DOI 10.15276/imms.v14.no4.350
УДК 004.42:004.272:519.612.2

Informatics and Mathematical Methods in Simulation
Vol.14 (2024), No. 4, pp. 350-357

КРОСПЛАТФОРМЕНА СИСТЕМА АНАЛІЗУ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ

О. О. Жульковський[1], Г. Я. Вохмянін[1], І. І. Жульковська[2],
Ю. В. Ульяновська[2], В. А. Рябоволенко[2]

[1]Дніпровський державний технічний університет
2, Дніпробудівська вул., м. Кам'янське, 51918, Україна
[2]Університет митної справи та фінансів
2/4, Володимира Вернадського вул., м. Дніпро, 49000, Україна
Email: olalzh@ukr.net

Представлено результати розроблення та дослідження спеціалізованої системи для автоматизованого порівняльного аналізу ефективності обчислювальних алгоритмів, зокрема з використанням технології SIMD. Основною метою дослідження є створення кросплатформеного програмного забезпечення для проведення обчислювальних експериментів, що дозволяє порівнювати продуктивність стандартних та оптимізованих реалізацій алгоритмів розв'язання систем лінійних алгебраїчних рівнянь. Дослідження зосереджено на підходах до розроблення та порівнянні продуктивності реалізації фундаментальних методів розв'язання, зокрема методів Гауса та спряжених градієнтів. Для досліджуваних методів розроблено по дві версії реалізації – стандартна та оптимізована з використанням SIMD-інструкцій. Програмна реалізація алгоритмів виконана продуктивними засобами Microsoft Visual Studio C++ із використанням стандартної спеціалізованої бібліотеки immintrin.h та набору команд процесора AVX. Розроблений програмний комплекс базується на сучасному стеку технологій, зокрема використовує фреймворк Nextron, який поєднує можливості Electron для створення кросплатформених застосунків та Next.js для побудови інтерактивного користувацького інтерфейсу. Архітектура системи забезпечує модульність, масштабованість та зручність використання завдяки застосуванню компонентного підходу React. У роботі представлено результати експериментального дослідження ефективності використання SIMD-технології для пришвидшення обчислювальних алгоритмів. Проведені експерименти демонструють суттєве підвищення продуктивності реалізації паралельних обчислювальних алгоритмів у діапазоні 2.67–5.81 разів у залежності від розмірності СЛАР та обраного обчислювального методу. Виявлено, що ефективність SIMD-оптимізації зростає пропорційно збільшенню обсягу вхідних даних, тоді як для малих розмірностей накладні витрати на векторизацію можуть лімітувати загальну продуктивність. Результати дослідження підтверджують потенціал у використанні SIMD-технології для оптимізації обчислювальних алгоритмів та демонструють практичну цінність розробленого програмного забезпечення для автоматизації процесу проведення та аналізу обчислювальних експериментів. Отримані результати можуть бути використані при розробці інших оптимізованих алгоритмів та програмних систем, зокрема для комп'ютерного моделювання, орієнтованих на високопродуктивні обчислення.

Ключові слова: кросплатформений застосунок, SIMD, паралелізм на рівні даних, пришвидшення обчислень, immintrin.h, Electron, Next.js, Nextron.

Вступ. Наукові дослідження, пов'язані із проведенням обчислювальних експериментів, потребують значних часових витрат на підготовку та аналіз експериментальних даних, а процес систематизації та візуалізації отриманих результатів часто виконується вручну з використанням непов'язаних між собою інструментів. Такі підходи мають низьку ефективність із-за збільшення часу проведення досліджень, високої ймовірності виникнення помилок під час обробки даних, залучення додаткових ресурсів тощо. У цьому контексті розроблення спеціалізованого застосунку для автоматизації процесу

проведення обчислювальних експериментів є актуальним завданням, вирішення якого дозволить створити проблемно-орієнтоване програмне забезпечення не тільки для проведення експериментів, а й автоматизованої обробки результатів та їх візуалізації у вигляді інформативних графіків та таблиць.

