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»

Проектував	О.Э Гасанов
Керівник роботи	А. О. Сенько
Нормоконтроль	Д. І. Кузнєцов
Завідувач кафедри	А. І. Купін

Кривий Ріг
2026

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

Ступінь вищої освітибакалавр
Спеціальність123«Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

“____” _____ 20__ року

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«КОМП’ЮТЕРНА СИСТЕМА З ПАРАЛЕЛЬНОЮ АРХІТЕКТУРОЮ НА ОСНОВІ МУЛЬТИЯДЕРНИХ ПРОЦЕСОРІВ INTEL»

Спеціальність: Комп’ютерна інженерія

Освітній рівень: бакалавр

Виконав(ла): студент(ка) 4 курсу _____

Науковий керівник: _____

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Тема роботи: «Комп’ютерна система з паралельною архітектурою на основі мультиядерних процесорів Intel».

Вихідні дані: мультиядерні процесори Intel; C++17; OpenMP; методи оцінювання продуктивності; показники прискорення та ефективності.

Зміст роботи: теоретичний аналіз паралельних архітектур; аналіз мультиядерних процесорів Intel; проєктування системи; програмна реалізація; тестування та оцінювання результатів.

Перелік графічних матеріалів: класифікація паралельних архітектур; структура Intel CPU; архітектура прототипу; алгоритм тестування; графік прискорення.

Дата видачі завдання: _____ 2026 р. Строк подання роботи: _____ 2026 р.

АНОТАЦІЯ

У бакалаврській кваліфікаційній роботі досліджено комп’ютерну систему з паралельною архітектурою на основі мультиядерних процесорів Intel. Розглянуто теоретичні основи паралельних систем, організацію потоків і спільної пам’яті, методи оцінювання продуктивності, особливості фізичних і логічних ядер, кеш-пам’яті, Hyper-Threading, Turbo Boost, векторних розширень та OpenMP. У практичній частині спроєктовано і реалізовано застосунок C++17/OpenMP для порівняння послідовного та паралельного виконання. Система вимірює час, розраховує прискорення та ефективність, формує CSV-звіт і дозволяє оцінити масштабованість залежно від кількості потоків.

Ключові слова: паралельна архітектура, Intel, мультиядерний процесор, C++, OpenMP, потоки, кеш-пам’ять, прискорення, ефективність.

ABSTRACT

The bachelor qualification paper investigates a computer system with a parallel architecture based on multicore Intel processors. The work examines theoretical foundations of parallel systems, thread and shared-memory organization, performance evaluation methods, physical and logical cores, cache memory, Hyper-Threading, Turbo Boost, vector extensions, and OpenMP. The practical part presents a C++17/OpenMP prototype for comparing sequential and parallel execution. The system measures execution time, calculates speedup and efficiency, generates a CSV report, and evaluates scalability depending on the number of threads.

Keywords: parallel architecture, Intel, multicore processor, C++, OpenMP, threads, cache memory, speedup, efficiency.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Таблиця 0.1

Позначення	**Пояснення**
CPU	центральний процесор
Core	фізичне процесорне ядро
Thread	потік виконання
OpenMP	модель паралельного програмування для C/C++/Fortran
SIMD	одна інструкція над багатьма даними
AVX	векторні розширення Intel
S	прискорення
E	ефективність
CSV	текстовий формат табличних даних

ВСТУП

Актуальність теми бакалаврської кваліфікаційної роботи зумовлена тим, що сучасні комп'ютерні системи дедалі частіше досягають приросту продуктивності не за рахунок механічного збільшення тактової частоти, а завдяки ефективному використанню багатоядерності, потоків, кеш-пам'яті та векторних обчислень. Для комп'ютерної інженерії це означає необхідність розуміти не лише будову процесора, а й те, як програмна модель взаємодіє з апаратною архітектурою.

Мультиядерні процесори Intel є поширеною апаратною платформою для персональних комп'ютерів, робочих станцій і серверів. Вони поєднують фізичні ядра, логічні потоки, багаторівневу кеш-пам'ять, динамічне керування частотою, а в окремих поколіннях також гібридну архітектуру з продуктивними та енергоефективними ядрами. Саме тому їх доцільно розглядати як основу навчально-дослідної паралельної системи.

Метою роботи є дослідження принципів побудови комп'ютерної системи з паралельною архітектурою на основі мультиядерних процесорів Intel та розроблення програмної реалізації, що дозволяє оцінити вплив паралельного виконання на продуктивність типових обчислювальних задач.

Для досягнення мети поставлено такі завдання: проаналізувати теоретичні основи паралельних комп'ютерних архітектур; розглянути особливості мультиядерних процесорів Intel; сформувати вимоги до системи; спроєктувати програмно-апаратну структуру; реалізувати прототип мовою C++ з використанням OpenMP; провести тестування та оцінити прискорення й ефективність.

Об'єктом дослідження є комп'ютерні системи з паралельною архітектурою. Предметом дослідження є методи організації та оцінювання паралельних обчислень на мультиядерних процесорах Intel із використанням моделі спільної пам'яті.

Практична цінність роботи полягає у створенні робочого програмного прототипу, який виконує послідовні та паралельні версії декількох алгоритмів, вимірює час, розраховує прискорення й ефективність, а також формує CSV-файл із результатами. Такий підхід дозволяє перевірити теоретичні положення через експеримент.

