КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (33), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.33.1215
УДК 004.272.43:511.174

Чикунов Павло Олександрович
кандидат технічних наук, доцент, доцент кафедри програмної інженерії,
Національний університет "Одеська юридична академія", м. Одеса, Україна,
ORCID: 0000-0003-4959-7744
pavel@onua.edu.ua

Манаков Сергій Юрійович
кандидат технічних наук, доцент, доцент кафедри програмної інженерії,
Національний університет "Одеська юридична академія", м. Одеса, Україна,
ORCID: 0000-0001-5930-4592
manakov_serhii@onua.edu.ua

Трофименко Олена Григорівна
кандидат технічних наук, доцент, доцент кафедри програмної інженерії,
Національний університет "Одеська юридична академія", м. Одеса, Україна,
ORCID: 0000-0001-7626-0886
trofymenko@onua.edu.ua

ЕФЕКТИВНІСТЬ ПЛАТФОРМИ NVIDIA CUDA ДЛЯ СИСТЕМАТИЧНОГО ПОШУКУ ПРОСТИХ ЧИСЕЛ

Анотація. Стаття присвячена дослідженню ефективності використання платформи NVIDIA CUDA для систематичного пошуку простих чисел у діапазоні $1 \div 10^9$. Метою роботи є оптимізація процесів пошуку та тестування простих чисел із застосуванням GPGPU. Завданнями є програмна CPU- та CUDA-реалізація алгоритмів просіювання (решето Ератосфена, колісна факторизація, решето Аткіна, решето Сундарама), ймовірнісних тестів простоти (Мілера-Рабіна, Ферма, Соловея-Штрасена, Люка-Лемера), а також порівняння їх швидкодії та масштабованості, визначення переваг і обмежень GPGPU. Реалізовано алгоритми решета Ератосфена, Сундарама, колісної факторизації, тестів Мілера-Рабіна та Люка-Лемера для CPU і GPU з використанням Visual Studio та NVIDIA CUDA Toolkit. Проведено експерименти в діапазонах від мільйона до мільярда чисел із фіксацією часу пошуку, кількості знайдених простих чисел та максимального значення. Для оцінки системних характеристик застосовано Microsoft Concurrency Visualizer, що дозволив проаналізувати споживання ресурсів CPU, рівень синхронізації та ефективність розподілу навантажень. Авторами розроблено інтегральний показник продуктивності алгоритмів пошуку, який враховує пропускну здатність, ефективність паралелізації та синхронізаційні втрати. Результати показали значне скорочення часу пошуку у CUDA-реалізаціях для алгоритмів з високим паралелізмом. Решето Ератосфена забезпечує стабільне прискорення від 2 до 8 разів, колісна факторизація – до 32 разів, тест Мілера-Рабіна – до 3 разів. Водночас GPU-підхід виявив обмеження для послідовних алгоритмів: решето Сундарама працює повільніше до 3 разів, тест Люка-Лемера – у 4 рази. Профілювання виявило високий рівень синхронізації (понад 80%), що знижує ефективність та вказує на можливість подальшої оптимізації. Також зафіксовано зменшення навантаження на CPU в середньому на 20%, що відкриває перспективи створення гібридних обчислювальних систем із поєднанням ресурсів CPU та GPU. На основі дослідження сформульовано рекомендації щодо вибору алгоритмів та підходів до їх реалізації на GPU для високопродуктивних обчислень.

Ключові слова: високопродуктивні обчислення; прості числа; GPGPU; CUDA; C++; генерація псевдовипадкових чисел; алгоритми просіювання; паралельні обчислення; профілювання.

ВСТУП

Постановка проблеми. Актуальність дослідження процесів прискорення пошуку простих чисел зумовлена їхньою важливою роллю в сучасних інформаційних технологіях. Прості числа є основою

криптографічних алгоритмів, які забезпечують безпеку цифрових комунікацій, захист персональних даних, роботу блокчейн-мереж та електронної комерції [1]. Пошук великих простих стимулює розвиток нових математичних методів, алгоритмів і архітектур обчислювальних систем, орієнтованих на підвищення продуктивності [2, 3].

Високопродуктивні обчислення (High-Performance Computing, HPC) спрямовані на ефективне використання апаратних і програмних ресурсів для вирішення складних задач із великими обсягами даних [4]. Пошук простих чисел є класичним прикладом HPC, оскільки потребує інтенсивних обчислень і масштабування на багатоядерних GPU. Одним із напрямів HPC є GPGPU (General-Purpose computing on Graphics Processing Units), що забезпечує виконання обчислень на графічних процесорах, які мають сотні й тисячі ядер та швидку спільну пам'ять для паралельних потоків. Реалізація алгоритмів пошуку простих на GPU демонструє суттєве прискорення, порівняно з CPU-застосуваннями [5, 6]. Завдяки використанню спеціалізованих API, як-от NVIDIA CUDA [2, 7] та AMD ROCm [4], розробники отримують інструменти для організації паралельних обчислень і оптимізації використання ресурсів. Це дає змогу виконувати задачі, які раніше вимагали суперкомп'ютерів на сучасних графічних процесорах.