SIMD (Single Instruction, Multiple Data – одиночний потік команд, множинний потік даних) є одним із сучасних інструментів для пришвидшення виконання алгоритмів, які працюють з великими обсягами однорідних даних, шляхом одночасної обробки декількох елементів у межах однієї інструкції [1]. Порівняння продуктивності класичних алгоритмів і їх оптимізованих версій дозволяє визначити не лише швидкість виконання, але й вплив різних факторів, таких як розмір вхідних даних, конфігурація обладнання та оптимізація коду [2].

Окремі бібліотеки мови програмування JavaScript формують сучасний стек технологій для розроблення кросплатформеного застосунку з підтримкою вебтехнологій [3]. Їх поєднання дозволяє використовувати інструменти стилізації, маршрутизації та анімації всередині десктопних та мобільних застосунків.

Огляд літератури. Сучасним фреймворком для створення кросплатформених застосунків із використанням звичних вебтехнологій, зокрема HTML, CSS та JavaScript, є Electron [3]. З його використанням можна перетворювати вебзастосунки, розроблені за допомогою мови програмування JavaScript, на десктопні застосунки для операційних систем Windows, MacOS та Linux. Вихідний застосунок залишається з тим самим набором програмного коду для подальшого розроблення. В основі Electron лежить комбінація двох ключових компонентів: движок Chrome V8 для виконання JavaScript та фреймворк Node.js для доступу до системних ресурсів [4]. Архітектура Electron базується на двох основних процесах: головному процесі, який керує життєвим циклом застосунку та має доступ до системних API (Application Programming Interface – прикладний програмний інтерфейс), та процесі рендерингу, який відповідає за відображення користувацького інтерфейсу. Комунікація між процесами здійснюється через вбудований механізм IPC (Inter-Process Communication) [5].

Для створення користувацьких вебінтерфейсів часто використовується декларативна бібліотека React [6]. Вона надає компонентний підхід, де інтерфейс розбивається на незалежні компоненти. React використовує віртуальний DOM (Document Object Model– об’єктна модель документа) для оптимізації рендерингу та забезпечення високої продуктивності застосунків. Також він містить однонаправлений потік даних, за рахунок чого спрощуються відстеження змін стану застосунку [7]. Програмний код JavaScript поєднується з розміткою компонентів у JSX (JavaScript XML) синтаксисі [8].

Next.js розширює можливості React [9], надаючи готовий фреймворк для створення повноцінних вебзастосунків. Next.js забезпечує серверний рендеринг SSR (Server Side Rendering), статичну генерацію сторінок SSG (Static Site Generation), автоматичний роутинг на основі файлової системи, оптимізацію зображень, вбудовану підтримку CSS модулів та API маршрути [10]. Фреймворк також надає можливості для оптимізації продуктивності, включаючи автоматичне розділення коду, попереднє завантаження сторінок та оптимізацію зображень. Next.js спрощує розроблення складних вебзастосунків, надаючи готові рішення для типових завдань та проблем [11].

Описані бібліотеки разом формують фреймворк Nextron [12], який дозволяє використовувати всі переваги Next.js, включаючи серверний рендеринг та системи маршрутизації, в контексті десктопного застосунку Electron. Тобто застосунок Next.js виконується в процесі рендерингу Electron, в той час як основна логіка застосунку та взаємодія з операційною системою здійснюється через головний процес Electron.

Створення інтерактивних графіків та діаграм для вебзастосунків може відбуватися за допомогою бібліотеки Chart.js [7]. В її основі лежить елемент HTML5 Canvas, який забезпечує високу продуктивність при відображенні графіків. Бібліотека

використовує об'єктно-орієнтований підхід, де кожен тип графіка представлений окремим класом, що успадковується від базового класу Chart [13]. Архітектура побудована за модульним принципом, дозволяючи імпортувати лише необхідні компоненти, зменшуючи загальний розмір. Підтримує щонайменше вісім основних типів діаграм, кожен з яких може бути налаштований відповідними параметрами конфігурації.