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі викладено теоретичні засади. У другому розділі виконано аналіз Intel-архітектури та проєктування системи. У третьому розділі описано програмну реалізацію, тестування та результати.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПАРАЛЕЛЬНИХ КОМП'ЮТЕРНИХ АРХІТЕКТУР

1.1. Поняття та класифікація паралельних комп'ютерних систем

У контексті теми роботи аспект «поняття паралельної системи» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [9].

З інженерної точки зору «поняття паралельної системи» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [12].

Практична важливість елемента «класифікація Флінна» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [12].

Для навчально-дослідного прототипу «класифікація Флінна» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [13].

Окремо слід урахувати, що «модель спільної пам’яті» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [13].

У межах проєктної частини роботи «модель спільної пам’яті» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [9].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «SIMD та MIMD» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [9].

З позиції оцінювання результатів «SIMD та MIMD» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [12].

У контексті теми роботи аспект «рівні паралелізму» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [12].

З інженерної точки зору «рівні паралелізму» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [13].

Практична важливість елемента «відмінність паралелізму й конкурентності» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [13].

Для навчально-дослідного прототипу «відмінність паралелізму й конкурентності» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [9].

Рисунок 1.1 — Класифікація паралельних архітектур

Таблиця 1.1

Критерій	Типи	Значення для роботи
Потоки команд/даних	SISD, SIMD, MIMD	визначає характер паралельного виконання
Пам’ять	спільна, розподілена, гібридна	впливає на модель програмування
Рівень	інструкції, дані, потоки, задачі	пояснює джерела прискорення
Масштаб	ядро, процесор, вузол	визначає межі дослідження

1.2. Особливості багатоядерної архітектури сучасних процесорів

У контексті теми роботи аспект «фізичні ядра» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «фізичні ядра» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [4].

Практична важливість елемента «логічні потоки» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [4].

Для навчально-дослідного прототипу «логічні потоки» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [8].

Окремо слід урахувати, що «локальні та спільні кеші» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [8].

У межах проєктної частини роботи «локальні та спільні кеші» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [11].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «енергетичні обмеження» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [11].

З позиції оцінювання результатів «енергетичні обмеження» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [1].

У контексті теми роботи аспект «гібридні P-core та E-core» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «гібридні P-core та E-core» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [4].

Практична важливість елемента «пропускна здатність пам’яті» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні

частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [4].

Для навчально-дослідного прототипу «пропускна здатність пам’яті» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [8].

1.3. Організація потоків, процесів і спільної пам’яті в паралельних системах

У контексті теми роботи аспект «процеси і потоки» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [6].

З інженерної точки зору «процеси і потоки» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [7].

Практична важливість елемента «спільний адресний простір» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні

частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [7].

Для навчально-дослідного прототипу «спільний адресний простір» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [16].

Окремо слід урахувати, що «гонки даних» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [16].

У межах проєктної частини роботи «гонки даних» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [6].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «редукції» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [6].

З позиції оцінювання результатів «редукції» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [7].

У контексті теми роботи аспект «планування static і dynamic» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [7].

З інженерної точки зору «планування static і dynamic» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [16].

Практична важливість елемента «false sharing» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [16].

Для навчально-дослідного прототипу «false sharing» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

1.4. Методи оцінювання продуктивності паралельних обчислень

У контексті теми роботи аспект «час виконання» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної

комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [9].

З інженерної точки зору «час виконання» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [12].

Практична важливість елемента «прискорення S» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [12].

Для навчально-дослідного прототипу «прискорення S» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [13].

Окремо слід урахувати, що «ефективність E» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [13].

У межах проєктної частини роботи «ефективність E» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [14].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «закон Амдала» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [14].

З позиції оцінювання результатів «закон Амдала» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [9].

У контексті теми роботи аспект «масштабованість» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [9].

З інженерної точки зору «масштабованість» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [12].

Практична важливість елемента «контроль коректності результатів» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [12].

Для навчально-дослідного прототипу «контроль коректності результатів» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [13].

Таблиця 1.2

Показник	**Формула**	**Зміст**
Час виконання	T	тривалість виконання задачі
Прискорення	S = T1 / Tp	відносний виграш паралельної версії
Ефективність	E = S / p	ступінь використання потоків
Масштабованість	зміна S при зростанні p	якість розпаралелення

1.5. Переваги та обмеження використання паралельної архітектури

У контексті теми роботи аспект «скорочення часу обчислень» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [8].

З інженерної точки зору «скорочення часу обчислень» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру,

але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [9].

Практична важливість елемента «краще використання апаратних ресурсів» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [9].

Для навчально-дослідного прототипу «краще використання апаратних ресурсів» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [15].

Окремо слід урахувати, що «обмеження послідовної частини» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [15].

У межах проєктної частини роботи «обмеження послідовної частини» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [16].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «накладні витрати синхронізації» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме

така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [16].

З позиції оцінювання результатів «накладні витрати синхронізації» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [8].

У контексті теми роботи аспект «конкуренція за пам’ять» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [8].

З інженерної точки зору «конкуренція за пам’ять» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [9].

Практична важливість елемента «складність налагодження» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [9].

Для навчально-дослідного прототипу «складність налагодження» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому,

але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [15].

1.6. Постановка задач на кваліфікаційну роботу

У контексті теми роботи аспект «завдання теоретичного аналізу» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «завдання теоретичного аналізу» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [5].

Практична важливість елемента «завдання архітектурного аналізу Intel» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [5].

