно з’єднаних елементів за формулою:

При умові, що усі елементи схеми є однаковими та мають однакову імовірність безвідмовної роботи , формула для трьох однакових паралельно з’єднаних елементів може мати такий вигляд:

Таким чином:

Отже, швидкість системи без відмов може використовувати цей самий результат, який є дуже хорошим.

Наступний графік демонструє ймовірність безвідмовної роботи системи з плином часу. Припустимо, що коефіцієнт відмов λ кожного блоку кеш-пам'яті становить (1/годину). Ми розрахуємо цю величину за допомогою необхідної формули (ймовірність безвідмовної роботи як функція часу t) і вставимо таке значення у вже записану формулу ймовірності безвідмовної роботи трьох паралельних подібних елементів.

Отримаємо рівняння:

Використовуючи отримане рівняння будується графік залежності імовірності безвідмовної роботи системи від часу напрацювання (рисунок 3.5).

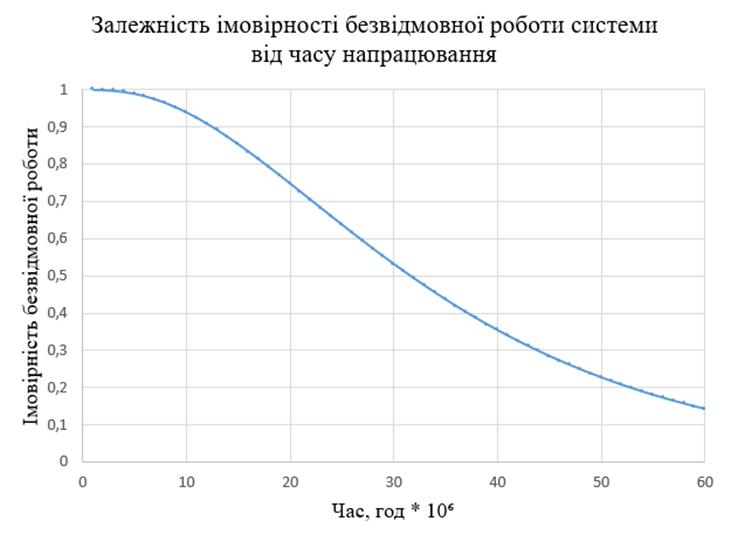


Рисунок 3.5 – Графік залежності імовірності безвідмовної роботи системи від часу напрацювання

З цього похідного графіку видно, що 90% ймовірності безвідмовної роботи досягається за 13 мільйонів годин, що еквівалентно 1484 рокам, а 50% ймовірності безвідмовної роботи системи досягається за 32 мільйони годин, що еквівалентно 3652 рокам, що відображає надійність системи.

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.03.АЧ

ВИСНОВКИ

У процесі кваліфікаційної роботи також було досліджено архітектуру комп'ютерних систем, основану на архітектурі пам'яті з кешуванням (COMA).

Перша частина зосереджена на теоретичних розглядах, пов'язаних з багатопроцесорними обчислювальними системами, їх типами та характеристиками, з прикладом архітектури COMA. Практичні реалізації представлені у вигляді таких додатків, як DDM (Data Diffusion Machine) та KSR-1 (Kendall Square Research).

Подвійне факторіальне декодування. Подвійний факторіал має деякі додаткові корисні властивості. ПЗ було розроблено на мові C#, і для створення графічного інтерфейсу користувача були використані форми Windows. Особлива увага приділялася проблемі гонок даних, властивих паралельним обчисленням. Реалізація програмного забезпечення включала можливість вимірювання часу виконання та можливість вибору певного режиму роботи (з рекурсією або без). Крім того, було реалізовано моніторинг навантаження в реальному часі для логічних ядер процесора.