Архітектурна реалізація SIMD базується на використанні спеціалізованих векторних регістрів та набору векторних інструкцій [14]. Ці регістри мають збільшену розрядність (128, 256 або 512 біт) та можуть зберігати множину елементів даних одночасно. Процесорні інструкції SIMD оперують цими регістрами як єдиним цілим, виконуючи паралельну обробку всіх елементів за один такт процесора [15]. У сучасних процесорних архітектурах існує декілька поколінь SIMD-розширень. Для архітектури x86 це послідовність технологій MMX, SSE (Streaming SIMD Extensions) різних версій, AVX (Advanced Vector Extensions) та найновіша AVX-512. Кожне нове покоління розширень збільшувало розмір векторних регістрів та додавало нові інструкції, розширюючи можливості паралельної обробки даних.

Заголовний файл immintrin.h у Visual Studio C++ є компонентом для роботи з SIMD-інструкціями на сучасних процесорах Intel та AMD. Даний файл надає програмний інтерфейс для доступу до внутрішніх функцій процесора, які забезпечують паралельну обробку даних на апаратному рівні. Інтерфейс immintrin.h забезпечує доступ до різних наборів SIMD-інструкцій, включаючи MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AVX2 та AVX-512 [16]. Кожен з цих наборів представляє еволюційний розвиток технології SIMD, розширюючи можливості паралельної обробки даних та підвищуючи продуктивність обчислень.

Мета роботи. Метою дослідження є розроблення спеціалізованого кросплатформеного застосунку для автоматизації процесу проведення обчислювальних експериментів із порівнянням продуктивності стандартних та оптимізованих реалізацій обчислювальних алгоритмів. В якості об'єкта для дослідження було обрано задачу розв'язання СЛАР різними методами – прогонки, класним методом Гауса та методом спряжених градієнтів – для встановлення та порівняння продуктивності кожного з них. Для реалізації оптимізованої версії перелічених алгоритмів застосовано технологію SIMD. Стандартні та оптимізовані алгоритми мають бути розроблені мовою програмування C++, оскільки вона є більш продуктивною, ніж JavaScript. Десктопний застосунок повинен надавати користувацький інтерфейс для проведення експериментів, запускаючи окремо розроблені версії алгоритмів.

Дослідження спрямоване на визначення потенціалу оптимізації обчислювальних процесів за рахунок використання паралелізму на рівні даних на сучасних процесорах. Досягнення поставленої мети передбачає вирішення комплексу взаємопов'язаних завдань:

- реалізувати фундаментальні алгоритми розв'язання СЛАР із використанням засобів мови програмування C++;
- реалізувати паралельні версії зазначених алгоритмів із використанням бібліотеки immintrin.h та інструкцій AVX для оптимізації векторних операцій;
- розробити кросплатформений застосунок на Nextron для проведення обчислювальних експериментів та представлення порівняльного аналізу у вигляді таблиць і графіків;
- із використанням розробленого застосунку провести обчислювальні експерименти та проаналізувати отримані результати.

Основна частина. Класичні та оптимізовані алгоритми розв'язання СЛАР реалізовані мовою C++ як окремі складові. Для проведення обчислень вони генерують однорідні дані випадковим чином на початку своєї роботи та вимірюють час розв'язання СЛАР

класичним і оптимізованим способами, виводячи отримані результати в консольний рядок.

Розроблене програмне забезпечення представляє собою кросплатформений застосунок (рис. 1) для порівняльного аналізу швидкодії послідовних та паралельних алгоритмів розв'язання СЛАР. Основним призначенням системи є візуалізація та оцінка ефективності паралельних обчислень із використанням векторних інструкцій AVX.