Для навчально-дослідного прототипу «завдання архітектурного аналізу Intel» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму

формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «формування вимог» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [6].

У межах проєктної частини роботи «формування вимог» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [8].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «проєктування структури системи» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [8].

З позиції оцінювання результатів «проєктування структури системи» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [1].

У контексті теми роботи аспект «реалізація C++/OpenMP» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий

ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «реалізація C++/OpenMP» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [5].

Практична важливість елемента «експериментальне тестування» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [5].

Для навчально-дослідного прототипу «експериментальне тестування» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Висновки до розділу 1

У першому розділі встановлено, що паралельна архітектура є багаторівневим явищем, яке охоплює апаратні, системні та програмні механізми.

Для теми роботи ключовими є модель спільної пам'яті, організація потоків, показники прискорення й ефективності, а також розуміння обмежень, що виникають через послідовні частини та накладні витрати.

Постановка задач у підрозділі 1.6 визначає подальший перехід до аналізу Intel-процесорів, проєктування системи та програмної реалізації.

РОЗДІЛ 2. АНАЛІЗ МУЛЬТИЯДЕРНИХ ПРОЦЕСОРІВ INTEL ТА ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

2.1. Архітектурні особливості мультиядерних процесорів Intel

У контексті теми роботи аспект «Intel Core і Xeon» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «Intel Core і Xeon» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [2].

Практична важливість елемента «P-cores та E-cores» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [2].

Для навчально-дослідного прототипу «P-cores та E-cores» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [3].

Окремо слід урахувати, що «виконавчі блоки» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [3].

У межах проєктної частини роботи «виконавчі блоки» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [4].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «апаратна багатопотоковість» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [4].

З позиції оцінювання результатів «апаратна багатопотоковість» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [8].

У контексті теми роботи аспект «векторні розширення» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [8].

З інженерної точки зору «векторні розширення» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [1].

Практична важливість елемента «керування частотою» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [1].

Для навчально-дослідного прототипу «керування частотою» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [2].

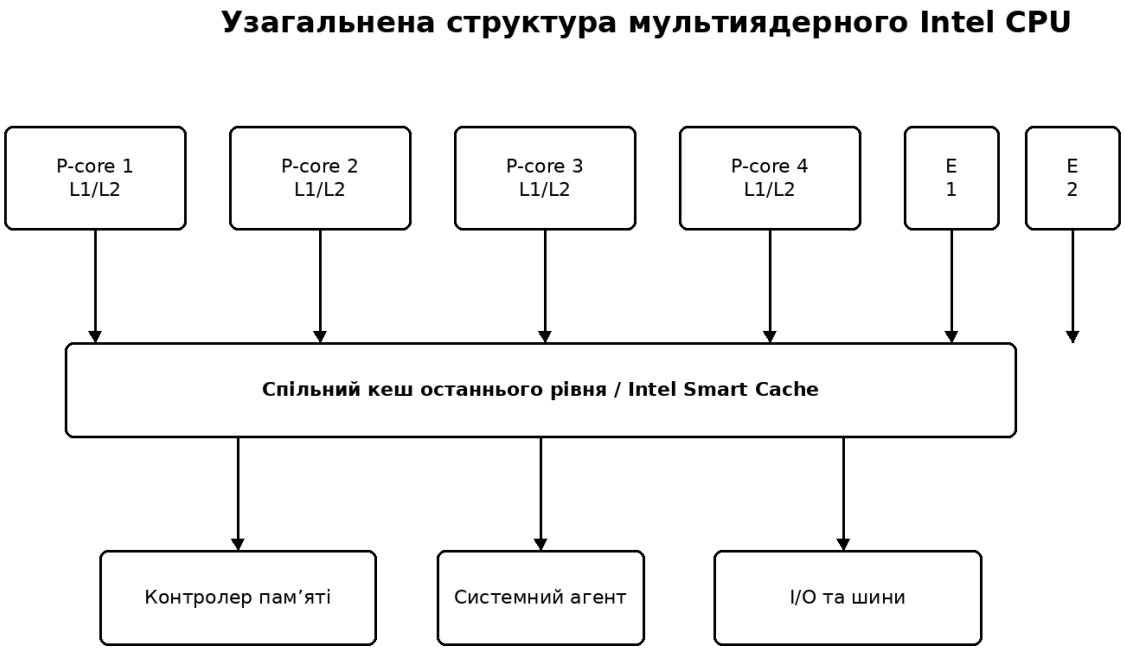

Рисунок 2.1 — Узагальнена структура мультиядерного процесора Intel

Таблиця 2.1

Компонент	Призначення	Вплив на паралельність
Фізичні ядра	виконання потоків команд	збільшують кількість одночасних обчислень
Логічні потоки	додаткові контексти	можуть покращити використання ядра
Кеш L1/L2/L3	швидкий доступ до даних	зменшує затримки пам’яті
Векторні блоки	SIMD-операції	підвищують щільність обчислень

2.2. Організація ядер, потоків, кеш-пам’яті та між’ядерної взаємодії

У контексті теми роботи аспект «ядро як виконавець команд» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий

ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «ядро як виконавець команд» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [8].

Практична важливість елемента «логічний процесор» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [8].

Для навчально-дослідного прототипу «логічний процесор» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [9].

Окремо слід урахувати, що «кеш L1 L2 L3» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам'яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [9].

У межах проєктної частини роботи «кеш L1 L2 L3» пов'язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою

контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [11].