Аналіз останніх досліджень і публікацій. Інтерес дослідників до високопродуктивних обчислень для пошуку та тестування простих чисел з часом лише зростає. Для прискорення процесу пошуку простих чисел застосовують багатопотокові, паралельні та розподілені алгоритми, які виконуються на багатоядерних GPU. Більшість алгоритмів просіювання і тестів на простоту розроблені для послідовного виконання, тому мають низький ступінь природного паралелізму й потребують суттєвих модифікацій для інтеграції в HPC-системи. Розподілені та паралельні обчислення для пошуку великих простих чисел використовуються в онлайн-проєктах, зокрема метою проєкту GIMPS [1] є пошук великих простих чисел, а також перевірка фундаментальних гіпотез із теорії чисел. Дослідження [4] присвячено огляду останніх досягнень у сфері високопродуктивних обчислень, зокрема тенденцій до конвергенції HPC та штучного інтелекту, розвитку ексафлопсних систем та глобальних змін у суперкомп'ютерних інфраструктурах, які визначають нові стандарти продуктивності та енергоефективності. У роботі [8] розглянуто підходи до оптимізації алгоритмів просіювання простих чисел за допомогою кешування та адаптивного розподілу обчислень. Її автори дійшли висновку, що правильне налаштування параметрів алгоритму дозволяє значно скоротити час виконання на сучасних багатоядерних системах. У роботі [5] виконано оцінку ефективності GPGPU-реалізацій решета Ератосфена та решета Сундарама у діапазонах чисел до 10^9. Решето Ератосфена працювало найповільніше з усіх алгоритмів, що обумовлено послідовним характером ітерацій алгоритму. Решето Сундарама мало покращення продуктивності у діапазоні чисел $10^7 \div 10^9$, що пояснюється незалежністю ітерацій одна від одної. Дослідження [9] присвячено оптимізації алгоритмів Ератосфена та Сундарама з використанням стандарту MPI. Автори встановили, що у діапазоні $1 \div 10^4$ найкращим є решето Сундарама. Однак, у діапазоні $10^4 \div 10^5$ решето Сундарама виявило нестабільність та припинило працювати належним чином, на відміну від решета Ератосфена, яке функціонувало стабільно. У роботі [6] наведено результати спроби прискорення тестування чисел на простоту до 10^{20} з використанням GPGPU-системи з 448 обчислювальними ядрами, що дозволило зменшити час виконання тесту на два порядки і зменшити споживання електричної потужності, порівняно з CPU-рішеннями. Автори роботи [3] запропонували два паралельні алгоритми генерування простих чисел, засновані на статичній і динамічній стратегіях, які перевершили послідовний алгоритм на 72% і 67% відповідно. Проєкт CUDASieve [10] пропонує CUDA-реалізацію решета Ератосфена, яка виконує пошук простих чисел у діапазоні $38 \div 4 \cdot 10^9$ за 50 мс. Проєкт має високу ефективність використання GPU-пам'яті. У роботі [11] виконано дослідження ефективності решета Ератосфена у діапазоні $1 \div 10^8$. Її автори порівнювали програмні реалізації з використанням платформи NVIDIA CUDA у послідовному режимі мовою C++ та у паралельному режимі мовою C#. Ці експерименти показали прискорення у 8 разів процесу пошуку за допомогою NVIDIA CUDA, порівняно з послідовним режимом. У роботі [12] розроблено MPI-бібліотеку для AMD ROCm, яка забезпечує значне зростання пропускної комунікаційної здатності, порівняно з традиційними MPI-бібліотеками. Такий підхід дозволив покращити продуктивність GPGPU та забезпечити ефективну інтеграцію GPU у високопродуктивні обчислення. Дослідження [13] присвячено розробці інструмента для профілювання GPU-застосунків. Автори показали, як профілювання дозволяє оптимізувати продуктивність багатопотокових і багатоядерних обчислень, аналізувати вузькі місця та підвищувати ефективність паралельних і розподілених алгоритмів.

Проведений аналіз літературних джерел окреслив можливі перспективи вдосконалення наявних алгоритмів для систематичного пошуку простих чисел. По-перше, це стосується зменшення часової та обчислювальної складності алгоритмів через застосування сучасних математичних методів. А, по-друге, перспективним є використання графічних процесорів, які містять тисячі апаратних ядер і мають високу

пропускну здатність пам'яті при її обмеженому обсязі, використання API для організації GPGPU-обчислень.

Метою роботи є комплексний аналіз та оцінка ефективності прискорення процесів пошуку і тестування простих чисел із застосуванням технології GPGPU на платформі NVIDIA CUDA.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

1. Огляд методів пошуку простих чисел.

Для пошуку простих чисел зазвичай використовують алгоритми повного просіювання простих чисел, які гарантують той самий набір простих чисел у визначеному діапазоні, та ймовірнісні тести на простоту числа, які допускають малу ймовірність помилки. Оскільки класичні алгоритми мають послідовну природу, їх обчислювальна складність значно зростає зі збільшенням діапазону пошуку.

Класичним алгоритмом просіювання є решето Ератосфена [5, 8-9] з асимптотичною складністю $O(n \log\log n)$ та просторовою складністю $O(n)$. Класичне решето вдосконалюють прийомом «колісна факторизація» [8], асимптотична та просторова складність якого становить $O(n)$. Більш ефективним алгоритмом є решето Аткіна [8] з асимптотичною $O(n/\log\log n)$ та просторовою складністю $O(\sqrt{n})$) й решето Сундарама [5, 8-9], асимптотична складність якого $O(n \cdot \log n)$, а просторова – $O(n)$.

Найпростішим з імовірнісних тестів є тест Ферма [6, 14], який виконується у кілька ітерацій із різними псевдовипадковими значеннями, тому його асимптотична складність становить $O(k(\log n)^3)$, а просторова – $O(\log n)$. Більш надійним є тест Міллера-Рабіна [6, 14], який завдяки повторним перевіркам зменшує ризик хибнопозитивних результатів, з асимптотичною складністю $O(k \cdot \log^3 n)$ та просторовою $O(k \cdot \log n)$, де k – кількість перевірок. Тест Соловея-Штрасена [6, 15] базується на властивостях символу Якобі та має характеристики подібні до тесту Міллера-Рабіна.

Тест Люка-Лемера [1, 15] застосовують для пошуку простих чисел Мерсена виду $M_p=2^p-1$, де p – теж просте число (показник або експонента числа). Саме з його допомогою знайшли найбільш відоме число Мерсена $M_{136279841}$ (понад 41 млн цифр) у рамках проєкту GIMPS з використанням GPU NVIDIA A100/H100.

2. Огляд API GPGPU.

Компанії Intel, Microsoft, Google, AMD та NVIDIA розробили спеціалізоване програмно-апаратне забезпечення HPC [2], [4], [16], зокрема універсальні API для GPGPU-обчислень. Серед найпоширеніших – NVIDIA CUDA, OpenCL, Microsoft DirectCompute, ATI Stream та AMD ROCm, характеристики яких наведені у табл. 1.

Таблиця 1

Порівняння характеристик API GPGPU

Характеристика	ATI Stream Technology	AMD ROCm	NVIDIA CUDA
Рік запуску	2008	2016	2006
Статус та підтримка	Застаріла, припинена 2011 р.	Актуальна, активно оновлюється	Актуальна, активно оновлюється
Призначення	Наукові обчислення та обчислення загального призначення	Розподілені високопродуктивні обчислення, штучний інтелект, наукові обчислення, обчислення загального призначення	Розподілені високопродуктивні обчислення, штучний інтелект, наукові обчислення, обчислення загального призначення
Мова програмування	Brook+, OpenCL	C++, OpenCL, ML-фреймворки	CUDA C/C++, Python, ML-фреймворки
Кросплатформність коду	Переносимий OpenCL-код	Переносимий CUDA-код	Переносимий CUDA-код
Екосистема	Обмежена, слабка підтримка сторонніх інструментів	Відкрита, інтеграція з TensorFlow, PyTorch, Triton	Розвинена
Сімейства відеокарт	Radeon HD, Radeon R9, FirePro	Radeon VII, Radeon Instinct, MI Series	GeForce, Quadro, Tesla, A100/H100