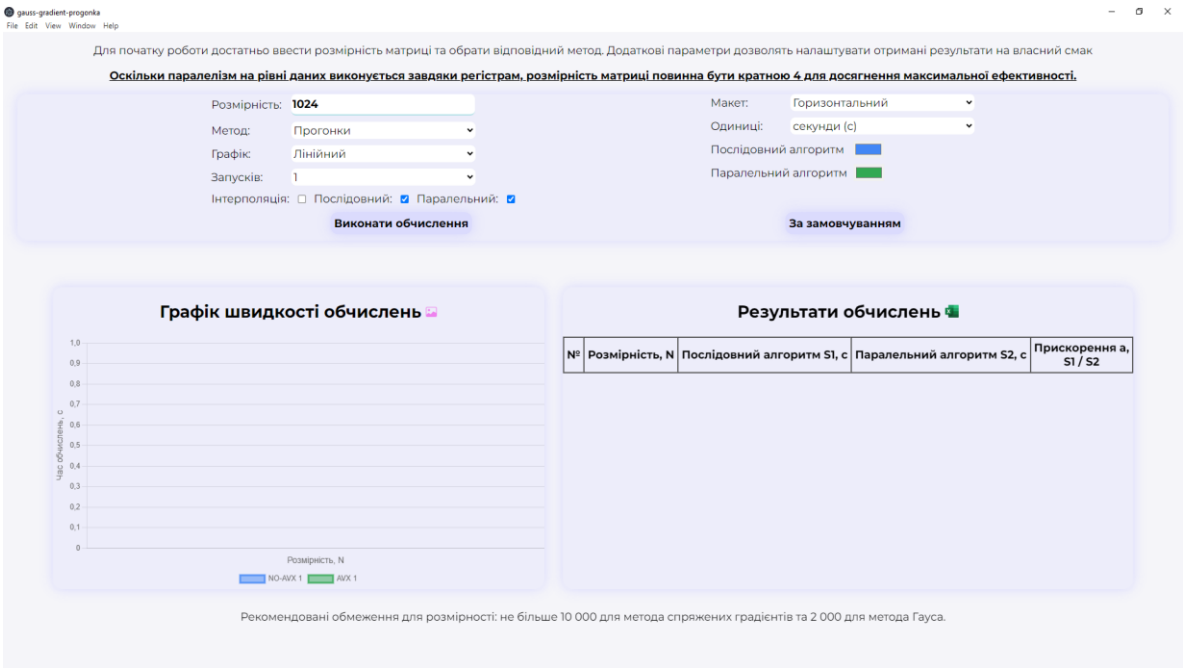

Рис. 1. Користувацький інтерфейс кросплатформеного застосунку

Архітектура програмного забезпечення побудована на основі компонентного підходу з використанням фреймворку Next.js. Центральним елементом системи є компонент Home, який керує всією логікою роботи застосунку та станом даних. Для зберігання та управління даними використовується локальний стан на основі React hooks, що дозволяє динамічно оновлювати інтерфейс при зміні параметрів обчислень.

Користувацький інтерфейс застосунку розділений на декілька функціональних блоків. У верхній частині розташована панель налаштувань, де користувач може задати основні параметри обчислень: розмірність матриці, метод розв'язання (прогонки, класичний Гауса або спряжених градієнтів), тип візуалізації результатів, кількість запусків для усереднення результатів та додаткові параметри відображення даних. Необхідною є вимога щодо кратності розмірності матриці числу 4, що обумовлено специфікою роботи векторних регістрів AVX.

Взаємодія користувача з системою описана у вигляді діаграми послідовності (рис. 2).

Процес обчислень запускається окремою функцією, яка виконує послідовний запуск алгоритмів, розроблених на C++, для різних розмірностей матриці. Система автоматично розбиває діапазон розмірностей на фіксовані кроки для побудови графіків залежності часу обчислень від розміру задачі. Результати кожного запуску зберігаються окремо для послідовного та паралельного алгоритмів з метою подальшого проведення порівняльного аналізу.

Візуалізація результатів реалізована за допомогою компонента Chart, який надає можливість відображення даних у різних форматах: лінійний графік, гістограма, радарна діаграма, бульбашкова діаграма, полярна діаграма тощо. Для кожного типу візуалізації передбачені додаткові налаштування, такі як інтерполяція даних та

кольорове оформлення графіків. Система також включає табличне представлення результатів, де для кожної розмірності задачі відображаються час виконання послідовного та паралельного алгоритмів, а також досягнуте їх пришвидшення. Передбачена можливість експорту результатів у форматі Excel та збереження графіків у вигляді PNG-зображень для подальшого аналізу. За необхідності можна динамічно змінювати одиниці вимірювання часу (секунди або мілісекунди), налаштовувати макет відображення результатів (горизонтальний або вертикальний), та керувати відображенням окремих компонентів візуалізації. Система також надає рекомендації щодо обмежень на розмірність задачі для різних методів розв'язання, допомагаючи користувачу обрати оптимальні параметри для конкретного випадку.