Для спеціальності «Комп'ютерна інженерія» розгляд елемента «когерентність кешів» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам'яті [11].

З позиції оцінювання результатів «когерентність кешів» має бути пов'язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [1].

У контексті теми роботи аспект «локальність даних» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «локальність даних» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [8].

Практична важливість елемента «баланс навантаження» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися

нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [8].

Для навчально-дослідного прототипу «баланс навантаження» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [9].

2.3. Технології Intel Hyper-Threading, Turbo Boost та векторні розширення

У контексті теми роботи аспект «Hyper-Threading» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «Hyper-Threading» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [2].

Практична важливість елемента «Turbo Boost» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або

дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [2].

Для навчально-дослідного прототипу «Turbo Boost» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [3].

Окремо слід урахувати, що «AVX та AVX-512» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [3].

У межах проєктної частини роботи «AVX та AVX-512» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [5].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «векторизація» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [5].

З позиції оцінювання результатів «векторизація» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [1].

У контексті теми роботи аспект «динамічна частота» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [1].

З інженерної точки зору «динамічна частота» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [2].

Практична важливість елемента «залежність від моделі CPU» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [2].

Для навчально-дослідного прототипу «залежність від моделі CPU» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [3].

Таблиця 2.2

Технологія	Сутність	Практичне значення
Hyper-Threading	кілька логічних потоків на ядро	може підвищити завантаження ресурсів
Turbo Boost	динамічне підвищення частоти	залежить від температури та живлення

AVX/AVX-512	векторні операції	прискорення масивних обчислень
Thread Director	підказки планувальнику ОС	важливо для P-core/E-core систем

2.4. Вимоги до комп'ютерної системи з паралельною обробкою даних

У контексті теми роботи аспект «функціональні вимоги» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «функціональні вимоги» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «нефункціональні вимоги» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [6].

Для навчально-дослідного прототипу «нефункціональні вимоги» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову

точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [7].

Окремо слід урахувати, що «переносимість» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [7].

У межах проєктної частини роботи «переносимість» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [17].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «відтворюваність» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [17].

З позиції оцінювання результатів «відтворюваність» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [5].

У контексті теми роботи аспект «CSV-звіт» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий

ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «CSV-звіт» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «контроль коректності» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [6].

Для навчально-дослідного прототипу «контроль коректності» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [7].

Таблиця 2.3

Група вимог	**Зміст**	**Реалізація**
Функціональні	послідовна і паралельна версії	три тестові алгоритми
Продуктивність	вимірювання T, S, E	chrono та CSV
Переносимість	збірка у різних середовищах	C++17, OpenMP, CMake
Коректність	контроль результатів	checksum та порівняння

2.5. Проєктування структури програмно-апаратної системи

У контексті теми роботи аспект «межі системи» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект

пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «межі системи» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «модуль даних» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [6].

Для навчально-дослідного прототипу «модуль даних» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [7].

Окремо слід урахувати, що «модуль алгоритмів» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [7].

У межах проєктної частини роботи «модуль алгоритмів» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю,

формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [8].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «модуль метрик» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [8].

З позиції оцінювання результатів «модуль метрик» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [5].

У контексті теми роботи аспект «інтерфейс командного рядка» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «інтерфейс командного рядка» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «зв’язок із мультиядерним CPU» полягає в тому, що він пояснює різницю між теоретично можливим і фактично

отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [6].

Для навчально-дослідного прототипу «зв'язок із мультиядерним CPU» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [7].

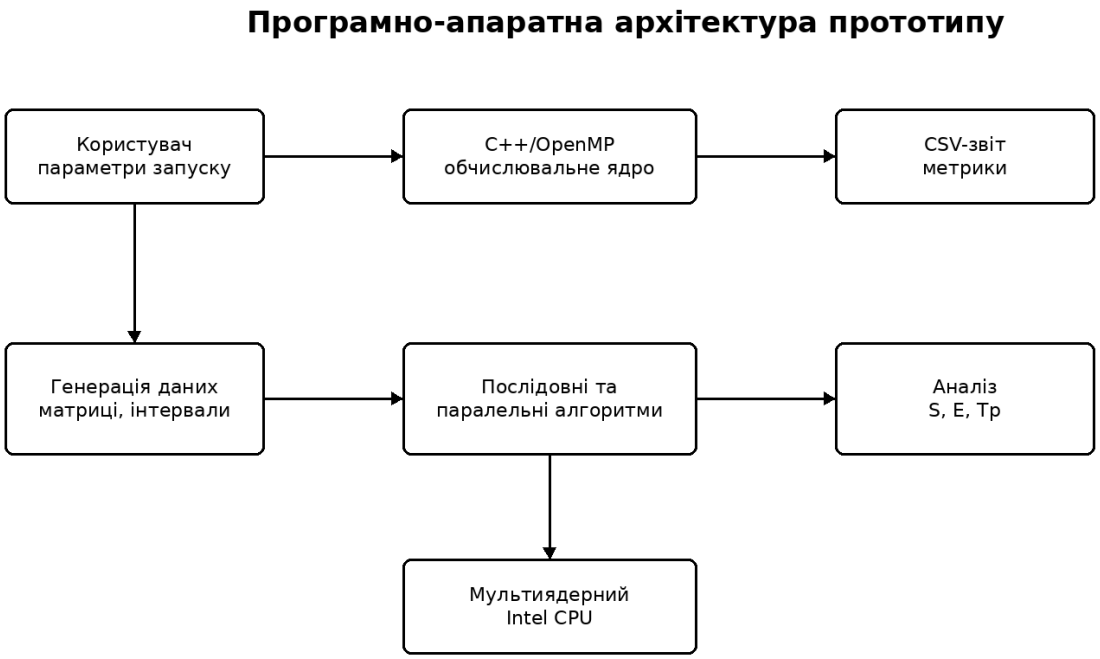

Рисунок 2.2 — Архітектура програмно-апаратної системи

Висновки до розділу 2

У другому розділі показано, що мультиядерні процесори Intel створюють апаратну основу для паралельного виконання завдяки фізичним і логічним ядрам, кеш-пам'яті, векторним блокам і динамічному керуванню частотою.