Вибір API залежить від наявного програмно-апаратного забезпечення, зокрема відеокарти та мови програмування. У середовище розробки Visual Studio можна інтегрувати API від різних виробників, як-от: NVIDIA CUDA Toolkit, AMD ROCm SDK, Alea GPU, Atimesh Hybridizer, ILGPU. Це дослідження

виконувалося на відеокарті середнього рівня сімейства NVIDIA GeForce у середовищі розробки Visual Studio, тому вибір API CUDA є природним рішенням для реалізації та тестування відповідних алгоритмів. API CUDA підтримує C, C++, Fortran, Python, а також містить бібліотеку cuBLAS для високопродуктивної лінійної алгебри, cuFFT для швидких перетворень Фур'є, cuRAND для генерації псевдовипадкових чисел. Це дозволяє розробникам зосередитися на розпаралелюванні алгоритмів, а не на низькорівневій реалізації. CPU і GPU працюють як незалежні пристрої, де частина коду виконується на CPU (host-код), зокрема генерація псевдовипадкових чисел для імовірнісних тестів, керування потоком виконання програми та координація між процесами. Продуктивність обмежується обміном даними між пам'яттю CPU і GPU, тому потрібен коректний розподіл інтервалів між блоками та потоками.

3. Програмна реалізація алгоритмів пошуку та тестування простих чисел.

У дослідженні реалізовано детерміновані алгоритми просіювання (решето Ератосфена, метод колісної факторизації, решето Аткіна, решето Сундарама) та ймовірнісні тести простоти (Мілера-Рабіна, Ферма, Соловея-Штрасена), а також тест Люка-Лемера. Для кожного алгоритму створено CPU- та CUDA-реалізації. Програмну частину виконано у Visual Studio 2022 на C++ з використанням NVIDIA CUDA Toolkit: host-код на CPU відповідає за логіку програми, пам'ять та обмін даними, тоді як device-код реалізує паралельні обчислення на GPU.

На рис. 1 показано процес пошуку простих чисел: спочатку ініціалізується масив чисел та розбивається на інтервали для подальшої обробці у незалежних потоках CUDA. Далі вибирається алгоритм просіювання чи тест простоти й розподіляються обчислення між тисячами потоків, кожен з яких обробляє свій інтервал. На завершальному етапі результати (прості числа, їх кількість і найбільше число) передаються на CPU, звільняється GPU-пам'ять, після чого здійснюється верифікація результатів і профілювання у Concurrency Visualizer.

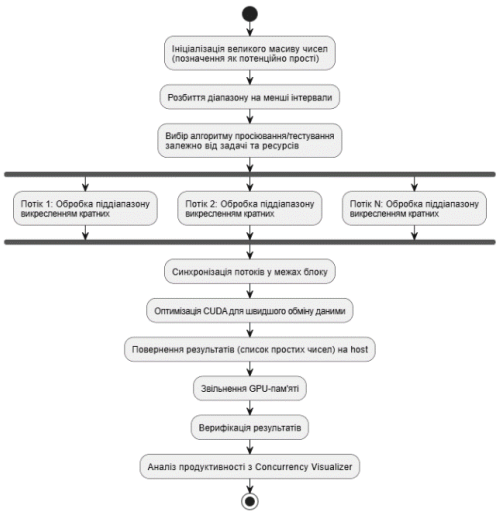

Рис. 1. UML-діаграма діяльності процесу пошуку простих чисел із використанням CUDA

4. Результати швидкодії та точності CPU- та CUDA-реалізацій алгоритмів.

Експериментальні дослідження кожного з алгоритмів проводилися для п'яти незалежних запусків у кожному з діапазонів від 1 до 10^6, 10^7, 10^8 та 10^9. Під час запусків фіксувався час виконання алгоритму, кількість знайдених простих чисел, пропускна здатність, відсоток завантаження CPU та GPU, показники синхронізації потоків, а також обсяг використаної пам'яті графічного процесора. Для аналізу результатів вибиралося медіанне значення, що дозволяло зменшити вплив випадкових коливань часу виконання та системних факторів. Спроба збільшити діапазон пошуку до 10^{10} призводила до значного зростання часу виконання (понад годину очікування з примусовим припиненням програми) або до виходу за межі цілого числа типу «unsigned long».Тестові запуски відбувалися на обчислювальній системі: центральний процесор AMD Ryzen 5 (6 фізичних ядер, 3.6 ГГц), оперативна пам'ять 16 ГБ DDR4-3200, відеокарта NVIDIA GeForce GTX 1070 (15 потокових мультипроцесорів по 128 CUDA-ядер, 8 ГБ GDDR5), ОС Windows 11, IDE Visual Studio 2022, пакет NVIDIA CUDA Toolkit, візуалізатор паралелізму Concurrency Visualizer. Результати виконання CPU- та CUDA-реалізацій алгоритмів наведено у табл. 2 і 3. Ймовірнісні тести на простоту мають пропуски простих чисел та хибнопозитивні значення, але їхня кількість менша за $3{,}5 \cdot 10^{-5}$%.

Таблиця 2

Порівняння швидкодії CPU- та CUDA-реалізацій алгоритмів пошуку простих чисел

Назва алгоритму	Діапазон пошуку	Час пошуку, с	
		CPU-реалізація	CUDA-реалізація
Решето Ератосфена	$1÷10^6$	0,910	0,389
	$1÷10^7$	8,640	1,457
	$1÷10^8$	96,120	12,004
	$1÷10^9$	946,540	123,263
Колісна факторизація	$1÷10^6$	1,010	0,456
	$1÷10^7$	8,835	0,700
	$1÷10^8$	91,634	3,182
	$1÷10^9$	955,188	29,437
Решето Аткіна	$1÷10^6$	0,017	0,851
	$1÷10^7$	0,486	1,248
	$1÷10^8$	4,375	5,684
	$1÷10^9$	53,249	51,132
Решето Сундарама	$1÷10^6$	0,423	1,578
	$1÷10^7$	4,812	12,76
	$1÷10^8$	54,574	125,184
	$1÷10^9$	661,319	1023,170
Тест на простоту Мілера-Рабіна	$1÷10^6$	0,269	0,385
	$1÷10^7$	3,048	1,429
	$1÷10^8$	34,405	12,057
	$1÷10^9$	397,361	118,568
Тест на простоту Ферма	$1÷10^6$	0,323	1,456
	$1÷10^7$	3,457	15,431
	$1÷10^8$	37,115	174,431
	$1÷10^9$	423,807	1818,147
Тест на простоту Соловея-Штрасена	$1÷10^6$	0,277	1,437
	$1÷10^7$	3,044	15,119
	$1÷10^8$	29,942	172,249
	$1÷10^9$	316,877	2681,231

Результати CUDA-реалізації решета Ератосфена показали суттєве зменшення часу виконання пошуку у широкому діапазоні від 57% до 88% при збільшенні діапазону пошуку. У найменшому діапазоні $1÷10^6$ прискорення становить 2,34 рази, у діапазоні $1÷10^7$ досягається прискорення у 5,94 рази, у діапазоні $1÷10^8$ спостерігається прискорення у 8,01 разів, а у найбільшому діапазоні $1÷10^9$ – прискорення у 7,68 разів. Тобто можна обґрунтовано стверджувати, що CUDA-реалізація решета Ератосфена має високу ефективність для всіх досліджуваних діапазонів.