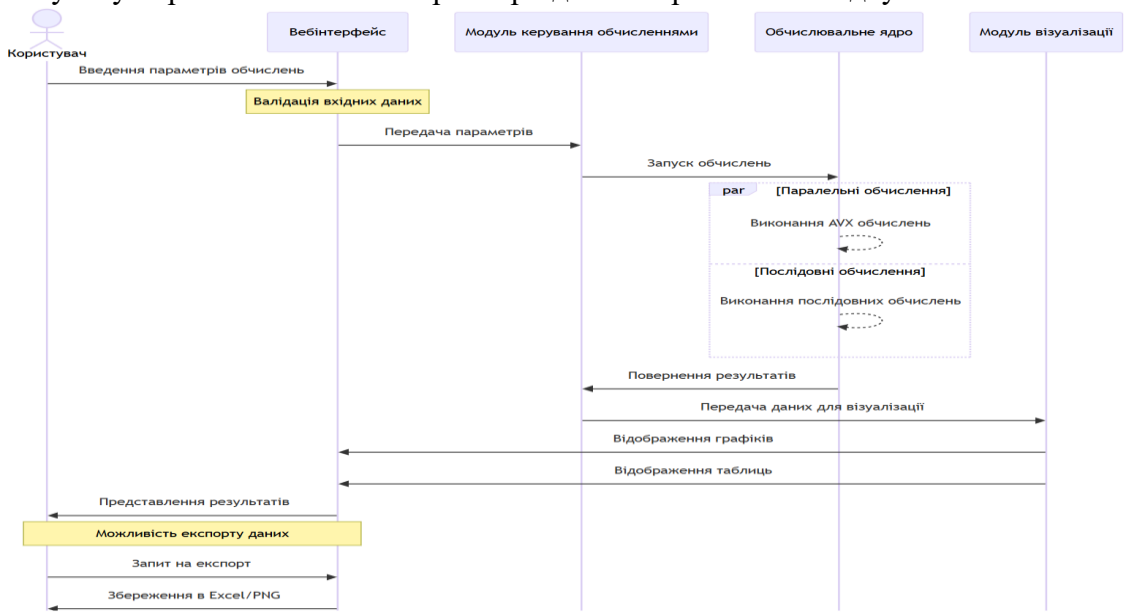

Рис. 2. Діаграма послідовності взаємодії користувача з системою

Обробка станів завантаження та помилок забезпечує стабільну роботу системи навіть при тривалих обчисленнях. Програмне забезпечення автоматично блокує елементи керування під час виконання обчислень та надає візуальний зворотний зв'язок про стан процесу. Архітектура програмної системи розділена на декілька ключових модулів. Модуль вебінтерфейсу відповідає за представлення користувацького інтерфейсу та взаємодію з користувачем. Він реалізований за допомогою React-компонентів та включає в себе форми введення параметрів, елементи керування та контейнери для відображення результатів. Модуль керування обчисленнями є центральним компонентом системи, який координує роботу інших модулів. Він обробляє користувацькі введення, виконує валідацію параметрів, керує процесом обчислень та розподіляє результати між іншими компонентами. У цьому модулі реалізована основна бізнес-логіка програми, включаючи функцію для запуску обчислень та обробку отриманих результатів. Модуль обчислювального ядра відповідає за безпосереднє виконання математичних розрахунків. Він розділений на два підмодулі: модуль послідовних алгоритмів та модуль паралельних обчислень з використанням AVX. Кожен з цих підмодулів виконує відповідні алгоритми розв'язання СЛАР. Модуль візуалізації відповідає за графічне представлення результатів обчислень. Підмодуль відображення графіків використовує компонент Chart для побудови різних типів діаграм, а підмодуль табличного представлення формує структуровані таблиці результатів. Модуль експорту даних забезпечує можливість збереження результатів у різних форматах. Він взаємодіє з компонентами візуалізації для експорту графіків у форматі PNG та таблиць у форматі Excel, використовуючи бібліотеки XLSX та html2canvas для реалізації функціональності