Сформовано вимоги до програмно-апаратної системи та запропоновано структуру прототипу, яка поєднує модуль даних, алгоритмічний модуль, модуль метрик і CSV-звіт.

Проєктні рішення обрано так, щоб програмна реалізація була відтворюваною, зрозумілою і безпосередньо пов’язаною з темою паралельної архітектури.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНОЇ СИСТЕМИ

3.1. Вибір засобів програмної реалізації

У контексті теми роботи аспект «C++17» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «C++17» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «C++17» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [7].

Для навчально-дослідного прототипу «OpenMP» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «OpenMP» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [7].

У межах проєктної частини роботи «OpenMP» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [17].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «std::chrono» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [7].

З позиції оцінювання результатів «std::chrono» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [17].

У контексті теми роботи аспект «std::chrono» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «CMake» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [17].

Практична важливість елемента «CMake» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [5].

Для навчально-дослідного прототипу «CMake» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «компіляторні оптимізації» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [5].

У межах проєктної частини роботи «компіляторні оптимізації» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [6].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «компіляторні оптимізації» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [7].

З позиції оцінювання результатів «переносимість» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [6].

У контексті теми роботи аспект «переносимість» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [7].

З інженерної точки зору «переносимість» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [17].

Таблиця 3.1

Засіб	Призначення	Причина вибору
C++17	реалізація алгоритмів	продуктивність і контроль ресурсів
OpenMP	паралельне виконання циклів	проста модель спільної пам’яті
std::chrono	вимірювання часу	стандартний механізм
CMake	збірка проєкту	автоматичний пошук OpenMP

3.2. Розробка алгоритмів послідовного та паралельного виконання обчислень

У контексті теми роботи аспект «множення матриць» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [6].

З інженерної точки зору «множення матриць» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [12].

Практична важливість елемента «множення матриць» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [13].

Для навчально-дослідного прототипу «чисельне інтегрування» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [12].

Окремо слід урахувати, що «чисельне інтегрування» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є

розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [13].

У межах проєктної частини роботи «чисельне інтегрування» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [14].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «підрахунок простих чисел» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [13].

З позиції оцінювання результатів «підрахунок простих чисел» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [14].

У контексті теми роботи аспект «підрахунок простих чисел» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [6].

З інженерної точки зору «послідовна версія» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не

забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [14].

Практична важливість елемента «послідовна версія» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [6].

Для навчально-дослідного прототипу «послідовна версія» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [12].

Окремо слід урахувати, що «паралельна версія» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [6].

У межах проєктної частини роботи «паралельна версія» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [12].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «паралельна версія» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така

організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [13].

З позиції оцінювання результатів «контрольна сума» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [12].

У контексті теми роботи аспект «контрольна сума» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [13].

З інженерної точки зору «контрольна сума» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [14].

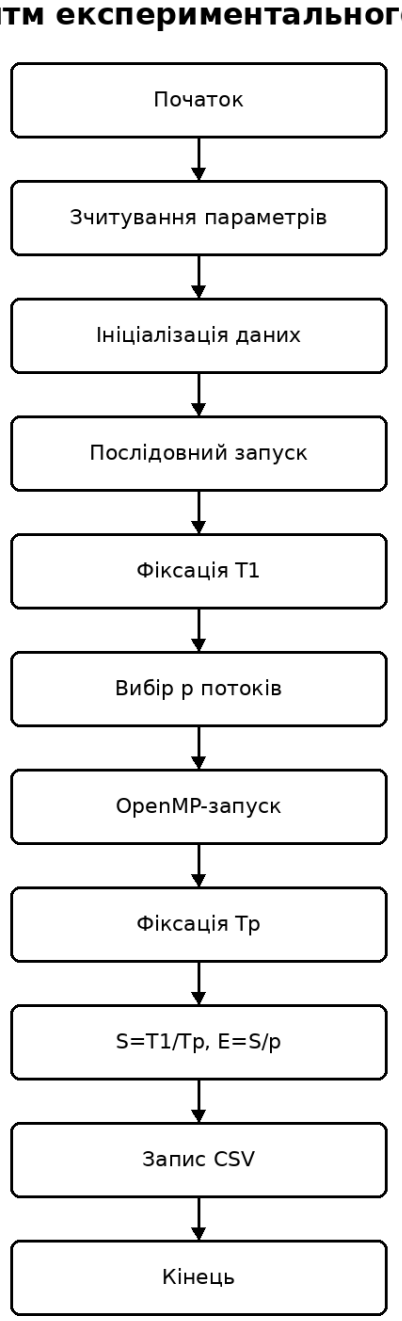

Рисунок 3.1 — Алгоритм експериментального тестування

Таблиця 3.2

Задача	**Тип навантаження**	**Спосіб паралелізації**
Множення матриць	масивні обчислення	розподіл рядків
Чисельне інтегрування	незалежні ітерації	parallel for reduction
Підрахунок простих чисел	нерівномірні ітерації	dynamic schedule

3.3. Реалізація програмної системи на основі C++ та OpenMP

У контексті теми роботи аспект «структура main.cpp» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «структура main.cpp» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «структура main.cpp» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [7].