Таблиця 3

Порівняння швидкодії CPU- та CUDA-реалізацій алгоритму пошуку простих чисел Мерсена

Діапазон показників *p*	Час виконання, с		Знайдених чисел	Максимальне число
	CPU-реалізація	CUDA-реалізація		
1÷100 000	2,42	9,631	308	$2^{99839}–1$
1÷200 000	8,71	37,204	567	$2^{199373}–1$
1÷300 000	18,86	70,717	819	$2^{299969}–1$
1÷400 000	32,86	123,120	1057	$2^{399937}–1$
1÷500 000	50,47	219,920	1293	$2^{499969}–1$

Результати CUDA-реалізації колісної факторизації були найбільш вражаючі серед усіх досліджуваних алгоритмів через значне зменшення часу виконання пошуку від 56% до 97% при збільшенні діапазону пошуку. У діапазоні $1÷10^6$ прискорення становило 2,21 рази, у діапазоні $1÷10^7$ – 12,62 разів, у діапазоні $1÷10^8$ – 28,81 разів, а у діапазоні $1÷10^9$ – рекордне прискорення у 32,46 рази. Тобто CUDA-реалізація колісної факторизації показала найкращі результати серед усіх досліджуваних алгоритмів.

Результати CUDA-реалізації решета Аткіна виявило сповільнення пошуку на малих діапазонах. Так, у діапазоні $1 \div 10^6$ час пошуку зріс у 50 разів, порівняно з CPU-реалізацією, що пояснюється накладними витратами на ініціалізацію масивів і передачу даних на GPU. Однак, при збільшенні діапазону ситуація покращувалася: у діапазоні $1 \div 10^7$ час на GPU перевищив CPU лише у 2,57 рази, а у $1 \div 10^8$ спостерігається зростання часу на GPU лише у 1,3 рази. При діапазоні $1 \div 10^9$ ефективність GPU майже зрівнялася з CPU – час пошуку на CUDA лише на 3,9% менший. Отже, CUDA-реалізація решета Аткіна ефективна лише для дуже великих обчислювальних задач, а для малих діапазонів накладні витрати переважають переваги паралелізації.

Результати CUDA-реалізації решета Сундарама показали протилежну тенденцію – CUDA-реалізація у всіх випадках поступається CPU-реалізації. У діапазоні $1 \div 10^6$ час пошуку зріс у 3,73 рази, у діапазоні $1 \div 10^7$ спостерігалося збільшення часу пошуку у 2,65 рази, у діапазоні $1 \div 10^8$ час – у 2.29 рази, а у діапазоні $1 \div 10^9$ – у 2,76 рази. Хоча при збільшенні діапазону різниця поступово зменшувалася. Отже, CUDA-реалізація решета Сундарама не є ефективною через особливості алгоритму, які не підходять для ефективної паралелізації на GPU.

Результати CUDA-реалізації тесту Ферма показали значне зростання часу пошуку на GPU, порівняно з CPU, в усіх діапазонах. У діапазоні $1 \div 10^6$ спостерігається збільшення часу у 4,5 рази, у $1 \div 10^7$ – у 4,46 рази, у $1 \div 10^8$ – у 4,71 рази, а у $1 \div 10^9$ – у 4,29 рази. Це свідчить про те, що алгоритм тесту Ферма не підходить для ефективної реалізації на GPU, оскільки його обчислювальна структура не дозволяє вигідно використовувати паралельні потоки.

Результати CUDA-реалізації тесту Мілера-Рабіна виявили спочатку сповільнення пошуку на 43% у діапазоні пошуку $1 \div 10^6$, що пояснюється накладними витратами на ініціалізацію масивів та передачу даних у GPU. Однак, далі ситуація покращилася: у діапазоні $1 \div 10^7$ час зменшився у 2,13 рази, у діапазоні $1 \div 10^8$ досягається прискорення у 2,85 рази, а у діапазоні $1 \div 10^9$ – прискорення у 3,35 рази. Тому можна стверджувати, що CUDA-реалізація тесту Мілера-Рабіна є ефективною для подібних обчислювальних задач.

Результати CUDA-реалізації тесту Соловея-Штрасена показали аналогічну тенденцію: час пошуку на GPU перевищив CPU-реалізацію на всіх діапазонах. У діапазоні $1 \div 10^6$ час зріс у 5,18 рази, у $1 \div 10^7$ – у 4,97 рази, у $1 \div 10^8$ – у 5,75 рази, а у $1 \div 10^9$ – у 8,47 разів. Така тенденція пояснюється інтенсивним використанням умовних перевірок та серійним характером обчислень у алгоритмі, що погано масштабується на GPU. Відтак, CUDA-реалізація тесту Соловея-Штрасена не є ефективною для великих обчислювальних задач.

Тест Люка-Лемера для простих чисел Мерсена показав суттєве збільшення часу виконання CUDA-реалізацій в середньому у 4 рази в усіх досліджуваних діапазонах. У діапазоні $1 \div 10^5$ час зріс у 3,98 рази, у діапазоні $1 \div 2 \cdot 10^5$ – у 4,27 рази, у діапазоні $1 \div 3 \cdot 10^5$ – у 3,75 рази. Отже, що CUDA-реалізація тесту Люка-Лемера не є ефективною через послідовну природу алгоритму та високу залежність між ітераціями. Порівняння часу виконання пошуку CPU- та CUDA-реалізацій для чотирьох діапазонів чисел дозволило дійти висновку про те, що ефективність CUDA-реалізацій решета Ератосфена та колісної факторизації, а також тесту Мілера-Рабіна поступово та стабільно покращується зі зростанням діапазону пошуку, тоді як решето Сундарама має протилежну тенденцію (рис. 2).