експорту. Модуль управління станом забезпечує централізоване зберігання та управління даними програми. Він реалізований з використанням React hooks (хук useState) та відповідає за синхронізацію стану між різними компонентами системи. Цей модуль зберігає такі дані як результати обчислень, налаштування візуалізації, параметри користувацького інтерфейсу тощо. Модуль конфігурації містить налаштування системи, включаючи параметри за замовчуванням, обмеження на розмірності матриць для різних методів, колірні схеми для візуалізації та інші константи програми. Він також відповідає за валідацію користувацьких налаштувань. Взаємодія між модулями організована за принципом слабкого зв'язування, що забезпечує можливість незалежного розвитку та модифікації кожного з них. Система використовує події та зворотні виклики для комунікації між модулями, підтримуючи асинхронну природу застосунку.

Результат та обговорення. В результаті розроблення було реалізовано кросплатформений застосунок для автоматизованого проведення обчислювальних експериментів. Дослідження ефективності паралельних обчислень проводилося на основі порівняльного аналізу часу виконання послідовних та паралельних реалізацій трьох різних методів розв'язання СЛАР. Паралельна реалізація базується на використанні технології SIMD за допомогою набору інструкцій AVX. Для проведення експериментів використовувалася розроблена програмна система. Результати вимірювань часу виконання алгоритмів представлені у вигляді графіків залежності часу обчислень від розмірності задачі. Для забезпечення точності вимірювань система підтримує представлення результатів як у секундах, так і в мілісекундах, дозволяючи детально аналізувати продуктивність на різних масштабах даних. Головним показником ефективності паралельної реалізації є коефіцієнт прискорення. Експериментальні дані демонструють стабільне пришвидшення для всіх досліджуваних методів, при цьому спостерігається нелінійна залежність коефіцієнта прискорення від розмірності задачі. На рис. 3 наведено результати одного з обчислювальних експериментів. Для тестування був використаний метод спряжених градієнтів при максимальній розмірності матриці – 2048. Всі три запуски надали приблизно однаковий результат. Досягнуте пришвидшення обчислень знаходиться в межах 2.67–5.81 в залежності від розмірності матриці. Для меншої розмірності матриці фіксується менше пришвидшення через накладні витрати на векторизацію.

Висновки. Розроблений програмний комплекс дозволяє автоматизувати проведення експериментів з порівняльного аналізу обчислювальних методів із використанням технології SIMD. Результати засвідчують, що використання SIMD-інструкцій сприяє підвищенню продуктивності обчислень, зокрема для задач, які включають обробку великих об'ємів даних. Кросплатформений застосунок може використовуватись для проведення експериментів та візуалізації отриманих результатів будь-яких окремо розроблених обчислювальних алгоритмів. Експерименти, проведені в межах дослідження, виявили суттєве пришвидшення виконання алгоритмів розв'язання СЛАР, таких як методи прогонки, класичний метод Гауса та метод спряжених градієнтів. Залежно від розмірності задачі та методу, коефіцієнт прискорення варіювався в межах від 2.67 до 5.81 разів. Зокрема, для малих розмірностей матриць ефект пришвидшення обмежений через накладні витрати на векторизацію, однак зі збільшенням об'ємів даних ефективність SIMD-інструкцій стає більш помітною. Отримані результати пришвидшення обчислень кореспондуються з результатами досліджень авторів [17].

