https://doi.org/10.31891/2219-9365-2025-82-61
УДК 004.056:004.852:004.75

КОЗЕЛЬСЬКИЙ Олександр
Хмельницький національний університет
https://orcid.org/0009-0002-7157-6499
e-mail: oleksandr.kozelskiy@khmnu.edu.ua

МЕТОД ТЕНЗОРНОЇ ДЕКОМПОЗИЦІЇ ДЛЯ АДАПТИВНОГО РОЗПОДІЛУ РЕСУРСІВ У СИСТЕМАХ РЕАЛЬНОГО ЧАСУ

У роботі представлено метод динамічного розподілу ресурсів на основі тензорної декомпозиції, спрямований на значне підвищення ефективності управління обчислювальними ресурсами та планування в системах реального часу. Запропонований підхід забезпечує всебічний аналіз багатовимірних характеристик процесів — включно з обсягом споживаних ресурсів, пріоритетними вагами та часовими параметрами виконання — що формує глибше розуміння внутрішньої динаміки та взаємозв'язків між компонентами системи.

Завдяки можливості операцій з тензорами, метод додає системам гнучкість і масштабованість: від вбудованих систем із жорсткими обмеженнями на пам'ять і енергоспоживання до розподілених кіберфізичних платформ із великою кількістю вузлів і змінними робочими навантаженнями. Інтеграція алгоритмів розкладання тензорів відбувається прозоро для стандартних каналів передачі даних і повністю сумісна з типовими схемами управління пам'яттю, підтримуваними сучасними процесорними архітектурами. Результати експериментальних досліджень підтверджують, що застосування цього підходу дозволяє зменшити середній час простою задач, скоротити число хибних спрацьовувань механізмів планування та покращити баланс навантажень між обчислювальними ядрами. За рахунок масштабованості та адаптивності метод може стати основою для подальших досліджень у галузі інтелектуального управління ресурсами.

Ключові слова: операційна система, RTOS, тензорна декомпозиція, планувальник задач, управління ресурсами

KOZELSKYI Oleksand
Khmelnytskyi National University

A TENSOR DECOMPOSITION–BASED METHOD FOR ADAPTIVE RESOURCE ALLOCATION IN REAL-TIME SYSTEMS

The paper presents a tensor-decomposition–based method for dynamic resource allocation, aimed at substantially improving computational resource management and scheduling in real-time systems. The proposed approach enables a comprehensive analysis of the multidimensional characteristics of processes—including resource consumption profiles, priority weights, and execution-time parameters—thereby yielding a deeper understanding of the internal dynamics and interdependencies among system components. Unlike traditional models that rely on linear approximations or simplified heuristics, tensor decomposition allows for capturing complex correlations across multiple dimensions simultaneously, which results in more precise and context-aware allocation strategies.

By leveraging tensor operations, the method endows platforms with both flexibility and scalability: it can be applied equally to resource-constrained embedded systems, where optimization must occur under strict hardware limitations, and to large-scale cyber-physical environments featuring numerous nodes, heterogeneous architectures, and variable workloads. Integration of the tensor-decomposition algorithms occurs transparently over standard data-transfer channels and remains fully compatible with conventional memory-management schemes supported by modern processor architectures. This ensures that the method does not require disruptive infrastructure changes, making it suitable for practical deployment in existing technological ecosystems.

Results from experimental evaluations confirm that this approach reduces average task idle time, decreases the incidence of spurious scheduling events, and improves load balancing across computational cores. Furthermore, the method demonstrates resilience under fluctuating workloads and dynamically changing operational conditions, maintaining efficiency even in highly volatile environments. Thanks to its scalability, adaptability, and potential for integration with intelligent decision-making modules, the proposed solution also provides a solid foundation for future research in intelligent resource management, including predictive scheduling, anomaly detection, and adaptive optimization of distributed computing infrastructures.