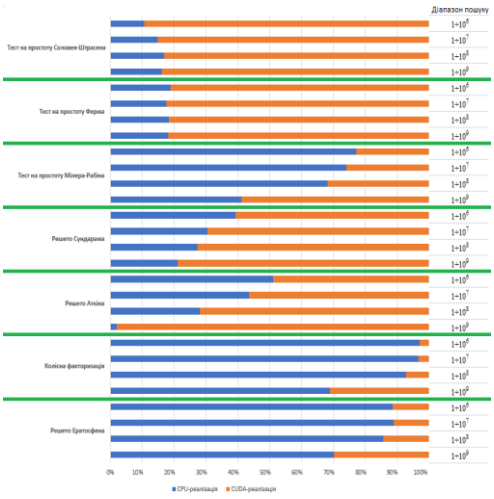

Рис. 2. Нормована гістограма відносних часток часу виконання CPU- та CUDA-реалізацій алгоритмів пошуку та тестування простих чисел

Аналіз результатів тесту Люка-Лемера (рис. 3) дозволив з'ясувати, що збільшення діапазону чисел практично не впливає на відносну ефективність процесу пошуку, CUDA-реалізація стабільно програє CPU в усіх діапазонах.

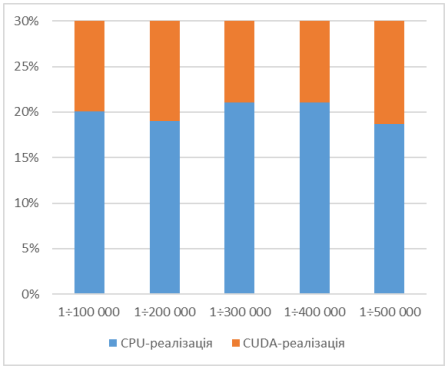

Рис. 3. Нормована гістограма відносних часток часу виконання CPU- та CUDA-реалізацій тесту простоти Люка-Лемера (для зручності стовпчик по осі Y обмежено 30%)

5. Профілювання та аналіз системних ресурсів.

Налагодження та оптимізація паралельних програм потребують візуалізатора Microsoft Concurrency Visualizer, який надає широкі можливості профілювання, перевірки продуктивності та моніторингу поведінки обчислювальної системи [13, 17]. Аналіз використання ресурсів дозволяє зрозуміти вплив конфігурації програми та налаштувань ОС на загальну продуктивність. Візуалізатор застосовує системні події Windows для моніторингу багатопотокових застосунків і формує графічні звіти, які спрощують виявлення вузьких місць алгоритмів. Трасування блоків CUDA-коду неможливе через відсутність доступу з Visual Studo до середовища виконання CUDA.

Детальне профілювання всіх CUDA- та CPU-реалізацій алгоритмів пошуку простих чисел засобами Concurrency Visualizer показало високий рівень міжпотокової синхронізації й міжпроцесорних обмінів даними CUDA-реалізацій алгоритмів пошуку, який перевищує 80% часу виконання, що погано впливає на ефективність пошуку. Аналіз метрик продуктивності (рис. 4) виявив закономірність: при порівнянні навантаження на центральний процесор у CUDA-реалізаціях та традиційних CPU-реалізаціях помітно зменшується навантаження на CPU, саме в CUDA-реалізаціях. Найбільш вдалою у цьому аспекті є CUDA-реалізація тесту простоти Люка-Лемера, де навантаження на CPU зменшено на 22%.

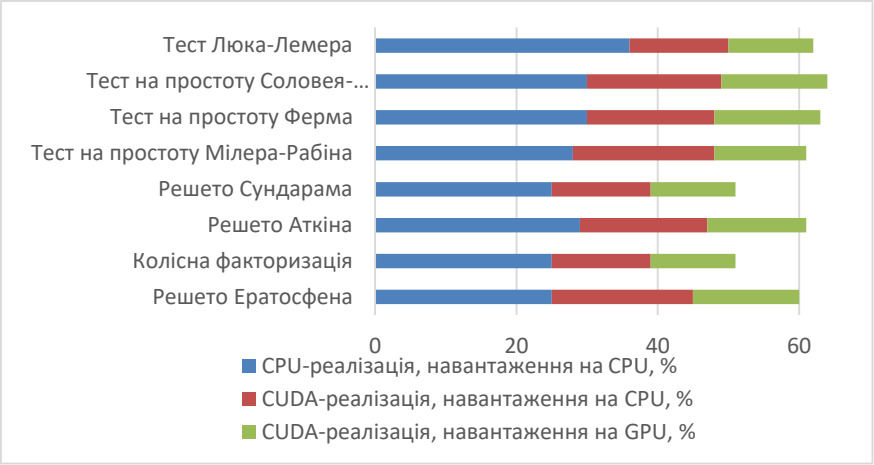

Рис. 4. Нормована гістограма відносного навантаження на процесори CPU- та CUDA-реалізацій алгоритмів пошуку та тестування простих чисел

6. Формалізована оцінка ефективності алгоритмів пошуку та тестування простих чисел. У ході дослідження використано авторський підхід до формалізованої оцінки ефективності, який дозволяє уніфіковано порівнювати різні алгоритми. Запропоновані критерії ґрунтуються на характеристиках часу, паралельності, варіативності результатів, інтенсивності обмінів та використання обчислювальних ресурсів.

Для формального аналізу запропоновано систему метрик, що дозволяє оцінити ефективність алгоритмів у контексті масштабованості, придатності до паралелізації, обчислювальної вартості та впливу синхронізації.

Коефіцієнт прискорення визначається як відношення часу виконання алгоритму на CPU до часу виконання на GPU:

$$S = \frac{T_{CPU}}{T_{GPU}}$$

Ефективність паралелізації визначається як відношення коефіцієнта прискорення до кількості активних потокових мультипроцесорів GPU:

$$E = \frac{S}{P}$$

Коефіцієнт синхронізаційних втрат визначається як відношення сумарного часу очікування, блокування, синхронізації, до загального часу виконання алгоритму на GPU:

$$K_{sync} = \frac{T_{sync}}{T_{GPU}}$$

Пропускна здатність алгоритму визначається як відношення кількості оброблених елементів (потенційно простих чисел) до часу виконання пошуку:

$$S = \frac{T_{CPU}}{T_{GPU}}$$

$$\Phi = \frac{N}{T}$$

Для ранжування алгоритмів за сукупністю експлуатаційних характеристик введено інтегральний показник якості, що дозволяє оцінити ефективність алгоритму, враховуючи фактичну швидкість роботи, що характеризується пропускною здатністю, втрати продуктивності на синхронізацію, а також ефективність обчислювальної архітектури:

$$Q = \Phi \cdot \left(1 - K_{sync}\right) \cdot E$$

У табл. 4 порівняно ефективності CPU- та CUDA-реалізацій алгоритмів для діапазону $1 \div 10^9$. Для кожного алгоритму наведено час виконання на CPU та GPU, коефіцієнт прискорення S, пропускну здатність Ф, ефективність паралелізації E та інтегральний показник якості Q.