Для навчально-дослідного прототипу «parallel for» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «parallel for» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам'яті та коректність

синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [7].

У межах проєктної частини роботи «parallel for» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [18].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «reduction» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [7].

З позиції оцінювання результатів «reduction» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [18].

У контексті теми роботи аспект «reduction» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «dynamic schedule» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він

розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [18].

Практична важливість елемента «dynamic schedule» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [5].

Для навчально-дослідного прототипу «dynamic schedule» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «визначення кількості потоків» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [5].

У межах проєктної частини роботи «визначення кількості потоків» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [6].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «визначення кількості потоків» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [7].

З позиції оцінювання результатів «збереження results.csv» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [6].

У контексті теми роботи аспект «збереження results.csv» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [7].

З інженерної точки зору «збереження results.csv» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [18].

Фрагмент паралельної редукції в OpenMP:

```
#pragma omp parallel for reduction(+:sum) schedule(static)
for (long long i = 0; i < steps; ++i) {
   const double x = h * (static_cast<double>(i) + 0.5);
   sum += 4.0 / (1.0 + x * x);
}
```

3.4. Проведення тестування роботи системи

У контексті теми роботи аспект «компіляція з -fopenmp» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект

пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «компіляція з -fopenmp» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [6].

Практична важливість елемента «компіляція з -fopenmp» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [8].

Для навчально-дослідного прототипу «параметри запуску» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «параметри запуску» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [8].

У межах проєктної частини роботи «параметри запуску» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю,

формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [17].

Для спеціальності «Комп'ютерна інженерія» розгляд елемента «порівняння T1 і Tp» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам'яті [8].

З позиції оцінювання результатів «порівняння T1 і Tp» має бути пов'язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [17].

У контексті теми роботи аспект «порівняння T1 і Tp» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [5].

З інженерної точки зору «перевірка контрольних значень» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [17].

Практична важливість елемента «перевірка контрольних значень» полягає в тому, що він пояснює різницю між теоретично можливим і фактично

отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [5].

Для навчально-дослідного прототипу «перевірка контрольних значень» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «умови відтворення» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [5].

У межах проєктної частини роботи «умови відтворення» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [6].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «умови відтворення» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [8].

З позиції оцінювання результатів «стабільність вимірювань» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що

паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [6].

У контексті теми роботи аспект «стабільність вимірювань» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [8].

З інженерної точки зору «стабільність вимірювань» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [17].

Таблиця 3.3

Параметр	**Значення контрольного запуску**
Розмір матриці	420 x 420
Кроки інтегрування	12 000 000
Межа для простих чисел	800 000
Кількість потоків	1, 2, 4, 8
Компіляція	g++ -std=c++17 -O2 -fopenmp

3.5. Аналіз результатів тестування та оцінювання продуктивності

У контексті теми роботи аспект «таблиця результатів» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп'ютерної системи. Для мультиядерних процесорів Intel цей аспект пов'язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам'яті та як вимірюється отриманий

ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [6].

З інженерної точки зору «таблиця результатів» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв'язок апаратної архітектури, OpenMP-директив і експериментальних метрик [8].

Практична важливість елемента «таблиця результатів» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [9].

Для навчально-дослідного прототипу «графік прискорення» є зручним об'єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [8].

Окремо слід урахувати, що «графік прискорення» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам'яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [9].

У межах проєктної частини роботи «графік прискорення» пов'язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою

контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [12].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «пояснення нелінійного масштабування» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [9].

З позиції оцінювання результатів «пояснення нелінійного масштабування» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [12].

У контексті теми роботи аспект «пояснення нелінійного масштабування» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [13].

З інженерної точки зору «ефективність потоків» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [12].

Практична важливість елемента «ефективність потоків» полягає в тому, що він пояснює різницю між теоретично можливим і фактично отриманим результатом. Навіть за наявності кількох ядер програма може масштабуватися

нерівномірно через накладні витрати, особливості кешування, послідовні частини або дисбаланс навантаження. Тому в роботі використано не лише опис, а й вимірювання часу виконання [13].

Для навчально-дослідного прототипу «ефективність потоків» є зручним об’єктом аналізу, оскільки його вплив можна показати на невеликому, але повністю робочому застосунку. Послідовна версія алгоритму формує базову точку порівняння, а паралельна версія демонструє, як змінюється час виконання після залучення кількох потоків [6].

Окремо слід урахувати, що «роль пам’яті» не гарантує автоматичного виграшу продуктивності. У паралельних системах важливими є розмір задачі, частка незалежних ітерацій, характер доступу до пам’яті та коректність синхронізації. Ці умови пояснюють, чому одна задача масштабується добре, а інша швидко досягає насичення [13].

У межах проєктної частини роботи «роль пам’яті» пов’язується з конкретними вимогами до системи: простотою запуску, переносимістю, формуванням CSV-звіту, можливістю зміни кількості потоків і перевіркою контрольних значень. Це дозволяє зробити програмну реалізацію не декоративним додатком, а інструментом дослідження [6].