Keywords: operating system, RTOS, tensor decomposition, task scheduler, resource management

Стаття надійшла до редакції / Received 16.04.2025
Прийнята до друку / Accepted 11.05.2025

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Розробка нової архітектури RTOS потребує критичного переосмислення існуючих методів кластеризації, які не забезпечують ефективності, гнучкості та масштабованості за динамічних навантажень і жорстких часових обмежень. Необхідно створити адаптивні алгоритми з оптимізованою обчислювальною складністю — через паралельну обробку або спрощені моделі, що знижують витрати ресурсів. Важливим є забезпечення високої якості даних для навчання та швидке переналаштування параметрів планувальника завдань в умовах змінної доступності ресурсів і пріоритетів. Інтеграція штучного інтелекту, розподілених обчислень і хмарних сервісів підвищить передбачуваність продуктивності, надійність і масштабованість

системи, адаптуючи архітектуру до вимог сучасних кіберфізичних середовищ, де критично важливі оперативний відгук, стабільність і раціональне використання ресурсів.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Кластеризація в операційних системах реального часу (ОСРЧ) — головний механізм групування задач і ресурсів за подібними характеристиками, що забезпечує детермінованість і раціональне завантаження процесорів. На практиці найчастіше застосовують чотири підходи: Time Division Multiplexing (TDM), методи машинного навчання (ML), генетичні алгоритми (ГА) та граф-орієнтовані техніки. Кожен із них має виразні переваги й водночас суттєві обмеження.

TDM розподіляє фіксовані часові інтервали між задачами, гарантуючи регулярний і передбачуваний доступ до апаратних ресурсів [1]. Простота впровадження робить цей підхід привабливим для вбудованих платформ. Проте якщо задача не вичерпує відведений їй слот, невикористані цикли губляться, що знижує загальну ефективність. Система погано пристосовується до змінних навантажень і ризикує втратити масштабованість, коли кількість процесів росте [2]. У середовищах, де навантаження коливається, виникає потреба в динамічніших механізмах планування.

Методи ML — від кластеризації й навчання з підкріпленням до різних типів нейромереж — здатні автоматично аналізувати великі масиви телеметрії, прогнозувати пікові режими та коригувати розподіл ресурсів у реальному часі [3-5]. За рахунок динамічної адаптації вони підвищують стійкість систем до несподіваних змін. Основні бар'єри такі: необхідність великих обсягів якісних даних для навчання, значні обчислювальні витрати та складність інтеграції, особливо коли потрібна пояснюваність отриманих рішень [6]. Для ресурсно обмежених платформ це може бути критичним.

Генетичні алгоритми, що імітують еволюційний пошук через кросинговер і мутації, демонструють здатність знаходити глобально оптимальні або близькі до оптимуму розбиття навантаження за неповної інформації [8]. Вони стійкі до шумів і можуть обійти локальні мінімуми. Водночас ГА часто мають високу обчислювальну складність, повільно збігаються й потребують ретельного налаштування параметрів (розмір популяції, імовірності кросинговеру та мутації). У реальних-часових середовищах затримки, спричинені довгою збіжністю, є критичними, а масштабування на великі кластери потребує додаткових ресурсів.

Графові методи моделюють задачі й ресурси як вершини, а їхні залежності — як ребра [7]. Алгоритми спектральної кластеризації, мінімального розрізу чи максимального потоку детально враховують взаємодії, забезпечуючи точне балансування. Вони добре масштабу-ються, однак побудова й актуалізація великих графів вимагає значних обчислень. Динамічні зміни топології можуть потребувати повторного перерахунку матриць суміжності, що надто витратне під жорсткими дедлайнами. Крім того, графові моделі залежать від точної інформації про міжпроцесні залежності; у складних системах ці залежності часто змінюються, що ускладнює підтримку коректної моделі.