Таблиця 4

Порівняння ефективності CPU- та GPU-реалізацій алгоритмів у діапазоні $1 \div 10^9$

Алгоритм	Час виконання CPU, с	Час виконання на GPU, с	S	Ф	E	Q
Решето Ератосфена	946,540	123,263	7,68	8,11	0,512	0,8300
Колісна факторизація	955,188	29,437	32,46	33,97	2,164	14,6950
Решето Аткіна	53,249	51,132	1,04	19,56	0,069	0,2700
Решето Сундарама	661,319	1023,170	0,65	0,98	0,043	0,0080
Тест Мілера-Рабіна	397,361	118,568	3,35	8,43	0,223	0,3760
Тест Ферма	423,807	1818,147	0,23	0,55	0,015	0,0017
Тест Соловея-Штрасена	316,877	2681,231	0,12	0,37	0,008	0,0006

Комплексний аналіз результатів дослідження показав, що паралелізація процесів систематичного пошуку простих чисел з використанням платформи CUDA забезпечує високу продуктивність, однак її ефективність критично залежить від природних характеристик алгоритмів. У ході експериментальних

досліджень встановлено, що використання GPU може як кардинально покращити продуктивність, так і значно погіршити її.

Найвищу ефективність на GPU показала колісна факторизація, яка має найбільший інтегральний показник якості. Решето Ератосфена має середній інтегральний показник, що робить його продуктивним, але менш ефективним, порівняно з колісною факторизацією**.** Решето Аткіна забезпечує майже однаковий час на CPU і GPU, тому інтегральний показник залишається низьким. Решето Сундарама, тести Ферма та Соловея-Штрасена мають низькі значення інтегрального показника, що свідчить про низьку ефективність їх паралельної реалізації. Наведені дані свідчать, що ефективність алгоритмів значно залежить від їх природної паралельності.

В результаті профілювання програмних реалізацій алгоритмів пошуку встановлено, що використання платформи CUDA для решета Ератосфена, колісної факторизації та тесту Мілера-Рабіна забезпечує значне зменшення часу пошуку та суттєве зменшення навантаження на центральний процесор. У найкращих випадках зафіксовано вражаючі результати: зменшення часу пошуку простих чисел на 88% для решета Ератосфена, на рекордні 97% для колісної факторизації та на 70% для тесту Мілера-Рабіна у великих діапазонах.

Окремо слід відзначити CUDA-реалізацію решета Аткіна, яка показала доволі суперечливі результати: хоча навантаження на CPU було знижено на 28%, загальний виграш у часі виявився помірним і склав лише близько 25% у великих діапазонах. Це свідчить про те, що структура алгоритму, яка потребує значної кількості умовних перевірок та нетривіальних операцій із залишками, не повністю відповідає архітектурі GPU, де оптимальною є масово-паралельна обробка однотипних операцій. Водночас на малих діапазонах CUDA-реалізація решета Аткіна практично не має переваг, а іноді навіть поступається CPU через накладні витрати на ініціалізацію потоків.

CUDA-реалізація тесту на простоту Ферма показала доволі непогану масштабованість: у великих діапазонах вона забезпечує скорочення часу перевірки на 45%, порівняно з CPU-реалізацією, при цьому навантаження на CPU вдалося зменшити майже на третину. Водночас на малих діапазонах прискорення є малопомітним через накладні витрати, однак, на відміну від решета Сундарама, тут не спостерігається критичного зниження ефективності.

CUDA-реалізація тесту Соловея-Штрасена виявилася менш вдалою: хоча навантаження на CPU скоротилося на 25%, загальний час виконання у великих діапазонах скоротився лише на 15%, а в малих діапазонах навіть зріс на 10-12%. Це пояснюється тим, що алгоритм містить складні операції з обчислення символів Якобі та степеневих залишків, які не повністю вдається ефективно розподілити між GPU-ядрами.

До невдалих результатів, які демонструють обмеження GPU-підходу, можна віднести факт, що CUDA-реалізація решета Сундарама демонструє значне збільшення часу пошуку простих чисел на 273% у малих діапазонах, хоча при збільшенні діапазону різниця поступово зменшується, проте все одно залишається невигідною. Можна зробити обґрунтоване припущення, що у діапазонах чисел понад 10^{10} ситуація може покращитися через кращу амортизацію накладних витрат.

Також неефективною виявилася CUDA-реалізація процесу тесту простоти Люка-Лемера, яка показала стабільне чотирикратне зростання часу виконання, порівняно зі звичайною CPU-реалізацією в усіх без винятку досліджуваних діапазонах. Можна впевнено припустити, що при подальшому збільшенні діапазону ситуація принципово не зміниться через послідовну природу алгоритму, тому подальше дослідження CUDA-реалізації тесту Люка-Лемера не є практично доцільним.

Профілювання програм за допомогою Microsoft Concurrency Visualizer показало помітне зменшення навантаження на CPU для всіх CUDA-реалізацій, зокрема для тесту Люка-Лемера навантаження на CPU зменшено на 22%, попри загальне погіршення продуктивності. Практичним результатом цього важливого факту є можливість розробки ефективних гібридних GPGPU-застосунків, які збалансовано розподіляють навантаження між CPU/GPU при виконанні складних розрахунків та можуть одночасно виконувати кілька незалежних обчислювальних задач.

Рівень міжпотокової синхронізації CUDA-реалізацій алгоритмів досягає критичних 80%, що однозначно свідчить про неоптимальну реалізацію міжпроцесорних і міжпотокових обмінів даними. Це вказує на значний потенціал для подальшого вдосконалення через оптимізацію схем синхронізації та перегляд архітектури паралельних обчислень.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Важливим практичним результатом дослідження є демонстрація того, що не всі математичні алгоритми однаково добре підходять для паралелізації на GPU-архітектурі. Успішність GPU-прискорення критично залежить від внутрішньої структури алгоритму, ступеня незалежності

обчислювальних операцій, регулярності доступу до пам’яті та мінімальної потреби у синхронізації між потоками.

Для генерації великих простих чисел рекомендовано використовувати GPU-прискорені реалізації решета Ератосфена або колісної факторизації для систематичного пошуку у великих діапазонах. Для тестування простоти окремих великих чисел тест Мілера-Рабіна на GPU є ефективним при обробленні великих пакетів кандидатів. Тест Люка-Лемера для чисел Мерсена слід залишити на CPU через його послідовну природу.