Третя частина стосується тестування працездатності створеного програмного забезпечення та подальшого вимірювання його продуктивності. За отриманими емпіричними даними, максимальний коефіцієнт прискорення було досягнуто в різних режимах роботи програми. Крім того, було досліджено можливості відмовостійкості архітектури DDM COMA, у якій дублювання елементів кешу забезпечується їх паралелізацією. Було побудовано криву, що показує залежність між імовірністю безвідмовної роботи системи та часом її надійності, що дозволило оцінити можливість продовження використання COMA в умовах чітко визначеної надійності.

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.04.В

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

ВИСНОВКИ

Літера

Аркушів

ЗКІ-21

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. BigInteger Структура. URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.numerics.biginteger?view=net-7.0 (дата звернення 11.03.2023).

2. DDM – A Cache-Only Memory Architecture by Erik Hagersten, Anders Landlin and Seif Haridi. URL: https://www.academia.edu/30965970/DDM\_a\_cache\_only\_memory\_architecture (дата звернення 16.03.2023).

3. DDM – A Cache-Only Memory Architecture. URL: https://www.semanticscholar.org/paper/DDM-A-Cache-Only-Memory-Architecture-Hagersten-Landin/bf7b28933a570cf8c7beef3aaed1b58dd130fcc8 (дата звернення 17.03.2023).

4. Difference between Uniform Memory Access (UMA) and Non-uniform Memory Access (NUMA). URL: https://www.geeksforgeeks.org/difference-between-uniform-memory-access-uma-and-non-uniform-memory-access-numa/ (дата звернення 12.03.2023).

5. Double Factorial Calculator. URL: https://www.dcode.fr/double-factorial (дата звернення 09.03.2023).

6. KSR1 Multiprocessor: Analysis of Latency Hiding Techniques in a Sparse Solver. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=513e0a61ad2a55fc5f87606cfa55e3590f80ecaf (дата звернення 19.03.2023).

7. What is COMA?. URL: https://www.tutorialspoint.com/what-is-coma (дата звернення (16.03.2023).

8. Архитектура COMA. URL: http://db4.sbras.ru/elbib/data/show\_page.phtml?20+770 (дата звернення 14.03.2023).

9. Архітектура ccNUMA. URL: http://db4.sbras.ru/elbib/data/show\_page.phtml?20+768 (дата звернення 14.03.2023).

10. Архітектура nccNUMA. URL: http://db4.sbras.ru/elbib/data/show\_page.phtml?20+769 (дата звернення 14.03.2023).

11. Архітектура NUMA. URL: http://db4.sbras.ru/elbib/data/show\_page.phtml?20+767 (дата звернення 14.03.2023).

12. Когерентність кеш-пам’яті. URL: http://db4.sbras.ru/elbib/data/show\_page.phtml?77+773 (дата звернення 14.03.2023).

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.05.СВД

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Літера

Аркушів

ЗКІ-21

13. Конспект лекцій з дисципліни «Комп’ютерні системи» для студентів спеціальності «Кібербезпека». URL: https://nubip.edu.ua/sites/default/files/u34/konspekt\_ks.pdf (дата звернення 02.03.2023).

14. Посібник із класичних додатків (Windows Forms .NET). URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-7.0 (дата звернення 20.03.2023).

15. Потоки. Гонка даних та інші проблеми. URL: https://intuit.ru/studies/courses/10554/1092/lecture/27095 (дата звернення 21.03.2023).

16. Приклади організації систем паралельного оброблення інформації: GPU, мультипроцесори UMA, COMA, NUMA. URL: https://kvckr.me/avs/40.html (дата звернення 17.03.2023).

17. Симетричні (smp) багатопроцесорні ОС. Архітектура типу uma, coma, numa. URL: https://studfile.net/preview/9100610/page:30/ (дата звернення 15.03.2023).

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.05.СВД

ДОДАТКИ

Змн

Арк.