Порівняльний аналіз показує: жоден із наведених підходів не забезпечує одночасно максимальну ефективність, гнучкість і масштабованість. TDM гарантує детермінізм, але нееластичний; ML забезпечує адаптивність, однак затратний у плані даних і ресурсів; ГА шукають глобальний оптимум, проте повільні; графові алгоритми масштабні, та їх важко швидко перебудовувати. Тому сучасні дослідження зосереджені на комбінуванні сильних сторін різних методик [9, 10-12], оптимізації обчислювальної складності через паралелізм і розподілену обробку, а також на розробці пояснюваних моделей, здатних стабільно працювати у суворих часових межах.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Як було зазначено вище, у сучасних ОСРЧ ефективне управління ресурсами та балансування навантаження є критично важливими завданнями для забезпечення надійної та продуктивної роботи систем. З розвитком кіберфізичних систем (КФС) та Інтернету речей (IoT) з'являється необхідність в аналізі та оптимізації великих обсягів даних у режимі реального часу. У цьому контексті використання тензорної декомпозиції та методів машинного навчання[13] представляє новий підхід до вирішення проблем класифікації задач і балансування навантаження в ОСРЧ[14].

Початковим етапом у використанні підходу є збір даних, що містять інформацію про завдання, ресурси, часові параметри та інші важливі фактори. Це можуть бути показники тривалості виконання завдань, використання ресурсів, пріоритети, взаємозв'язки між завданнями та інші ключові характеристики. Відбувається дана операція у агрегуючому модулі.

Кожне завдання пов'язується з відповідними параметрами, які відображають його вимоги та поведінку в системі.

Нехай $X = \{x^1, x^2, \dots, x^n\}$ — множина задач, а $R = \{r^1, r^2, \dots, r^m\}$ — множина ресурсів. Кожна задача $x_i \in X$ описується набором характеристик $F^i = \{f^1, f^2, \dots, f^k\}$, де f^j представляє один з параметрів, таких як час виконання, споживання ресурсів, пріоритети та взаємозв'язки.

Тому, основним завданням є збір даних, що визначають набір ознак F_i для кожної задачі x_i, тобто кожна задача пов'язується з набором характеристик:

$$F_1 = \{f_1(x_i), f_2(x_i), \dots, f_k(x_i)\}, \forall x_i \in X \quad (1)$$

Перший крок у розробці алгоритму полягає у створенні тензора X, який містить багатовимірну інформацію про задачі, ресурси та час, тощо.

Нехай X - це тензор розмірності I×J×K, який містить дані про різні аспекти задач і ресурсів у ОСРЧ. Кожен елемент тензора X_{ijk} представляє значення певного параметра для i-ї задачі, j-го ресурсу і k-го часу.

$$X = [X_{ijk}], \text{де } i = 1, \dots, I, j = 1, \dots, J, k = 1, \dots, K, \quad (2)$$

$$X \in R^{I \times J \times K}, \quad (3)$$

де I - кількість задач, J - кількість ресурсів, K - кількість часових інтервалів.

Перед виконанням тензорної декомпозиції здійснимо нормування даних так:

$$X'_{ijk} = \frac{x_{ijk} - \min(X)}{\max(X) - \min(X)} \quad (4)$$

Тензорна декомпозиція. Виконується CP (CANDECOMP/PARAFAC) декомпозиція тензора X для виявлення основних компонентів[17]. Вона розкладає тензор X на суму зовнішніх добутків векторів. CP (CANDECOMP/PARAFAC) декомпозиція є одним із основних методів для аналізу багатовимірних даних, представлених у вигляді тензора. Вона дозволяє розкласти тензор на компактну суму ранк-одиничних компонент, що спрощує його аналіз та обробку. Основна ідея CP-декомпозиції полягає в представленні складного багатовимірного тензора через лінійну комбінацію векторів, що відповідають кожному виміру. Припустимо, у нас є тензор X порядку N, який представляє дані з N вимірами. Для кожного виміру n n існує певна кількість факторів $a_r^{(n)}$, де r=1,2,…,R — це кількість ранк-одиничних компонент. Тензор X можна записати як:

$$X \approx \sum_{r=1}^{R} a_r^{(1)} \circ a_r^{(2)} \circ \dots \circ a_r^{(n)}, \quad (5)$$

де ∘ — оператор зовнішнього добутку, $a_r \in R_n^I$ — вектор факторів для n-го виміру, R — ранг тензора (мінімальна кількість ранк-одиничних компонент для точного або наближеного представлення X).