Для спеціальності «Комп’ютерна інженерія» розгляд елемента «роль пам’яті» є важливим, тому що він поєднує апаратний і програмний рівні. Студент має пояснити не лише як написати код, а й чому саме така організація коду відповідає властивостям мультиядерного процесора та моделі спільної пам’яті [8].

З позиції оцінювання результатів «практичні висновки» має бути пов’язаний з кількісними показниками: часом виконання, прискоренням і ефективністю. Саме ці показники дозволяють обґрунтовано стверджувати, що паралельна реалізація є доцільною, а також пояснювати випадки, коли приріст продуктивності обмежений [6].

У контексті теми роботи аспект «практичні висновки» має розглядатися не як окремий термін, а як складова загальної логіки побудови паралельної комп’ютерної системи. Для мультиядерних процесорів Intel цей аспект пов’язаний із тим, як програмний код розподіляється між виконавчими ресурсами, як організовується доступ до пам’яті та як вимірюється отриманий ефект. Саме тому його аналіз потрібний для переходу від теоретичного опису до практичного прототипу [8].

З інженерної точки зору «практичні висновки» впливає на вибір алгоритму, структуру даних і спосіб організації потоків. Якщо цей фактор не враховувати, паралельна програма може мати формально правильну структуру, але не забезпечувати суттєвого прискорення. У кваліфікаційній роботі він розглядається через взаємозв’язок апаратної архітектури, OpenMP-директив і експериментальних метрик [9].

Таблиця 3.4

Задача	**Потоки**	**Час, с**	**S**	**E**
matrix_multiply	1	0.035003	1.000	1.000
matrix_multiply	2	0.023029	1.520	0.760
matrix_multiply	4	0.010682	3.277	0.819
matrix_multiply	8	0.008525	4.106	0.513
pi_integration	1	0.014703	1.000	1.000
pi_integration	2	0.007516	1.956	0.978
pi_integration	4	0.006333	2.322	0.580
pi_integration	8	0.003540	4.154	0.519
prime_count	1	0.073404	1.000	1.000
prime_count	2	0.037497	1.958	0.979
prime_count	4	0.024667	2.976	0.744
prime_count	8	0.011324	6.482	0.810

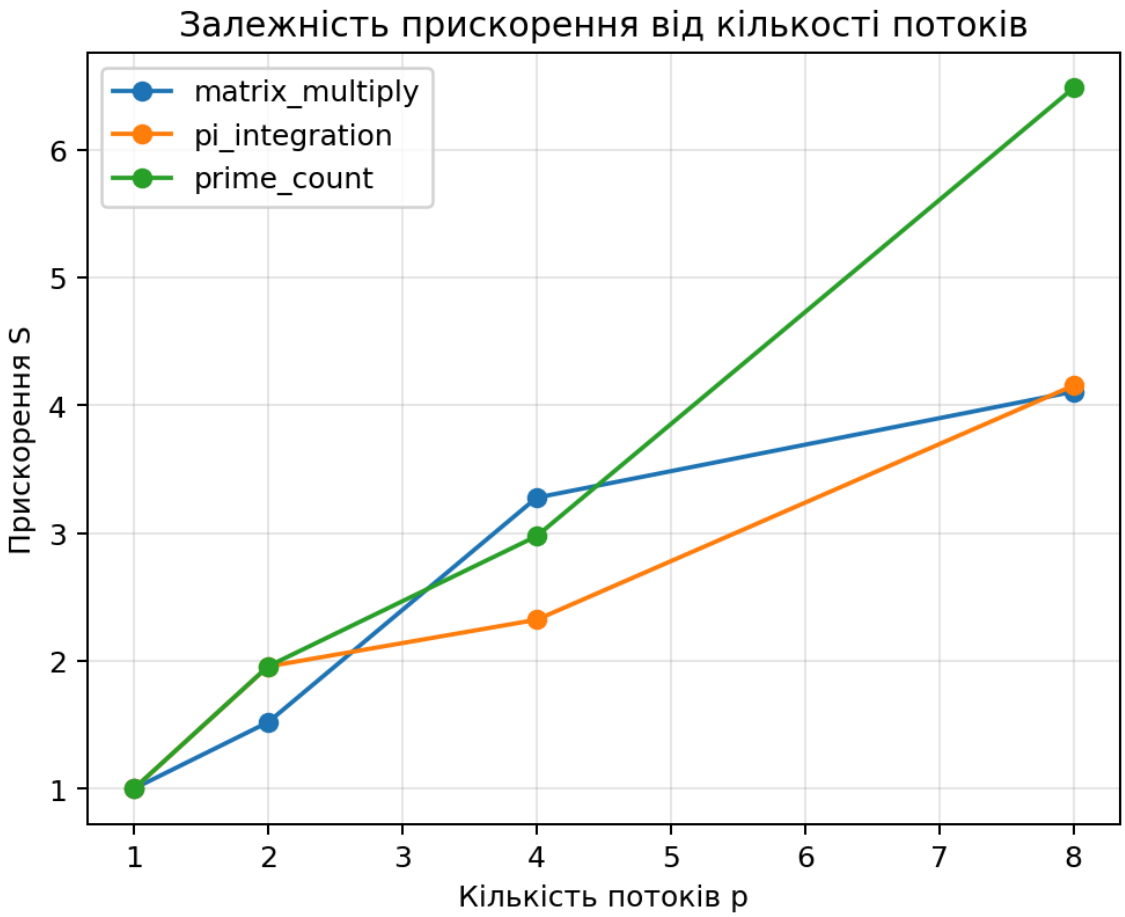

Рисунок 3.2 — Залежність прискорення від кількості потоків

Висновки до розділу 3

У третьому розділі реалізовано програмний прототип на C++17 з OpenMP, який порівнює послідовне та паралельне виконання кількох обчислювальних задач.