Можливими напрямками подальших досліджень є мінімізація міжпроцесорних та міжпотокових обмінів даними у CUDA-ядрах. Крім того, перспективним може бути концентрація на збільшенні ступеня паралелізму алгоритмів пошуку через розробку гібридних підходів, які поєднують обчислення на CPU та GPU. Важливим напрямком є запобігання процесам переповнення буфера за допомогою типів даних "biginteger" та "unsigned long long" мови C++. Вони забезпечать обробку діапазонів пошуку щонайменше до 10^{12}. Для аналізу продуктивності CUDA-ядр доцільно використовувати профайлери, як-от NVIDIA Nsight Compute та Nsight Systems, які дозволяють оптимізувати використання різних типів пам'яті, аналізувати ефективність ядер та виявляти вузькі місця в алгоритмах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Durant, L., Preda, M., Woltman, G., & Blosser, A. (2024). Discovery of the 52nd Mersenne prime 2^136279841-1. *Great Internet Mersenne Prime Search (GIMPS)*. https://www.mersenne.org/primes/press/M136279841.html
2. Bentz, J., Kumar, R., & Singh, A. (2025). NVIDIA Blackwell and NVIDIA CUDA 12.9 introduce family-specific architecture features. *NVIDIA Technical Blog*. https://developer.nvidia.com/blog/nvidia-blackwell-and-nvidia-cuda-12-9-introduce-family-specific-architecture-features/
3. Bahig, H. M., Hazber, M. A. G., Al-Utaibi, K., & Nassr, D. I. (2022). Efficient sequential and parallel prime sieve algorithms. *Symmetry*, *14*(12), 2527. https://doi.org/10.3390/sym14122527
4. Frąckiewicz, M. (2025). High-performance computing highlights (June-July 2025): Exascale era, HPC-AI convergence, and global supercomputing advances. *TechStock*. https://ts2.tech/en/high-performance-computing-highlights-june-july-2025-exascale-era-hpc-ai-convergence-and-global-supercomputing-advances/
5. Månsson, J. (2021). *Comparative study of CPU and GPGPU implementations of the sieves of Eratosthenes, Sundaram and Atkin* [Master’s thesis, Blekinge Institute of Technology]. DiVA. https://www.diva-portal.org/smash/get/diva2:1531686/FULLTEXT01.pdf
6. Asaduzzaman, A., Maiti, A., & Yip, C. (2014). Fast effective deterministic primality test using CUDA/GPGPU. *International Journal of Computer Applications*, *98*(11), 37-41. https://doi.org/10.5120/17234-7625
7. NVIDIA. (2025). *CUDA C++ best practices guide 12.9*. NVIDIA Developer Documentation. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
8. Ghidarcea, M., & Popescu, D. (2024). Prime time tactics: Sieve tweaks and boosters. *Algorithms*, *17*(7), 291. https://doi.org/10.3390/a17070291
9. Naik, H., Mishra, M., Routray, G., & Behera, M. (2020). Performance optimization by integrating memoization and MPI_Info object for sieve of prime numbers. *International Journal of Computer Applications*, *177*(47), 34-38. https://doi.org/10.5120/ijca2020920114
10. Seizert, C., Garland, M., & Temam, O. (2018). A highly efficient multi-GPU implementation of the sieve of Eratosthenes. *Journal of Parallel and Distributed Computing*, *113*, 90-100. https://doi.org/10.1016/j.jpdc.2017.10.016
11. Chykunov, P., & Chernetskyi, V. (2023). Organization of prime number search process using NVIDIA CUDA. In *Modern technologies in power engineering, electromechanics, control systems and mechanical engineering: Proceedings of the VI All-Ukrainian Scientific-Practical Internet Conference* (pp. 40-41). Kharkiv. https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi
12. Shafie Khorassani, K., Hashmi, J., Chu, C. H., Chen, C. C., Subramoni, H., & Panda, D. K. (2021). Designing a ROCm-aware MPI library for AMD GPUs: Early experiences. In *High Performance Computing* (Vol. 12728). Springer. https://doi.org/10.1007/978-3-030-78713-4_7
13. Shama, E., & Grant, R. (2025). NAV: A comparative analysis tool for Nsight Systems GPU traces. In *2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)* (pp. 536-543). IEEE. https://doi.org/10.1109/PDP66500.2025.00082

14. Massimo, J. (2020). *An analysis of primality testing and its use in cryptographic applications* [Doctoral dissertation, Royal Holloway, University of London]. https://pure.royalholloway.ac.uk/ws/portalfiles/portal/39023193/2020MassimoJPhD.pdf
15. Chykunov, P. O., Manakov, S. Yu., & Trofymenko, O. H.(2025).Efficiency of algorithms for systematic search of prime numbers. *Scientific Works of Donetsk National Technical University. Series: Problems of Modeling and Design Automation*,*2*(22),146-154.https://doi.org/10.31474/2074-7888-2025-2-22-146-154
16. Luo, W., Chen, X., Wang, Y., & Li, Z. (2025). Dissecting the NVIDIA Hopper architecture through microbenchmarking and multiple level analysis. *arXiv*. https://doi.org/10.48550/arXiv.2501.12084
17. Chykunov, P., & Kotov, A. (2023). Profiling multithreaded programs in the Visualizer Concurrency. In *Modern technologies in power engineering, electromechanics, control systems and mechanical engineering: Proceedings of the VI All-Ukrainian Scientific-Practical Internet Conference* (pp. 36-38). Kharkiv. https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (33), 2026
ISSN 2663 – 4023

Pavlo Chykunov
PhD, Associate Professor, Associate Professor at the Department of Software Engineering,
National University "Odesa Law Academy", Odesa, Ukraine,
ORCID: 0000-0003-4959-7744
pavel@onua.edu.ua

Serhii Manakov
PhD, Associate Professor, Associate Professor at the Department of Software Engineering,
National University "Odessa Law Academy", Odesa, Ukraine,
ORCID: 0000-0001-5930-4592
manakov_serhii@onua.edu.ua

Olena G. Trofymenko
PhD, Associate Professor, Associate Professor at the Department of Software Engineering,
National University "Odessa Law Academy", Odesa, Ukraine,
ORCID: 0000-0001-7626-0886
trofymenko@onua.edu.ua