У випадку багатовимірного тензора X, кожен елемент тензора можна виразити через вектори факторів усіх вимірів:

$$x_{i_1, i_1 \dots, i_N} \approx \sum_{r=1}^{R} a_{i_1, r}^{(1)} \circ a_{i_2, r}^{(n)} \circ \dots \circ a_{i_n, r}^{(n)}, \quad (6)$$

де $x_{i_1, i_1 \dots, i_N}$ — елемент тензора для індексів i_1, i_2, …, i_N, а $a_{i_n, r}^{(n)}$ — елемент вектора $a_r^{(n)}$, що відповідає n-му виміру, ∘ — оператор зовнішнього добутку. Цей підхід узагальнює концепцію тензорного розкладу для довільної кількості вимірів N.

Класифікація задач. На основі векторів факторів a_r, b_r,…, n_r, отриманих після декомпозиції, ми можемо класифікувати задачі за їх характеристиками. Нехай z_i - це вектор характеристик для i-ї задачі, який містить елементи a_r.

Модель машинного навчання. Отримані компоненти використовуються для навчання моделі машинного навчання, яка буде використовуватись для класифікації задач та прогнозування потреб ресурсів. Модель машинного навчання $f(z_i)$ навчається на основі векторів факторів для прогнозування класу задачі або її пріоритету. Нехай y_i - це клас задачі, тоді:

$$y_i = f(z_i), \quad (7)$$

де $z_i \in R^R$ — факторний вектор задачі i, а y_i - прогнозований клас або пріоритет задачі.

Для динамічного балансування навантаження застосовуємо модель g, яка за факторним вектором z_i і поточним часом t оцінює запит на j -й ресурс:

$$r_j(t) = g(z_i, t), \quad (8)$$

де g: $R^R \times R \to R$ — функція прогнозування. R^R — R-вимірний вектор факторів z_i, R — поточний час t) . $r_j(t)$ вимірюється в одиницях ресурсу.

Оптимізація розподілу задач. Щоб забезпечити рівномірне навантаження, мінімізуємо суму абсолютних відхилень від середніх запитів на кожному ресурсі:

$$\min_{\{r_j(t)\}} \sum_{j=1}^{J} | \sum_{i=1}^{I} r_j(t) - \bar{r}_j, \quad (9)$$

де $\bar{r}_j = \frac{1}{I} \sum_{i=1}^{I} r_j(t)$ — середнє навантаження на j -й ресурс за поточний момент t. Для розв'язання цієї задачі можна застосувати лінійне програмування, субградієнтний спуск або інші методи оптимізації залежно від властивостей моделі g [16, 17].

Таким чином збираються дані про всі процеси в ОСРЧ, включаючи час виконання, споживання ресурсів, пріоритети, взаємозалежності і т.д. При потребі кількість ознак можна збільшувати для точнішого аналізу даних тим самим збільшуючи вимір тензора. Створюється тензор, що містить багатовимірні дані про всі процеси та ресурси в системі. Виконується тензорна декомпозиція для виявлення основних компонентів, що впливають на продуктивність системи. Вона допомагає зрозуміти, які фактори впливають на продуктивність і як краще розподілити ресурси для оптимального виконання задач.