Тестування показало, що паралельна реалізація може забезпечити помітне прискорення, однак реальний результат залежить від структури алгоритму, планування потоків і обмежень пам’яті.

Розроблена система є придатною для навчально-дослідного аналізу мультиядерних процесорів і може бути розширена для подальших експериментів.

ВИСНОВКИ

У роботі досліджено комп’ютерну систему з паралельною архітектурою на основі мультиядерних процесорів Intel. Показано, що сучасна продуктивність визначається не лише кількістю ядер або тактовою частотою, а й здатністю програмного забезпечення ефективно використовувати апаратні ресурси.

У першому розділі розглянуто поняття та класифікацію паралельних систем, організацію потоків і спільної пам’яті, методи оцінювання продуктивності, переваги та обмеження паралельної архітектури. Доданий підрозділ 1.6 формалізує постановку задач на кваліфікаційну роботу.

У другому розділі проаналізовано архітектурні особливості процесорів Intel: фізичні й логічні ядра, кеш-пам’ять, Hyper-Threading, Turbo Boost, AVX-розширення та гібридну організацію P-core/E-core. На цій основі сформовано вимоги і структуру програмно-апаратної системи.

У третьому розділі реалізовано прототип мовою C++17 із використанням OpenMP. Програма порівнює послідовне й паралельне виконання для множення матриць, чисельного інтегрування та підрахунку простих чисел, формує results.csv і дає змогу оцінити масштабованість.

Результати підтверджують, що паралельна архітектура може забезпечити помітне прискорення для задач із достатньою кількістю незалежних операцій. Водночас ефективність залежить від структури алгоритму, локальності даних, накладних витрат, планування потоків і обмежень пам’яті.

Мету роботи досягнуто: виконано теоретичний аналіз, спроєктовано систему, створено програмну реалізацію та проведено оцінювання продуктивності. Розроблений прототип може бути використаний для подальших лабораторних досліджень мультиядерних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Intel. What Is Hyper-Threading? Intel Resources. 2026.

2. Intel. What Is Intel Turbo Boost Technology? Intel Resources. 2026.
3. Intel. Intel Advanced Vector Extensions 512 (Intel AVX-512). Product documentation. 2026.
4. Intel. What Is Performance Hybrid Architecture? Intel Support. 2026.
5. Intel. OpenMP Support in Intel oneAPI DPC++/C++ Compiler. Developer Guide and Reference. 2025.
6. OpenMP Architecture Review Board. OpenMP Application Programming Interface, Version 5.2. 2021.
7. Intel. OpenMP: Intel Advisor User Guide. 2024.
8. Intel. Managing Multi-core Performance. Intel oneMKL Developer Guide. 2025.
9. Hennessy J. L., Patterson D. A. Computer Architecture: A Quantitative Approach. 6th ed. Morgan Kaufmann, 2019.
10. Patterson D. A., Hennessy J. L. Computer Organization and Design RISC-V Edition. Morgan Kaufmann, 2021.
11. Stallings W. Computer Organization and Architecture: Designing for Performance. 11th ed. Pearson, 2022.
12. Quinn M. J. Parallel Programming in C with MPI and OpenMP. McGraw-Hill, 2004.
13. Grama A., Gupta A., Karypis G., Kumar V. Introduction to Parallel Computing. 2nd ed. Addison-Wesley, 2003.
14. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. MIT Press, 2022.
15. Sutter H. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. Dr. Dobb's Journal, 2005.
16. Williams A. C++ Concurrency in Action. 2nd ed. Manning, 2019.
17. Microsoft. OpenMP in Visual C++. Microsoft Learn. 2025.
18. Intel. Use the OpenMP Libraries. Intel oneAPI DPC++/C++ Compiler Documentation. 2026.

ДОДАТКИ

Додаток А. Структура програмного проєкту

```
intel_parallel_system_project/
├── CMakeLists.txt
├── README.md
├── src/
```

```
│   └── main.cpp
└── results/
    ├── sample_run.txt
    └── results.csv
```

Додаток Б. Інструкція запуску

Для запуску проєкту потрібно встановити компілятор C++ із підтримкою OpenMP. У Linux можна виконати: g++ -std=c++17 -O2 -fopenmp src/main.cpp -o intel_parallel_demo, після чого запустити ./intel_parallel_demo. Також доступна збірка через CMake: mkdir build; cd build; cmake ..; cmake --build .

Після запуску програма виводить результати в консоль і створює файл results.csv. У файлі містяться назва задачі, кількість потоків, час виконання, прискорення, ефективність і контрольне значення.

Додаток В. Фрагмент вихідного коду

Фрагмент реалізації паралельного чисельного інтегрування наведено у скороченому вигляді. У повному файлі main.cpp функція задає крок інтегрування, відкриває паралельну область OpenMP, розподіляє ітерації між потоками директивою parallel for reduction(+:sum), після чого повертає наближене значення інтеграла. Повний вихідний код подано в архіві програмної реалізації.