EFFICIENCY OF THE NVIDIA CUDA PLATFORM FOR SYSTEMATIC PRIME NUMBER SEARCH

Abstract. The article is devoted to the study of the efficiency of using the NVIDIA CUDA platform for systematic search for prime numbers in the range $1 \div 10^9$. The aim of this study is to optimize the processes of searching and testing prime numbers using GPGPU. The tasks are CPU and CUDA software implementation of sieving algorithms (sieve of Eratosthenes, wheel factorization, Atkin sieve, Sundaram sieve), probabilistic tests of primality (Miller-Rabin, Fermat, Solovay-Strassen, Lucas-Lehmer), as well as comparing their speed and scalability, determining the advantages and limitations of GPGPU. The algorithms of the sieve of Eratosthenes, Sundaram, wheel factorization, Miller-Rabin and Luc-Lemaire tests for CPU and GPU are implemented using Visual Studio and NVIDIA CUDA Toolkit. Experiments were conducted on ranges from one million to one billion numbers with search time, the number of primes found, and the maximum value recorded. Microsoft Concurrency Visualizer was used to assess system characteristics, which allowed analyzing CPU resource consumption, synchronization level, and load distribution efficiency. The authors developed an integral performance indicator for search algorithms that considers bandwidth, parallelization efficiency, and synchronization losses. The results showed a significant reduction in search time in CUDA implementations for algorithms with high parallelism. The Sieve of Eratosthenes provides a stable acceleration of 2 to 8 times, wheel factorization – up to 32 times, and the Miller-Rabin test – up to 3 times. At the same time, the GPU approach revealed limitations for sequential algorithms: the Sundaram sieve works up to 3 times slower, the Luc-Lemaire test – up to 4 times. Profiling revealed a high level of synchronization (over 80%), which reduces efficiency and indicates the possibility of further optimization. The results also showed an average decrease in CPU load of 20%, which opens prospects for creating hybrid computing systems with a combination of CPU and GPU resources. Based on the study, recommendations were formulated for the selection of algorithms and approaches to their implementation on GPUs for high-performance computing.

Keywords: high-performance computing; prime numbers; GPGPU; CUDA; C++; pseudorandom number generation; sieving algorithms; parallel computing; profiling.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Durant, L., Preda, M., Woltman, G., & Blosser, A. (2024). Discovery of the 52nd Mersenne prime 2^136279841-1. *Great Internet Mersenne Prime Search (GIMPS)*. https://www.mersenne.org/primes/press/M136279841.html
2. Bentz, J., Kumar, R., & Singh, A. (2025). NVIDIA Blackwell and NVIDIA CUDA 12.9 introduce family-specific architecture features. *NVIDIA Technical Blog*. https://developer.nvidia.com/blog/nvidia-blackwell-and-nvidia-cuda-12-9-introduce-family-specific-architecture-features/
3. Bahig, H. M., Hazber, M. A. G., Al-Utaibi, K., & Nassr, D. I. (2022). Efficient sequential and parallel prime sieve algorithms. *Symmetry*, *14*(12), 2527. https://doi.org/10.3390/sym14122527

4. Frąckiewicz, M. (2025). High-performance computing highlights (June-July 2025): Exascale era, HPC-AI convergence, and global supercomputing advances. *TechStock*. https://ts2.tech/en/high-performance-computing-highlights-june-july-2025-exascale-era-hpc-ai-convergence-and-global-supercomputing-advances/
5. Månsson, J. (2021). *Comparative study of CPU and GPGPU implementations of the sieves of Eratosthenes, Sundaram and Atkin* [Master's thesis, Blekinge Institute of Technology]. DiVA. https://www.diva-portal.org/smash/get/diva2:1531686/FULLTEXT01.pdf
6. Asaduzzaman, A., Maiti, A., & Yip, C. (2014). Fast effective deterministic primality test using CUDA/GPGPU. *International Journal of Computer Applications, 98*(11), 37-41. https://doi.org/10.5120/17234-7625
7. NVIDIA. (2025). *CUDA C++ best practices guide 12.9*. NVIDIA Developer Documentation. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
8. Ghidarcea, M., & Popescu, D. (2024). Prime time tactics: Sieve tweaks and boosters. *Algorithms, 17*(7), 291. https://doi.org/10.3390/a17070291
9. Naik, H., Mishra, M., Routray, G., & Behera, M. (2020). Performance optimization by integrating memoization and MPI_Info object for sieve of prime numbers. *International Journal of Computer Applications, 177*(47), 34-38. https://doi.org/10.5120/ijca2020920114
10. Seizert, C., Garland, M., & Temam, O. (2018). A highly efficient multi-GPU implementation of the sieve of Eratosthenes. *Journal of Parallel and Distributed Computing, 113*, 90-100. https://doi.org/10.1016/j.jpdc.2017.10.016
11. Chykunov, P., & Chernetskyi, V. (2023). Organization of prime number search process using NVIDIA CUDA. In *Modern technologies in power engineering, electromechanics, control systems and mechanical engineering: Proceedings of the VI All-Ukrainian Scientific-Practical Internet Conference* (pp. 40-41). Kharkiv. https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi
12. Shafie Khorassani, K., Hashmi, J., Chu, C. H., Chen, C. C., Subramoni, H., & Panda, D. K. (2021). Designing a ROCm-aware MPI library for AMD GPUs: Early experiences. In *High Performance Computing* (Vol. 12728). Springer. https://doi.org/10.1007/978-3-030-78713-4_7
13. Shama, E., & Grant, R. (2025). NAV: A comparative analysis tool for Nsight Systems GPU traces. In *2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)* (pp. 536-543). IEEE. https://doi.org/10.1109/PDP66500.2025.00082
14. Massimo, J. (2020). *An analysis of primality testing and its use in cryptographic applications* [Doctoral dissertation, Royal Holloway, University of London]. https://pure.royalholloway.ac.uk/ws/portalfiles/portal/39023193/2020MassimoJPhD.pdf
15. Chykunov, P. O., Manakov, S. Yu., & Trofymenko, O. H.(2025).Efficiency of algorithms for systematic search of prime numbers. *Scientific Works of Donetsk National Technical University. Series: Problems of Modeling and Design Automation,2*(22),146-154.https://doi.org/10.31474/2074-7888-2025-2-22-146-154
16. Luo, W., Chen, X., Wang, Y., & Li, Z. (2025). Dissecting the NVIDIA Hopper architecture through microbenchmarking and multiple level analysis. *arXiv*. https://doi.org/10.48550/arXiv.2501.12084
17. Chykunov, P., & Kotov, A. (2023). Profiling multithreaded programs in the Visualizer Concurrency. In *Modern technologies in power engineering, electromechanics, control systems and mechanical engineering: Proceedings of the VI All-Ukrainian Scientific-Practical Internet Conference* (pp. 36-38). Kharkiv. https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi

Отримано редакцією журналу / Received: 13.02.26
Прорецензовано / Revised: 27.02.26
Схвалено до друку / Accepted: 25.06.26