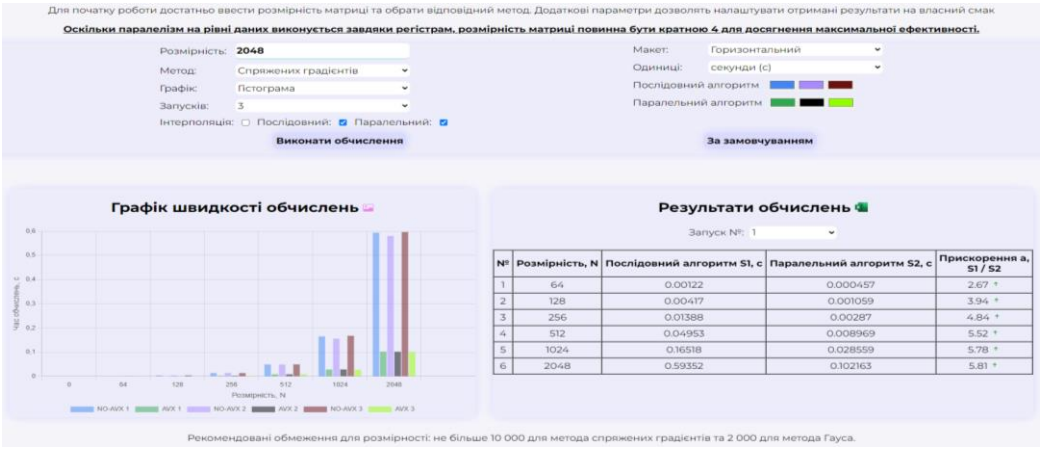

Рис. 3. Демонстрація результатів обчислювального експерименту

Описана архітектура програмного забезпечення базується на модульному підході та забезпечує гнучкість і масштабованість. Використання компонентного підходу з React та Next.js дозволяє підтримувати асинхронність та інтерактивність для сучасних систем, орієнтованих на обробку великих об’ємів даних. Модулі, відповідальні за візуалізацію, дозволяють представляти результати у вигляді графіків та таблиць, а також експортувати їх для подальшого аналізу, що розширює можливості використання системи в дослідницьких цілях. Дослідження також підтверджує доцільність і ефективність застосування SIMD-інструкцій для розв’язання обчислювальних задач, демонструючи переваги паралельної на рівні даних обробки у порівнянні зі стандартними підходами. Отримані результати можуть бути використані для розроблення нових комп’ютерних моделей та систем, орієнтованих на високопродуктивні обчислення.

Список літератури

1. Zhang W., Yan Z., Lin Y., Zhao C., Peng L. A High Throughput B+tree for SIMD Architectures. *IEEE Trans. Parallel Distrib. Syst.* 2020. V. 31. No. 3. P. 707–720. URL: https://doi.org/10.1109/tpds.2019.2942918
2. Naganawa Y., Kamei H., Kanetaka Y., Nogami H., Maeda Y., Fukushima N. SIMD-Constrained Lookup Table for Accelerating Variable-Weighted Convolution on x86/64 CPUs. *IEEE Access*. 2024. V. 12. P. 15800-15819. URL: https://doi.org/10.1109/access.2024.3354720
3. Kredpattanakul K., Limpiyakorn Y. Transforming JavaScript-Based Web Application to Cross-Platform Desktop with Electron. *Information Science and Applications 2018*. Singapore. 2018. P. 571–579. URL: https://doi.org/10.1007/978-981-13-1056-0_56
4. Srinivasa Rao M., Sandhya P., Sambana B., Mishra P. Application for Mood Detection of Students Using TensorFlow and Electron JS. *Springer Proceedings in Mathematics & Statistics*. Cham, 2023. P. 235–243. URL: https://doi.org/10.1007/978-3-031-15175-0_19
5. Electron Documentation. URL: https://www.electronjs.org/docs/latest/
6. Shevtsiv N.A., Striuk A.M. Cross platform development vs native development. *CEUR Workshop Proceedings*. 2021. V. 2832. P. 75–83. URL: https://doi.org/10.31812/123456789/4428
7. Bernatska N., Typilo I., Dzhumelia E. React Library: A Case Study on the Effective Instruments of the Design of an Environmental Monitoring System. *2023 IEEE 18th Int. Conf. Comput. Sci. Inf. Technol. (CSIT)*. 2023. P. 1–4. URL: https://doi.org/10.1109/csit61576.2023.10324124
8. React Documentation. URL: https://legacy.reactjs.org/docs/getting-started.html
9. Patel V. Analyzing the Impact of Next.JS on Site Performance and SEO. *International Journal of Computer Applications Technology and Research*. 2023. V. 12. P. 24–27. URL: https://doi.org/10.7753/IJCATR1210.1004