Підготовка прототипу. Для реалізації прототипу за основу візьмемо операційну систему FreeRTOS з модифікованими файлом tasks.c (основний файл планувальника FreeRTOS, де визначаються функції для управління задачами.У ньому змодифікуємо функцію vTaskSwitchContext(), щоб викликати кастомний метод для розподілу задач) і main.c, де додамо код для створення задач збору даних та виклику кастомного планувальника, який буде в новому файлі custom_scheduler.c (у цьому файлі буде визначення функцій збору даних, класифікації задач та оптимізації розподілу. У цьому файлі буде реалізована інтеграція з TensorFlow Lite для виконання моделей машинного навчання).

Методологія експерименту передбачає послідовне виконання кількох взаємопов'язаних етапів, модель якої описана в наступних джерелах [15]. Насамперед здійснюється збір експериментальних даних у середовищі FreeRTOS. Під час роботи системи або в процесі її моделювання реєструються ключові параметри, що характеризують поведінку кожної задачі: тип завдання (обробка даних, I/O, фонова активність тощо), завантаження центрального процесора, споживання оперативної пам'яті, час виконання та пріоритет у черзі планувальника. Для цього застосовуються циклічні виклики та апаратні таймери, що дозволяє імітувати умови реального часу й фіксувати телеметрію з необхідною точністю. За допомогою вбудованого HTTP-клієнта накопичені дані у вигляді структурованих пакетів передаються з мікроконтролера на хмарний сервіс, реалізований засобами C# (.NET 9), де вони автоматично конвертуються у стандартизований формат для подальшої обробки.

На основі отриманих вибірок формується тривимірний тензор: перший вимір відповідає ідентифікаторам задач, другий – вектору їх характеристик (CPU, пам'ять, пріоритет, час), третій – часовій мітці виконання. Використовуючи бібліотеку Math.NET Numerics, тензор піддається декомпозиції з подальшим сингулярним розкладом (SVD), що забезпечує зниження розмірності та виділення латентних факторів, релевантних для машинного навчання. Підготовлений набір факторів конвертується у формат TensorFlow Lite і передається на віддалений кластер для навчання моделі, оптимізованої під обмежені ресурси вбудованих систем.

Після завершення навчання сформована модель повертається на сервер .NET, де проходить валідацію, а далі завантажується у мікроконтролер. У компоненті custom_scheduler.c модель виконує інференцію, прогнозуючи оптимальний розподіл процесорного часу та пам'яті між активними задачами; результати негайно впливають на ухвалення рішень планувальником. Цикл збору нових даних, їхньої обробки, повторного навчання та оновлення моделі запускається кожні дві години, що забезпечує поступову адаптацію планувальника до змінних умов навантаження й підвищує ефективність використання системних ресурсів у довгостроковій перспективі.

У контексті наведеного циклу збору телеметрії та повторного навчання CP-ALS було обрано як базовий алгоритм тензорної декомпозиції, оскільки обсяг даних (≈ 2,9 млн елементів) дає змогу досягати збіжності за помірну кількість ітерацій. Усі розрахунки виконувалися у середовищі Math.NET Numerics 6.0 (C#) з рангом $R = 20$. Відносна похибка реконструкції після 180–200 ітерацій не перевищувала 3×10^{-4} а повний розклад тривав у середньому 12 с на орендованому виділеному сервері з процесором Intel Xeon E-2386G. Отримані двадцятивимірні факторні вектори зберігалися у форматі float16 та конвертувалися у TensorFlow Lite; на кластері з GPU Nvidia T4 за ними навчали компактну MLP-модель $g(z_i, t)$ з двома прихованими шарами (64 → 32 нейрони, ReLU). Після 50 епох середня абсолютна похибка прогнозу ресурсних потреб становила 3,2 %. Скомпільована модель (151 КБ) завантажувалася на мікроконтролер STM32H7, де час однієї інференції не перевищував 0,31 мс, що дозволяло використовувати прогноз у кожному виклику vTaskSwitchContext().