№\_Документа№ документа

Підпис

Дата

Аркуш

КНУ.ПК.123.25.06.06.Д

КНУ.ПК.123.22.01.Р

Розробив

Шнякін

Перевірив

Купін

Н.контроль

Купін

Затвердив

Купін

ДОДАТКИ

Літера

Аркушів

ЗКІ-21

Додаток А

Код головної форми паралельної програми для обчислення рекурсивного значення подвійного факторіалу:

using System;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Numerics;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DoubleFactorialProgram

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

this.Icon = new Icon("images/balance-sheet.ico");

int numThreads = Environment.ProcessorCount;

for (int proc = 1; proc <= numThreads; proc++)

{

comboBoxThreads.Items.Add(proc.ToString());

}

comboBoxThreads.SelectedIndex = 0;

}

private void buttonCalculate\_Click(object sender, EventArgs e)

{

buttonCalculate.Enabled = false;

this.Cursor = Cursors.WaitCursor;

// Обработка входного числа n

BigInteger number;

if (!BigInteger.TryParse(textBoxInputN.Text, dut number))

{

MessageBox.Show("Incorrect input value.", "Error!");

return;

}

// Обработка входного числа потоков

int threads;

int.TryParse(comboBoxThreads.Text, out threads);

var stopwatch = new Stopwatch(); // таймер

stopwatch.Start();

// Вычисление двойного факториала

BigInteger doubleFactorial = CalculateDoubleFactorial(number, threads, checkBoxRecursion.Checked);

stopwatch.Stop();

if (checkBoxWrite.Checked)

{

textBoxFactValue.Text = $"{number}!! = {doubleFactorial}";

}

textBoxTime.Text = $"\n{stopwatch.Elapsed.TotalSeconds} s";

this.Cursor = Cursors.Default;

}

static BigInteger CalculateDoubleFactorial(BigInteger n, int k, bool recursive)

{

int[] counters = Enumerable.Repeat(-1, k).ToArray(); // массив счетчиков для заполнения массивов

int[][] arrayOfArrays = new int[k][]; // массив массивов

BigInteger nEven = 0;

BigInteger nOdd = 0;

if (n % 2 == 0)

{

// подсчёт чётных чисел

for (int i = 2; i <= n; i += 2)

{

if (i % 2 == 0)

{

nEven++; }

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д

}

// Создаем пустые массивы

for (int i = 0; i < k; i++)

{

arrayOfArrays[i] = new int[(int)((nEven + k - i - 1) / k)]; // создаем массивы с размером, достаточным для всех чисел

}

// Заполняем массивы значениями

for (int i = 2, j = 1; (i <= (int)n) && (j <= (int)nEven); i += 2, j++)

{

int index = (j - 1) % k; // определяем номер массива

counters[index]++; // столбцы

arrayOfArrays[index][counters[index]] = i; // записываем i в соответствующий массив

}

}

else

{

// подсчёт нечётных чисел

for (int i = 1; i <= n; i += 2)

{

if (i % 2 != 0)

{

nOdd++;

}

}

// Создаем пустые массивы

for (int i = 0; i < k; i++)

{

arrayOfArrays[i] = new int[(int)((nOdd + k - i - 1) / k)]; // создаем массивы с размером, достаточным для всех чисел

}

// Заполняем массивы значениями

for (int i = 1, j = 1; (i <= (int)n) && (j <= (int)nOdd); i += 2, j++)

{

int index = (j - 1) % k; // определяем номер массива

counters[index]++;

arrayOfArrays[index][counters[index]] = i; // записываем i в соответствующий массив

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д

}

}

var tasks = new Task<BigInteger>[k]; // массив задач для потоков

BigInteger[] ansarr = new BigInteger[k];

// Создаем задачи для каждой строки массива, вычисляем либо рекурсивно, либо нет

if (recursive)

{

for (int i = 0; i < k; i++)

{

int temp = i;

tasks[i] = Task.Run(() => ansarr[temp] = MultiplyRecursive(arrayOfArrays[temp]));

}

} else

{

for (int i = 0; i < k; i++)

{

int temp = i;

tasks[i] = Task.Run(() => ansarr[temp] = MultiplySimple(arrayOfArrays[temp]));

}

}

Task.WaitAll(tasks);

BigInteger dFact = 1;

for (int i = 0; i < k; i++)