10. Jartarghar H.A., Rao S. G., Ashok K.A.R, Sharvani G.S., Dalali S. React Apps with Server-Side Rendering: Next.js. *J. Telecommunication, Electron. Comput. Eng. (JTEC)*. 2022. V. 14, no. 4. P. 25–29. URL: https://doi.org/10.54554/jtec.2022.14.04.005
11. Next.js Documentation. URL: https://nextjs.org/docs
12. Nextron Github. URL: https://github.com/saltyshiomix/nextron
13. Chart.js Documentation. URL: https://www.chartjs.org/docs/latest
14. Mustafa D., Alkhasawneh R., Obeidat F., Shatnawi A.S. MIMD Programs Execution Support on SIMD Machines: A Holistic Survey. *IEEE Access*. 2024. V. 12, P. 34354–34377. URL: https://doi.org/10.1109/access.2024.3372990
15. Intel Architecture Instruction Set Extensions Programming Reference. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/319433-024-697869.pdf
16. Intel Intrinsics Guide. URL: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
17. Zhulkovskyi O.O., Zhulkovska I.I., Vokhmianin H.Ya., Firsov O.D., Riabovolenko V.A. Research of progressive tools of parallel computations with the use of SIMD architecture. *Inform. math. methods simul*. 2023. V. 13, no. 3-4. P. 228–235 URL: https://doi.org/10.15276/imms.v13.no3-4.228

CROSS-PLATFORM SYSTEM FOR ANALYZING THE EFFICIENCY OF PARALLEL COMPUTING ALGORITHMS

O. O. Zhulkovskyi[1], H. Ya. Vokhmianin[1], I. I. Zhulkovska[2],
Yu. V. Ulianovska[2], V. A. Riabovolenko[2]

[1]Dniprovsky State Technical University
2, Dniprobudivska Ave., Kamianske city, 51918, Ukraine
[2]University of Customs and Finance
2/4, Volodymyr Vernadskyi Ave., Dnipro, 49000, Ukraine
Email: olalzh@ukr.net

The paper presents the results of the development and research of a specialized system for automated comparative analysis of the efficiency of computational algorithms, in particular, using SIMD technology. The main goal of the study is to create cross-platform software for conducting computational experiments that allows comparing the performance of standard and optimized implementations of algorithms for solving systems of linear algebraic equations. The study focuses on approaches to the development and comparison of the performance of fundamental methods for solving equations, in particular, Gauss and conjugate gradient methods. For the studied methods, two versions of the implementation were developed – standard and optimized using SIMD instructions. The software implementation of the algorithms was performed by the productive tools of Microsoft Visual Studio C++ using the standard specialized library immintrin.h and the AVX processor instruction set. The developed software system is based on a modern technology stack, in particular, it uses the Nextron framework, which combines the capabilities of Electron for creating cross-platform applications and Next.js for building an interactive user interface. The architecture of the system provides modularity, scalability, and usability through the use of the React component approach. The paper presents the results of an experimental study of the effectiveness of using SIMD technology to speed up computational algorithms. The experiments demonstrate a significant increase in the performance of parallel computing algorithms in the range of 2.67–5.81 times, depending on the dimensionality of the SLAE and the chosen computational method. It was found that the efficiency of SIMD optimization increases proportionally to the increase in the amount of input data, while for small dimensions the vectorization overhead can limit the overall performance. The results of the study confirm the potential of using SIMD technology to optimize computational algorithms and demonstrate the practical value of the developed software for automating the process of conducting and analyzing computational experiments. The results obtained can be used in the development of other optimized algorithms and software systems, in particular for computer modeling, focused on high-performance computing.
Keywords: cross-platform application, SIMD, data-level parallelism, computing acceleration, immintrin.h, Electron, Next.js, Nextron.