{

dFact \*= ansarr[i];

}

return dFact;

}

static BigInteger MultiplyRecursive(int[] arr)

{

int n = arr.Length;

if (n == 0)

{

return 1;

}

else if (n == 1)

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д

{

return arr[0]; }

else

{

int mid = n / 2;

int[] left = arr.Take(mid).ToArray();

int[] right = arr.Skip(mid).ToArray();

return MultiplyRecursive(left) \* MultiplyRecursive(right);

}

}

static BigInteger MultiplySimple(int[] arr)

{

BigInteger res = 1;

for (int i = 0; i < arr.Length; i++)

{

res \*= arr[i];

}

return res;

}

private void buttonReset\_Click(object sender, EventArgs e)

{

textBoxFactValue.Clear();

textBoxTime.Clear();

buttonCalculate.Enabled = true;

}

private Thread rtThread; // Поток для рисования графика производительности

private void buttonPerformance\_Click(object sender, EventArgs e)

{

// Проверяем, запущен ли поток

if (rtThread != null && rtThread.IsAlive)

{

// Если поток запущен, завершаем его

rtThread.Abort();

}

// Создаем и запускаем новый поток

rtThread = new Thread(() =>

{

Application.Run(new RealTimeForm());

});

rtThread.Start();

}

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д

private void Form1\_FormClosed(object sender, FormClosedEventArgs e) {

Application.Exit();

}

}

}

Додаток Б

Програмний код форми для відобрадення навантаження на логічні ядра процесора:

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace DoubleFactorialProgram

{

public partial class RealTimeForm : Form

{

private PerformanceCounter[] cpuCounters;

private List<Series> seriesList;

int currentCount = 0;

public RealTimeForm()

{

InitializeComponent();

this.Icon = new Icon("images/diagram.ico");

}

private void RealTimeForm\_Load(object sender, EventArgs e)

{

// Получение количества логических процессоров

int coreCount = Environment.ProcessorCount;

// Создание графиков для каждого ядра процессора

cpuCounters = new PerformanceCounter[coreCount];

seriesList = new List<Series>();

for (int i = 0; i < coreCount; i++)

{

PerformanceCounter counter = new PerformanceCounter("Processor", "% Processor Time", i.ToString());

cpuCounters[i] = counter;

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д

Series series = new Series("Core " + i);

series.ChartType = SeriesChartType.Line;

chart1.Series.Add(series);

// style chart

chart1.Series["Series1"].IsVisibleInLegend = false; // убираем лишний заголовок

chart1.ChartAreas[0].AxisX.MajorGrid.LineColor = Color.LightGray;

chart1.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.LightGray;

foreach (var s in chart1.Series)

{

s.BorderWidth = 2;

}

// end style

seriesList.Add(series);

}

// Настройка интервала таймера для обновления данных графиков

timer1.Interval = 1000; // обновление каждые 1 секунд

timer1.Start();

}

private void timer1\_Tick(object sender, EventArgs e)

{

// Получение текущей временной метки для использования на оси X

currentCount++;

// Обновление данных графиков для каждого ядра процессора

if (chart1 != null && seriesList != null)

{

for (int i = 0; i < cpuCounters.Length; i++)

{

double value = cpuCounters[i].NextValue();

seriesList[i].Points.AddXY(currentCount, value);

}

}

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д

// Ограничение количества точек на графиках

int maxPoints = 20; // максимальное количество точек на графике

foreach (var series in seriesList)

{

while (series.Points.Count > maxPoints)

{

series.Points.RemoveAt(0);

}

}

// Обновление области просмотра графика

chart1.ChartAreas[0].AxisX.ScaleView.Zoomable = true;

chart1.ResetAutoValues();

chart1.Invalidate();

}

private void RealTimeForm\_FormClosed(object sender, FormClosedEventArgs e)

{

timer1.Stop();

timer1.Dispose();

Application.ExitThread();

}

}

}

арк

№ документа

Підпис

Дата

Арк.

КНУ.ПК.123.25.06.06.Д