Щоб підтримувати актуальність планувальника, цикл оновлення запускався кожні дві години: до тензора додавалися нові 7200 записів, CP-ALS перераховувався з попередніми матрицями як початковим наближенням (тривалість ≈ 15 с на тому самому Xeon-сервісі), а MLP донавчалася протягом 30 епох. Гаряча заміна файлу .tflite у прошивці забезпечувала безперервну роботу системи й поступову адаптацію до зміни профілю навантаження без зупинки виконання задач.

Оскільки модель $g(z_i, t)$ реалізована як нейронна мережа, для розв'язання задачі мінімізації (9) застосовували метод субградієнтного спуску, який дає змогу ефективно знаходити наближене оптимальне рішення в умовах нелінійності та недиференційованості функції втрат. Такий підхід забезпечував достатню точність розподілу ресурсів при помірному навантаженні на обчислювальні ресурси та дозволяв впровадити оптимізаційний цикл у реальному часі без необхідності зведення задачі до лінійної форми. Вибір цього методу був обумовлений як архітектурою моделі, так і технічними особливостями програмно-апаратної платформи.

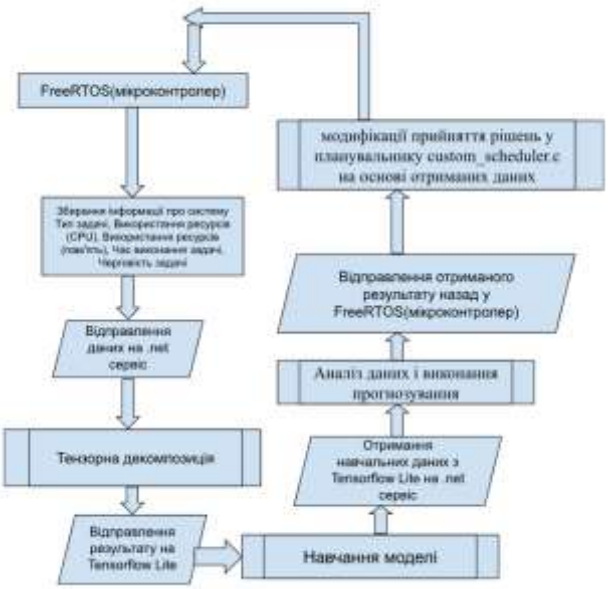
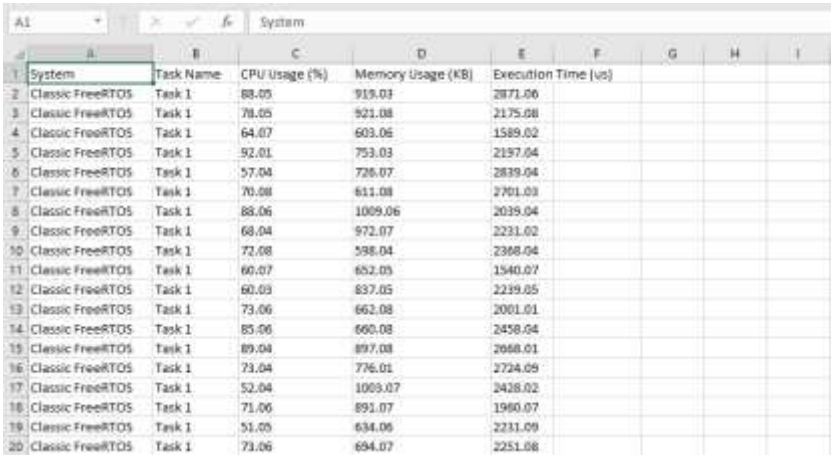


Рис. 1: Схема роботи прототипу покращення роботи ОСРЧ на основі класифікації задач у планувальнику за допомогою тензорної декомпозиції і машинного навчання

Проведення експериментів і порівняння ефективності. FreeRTOS використовує принцип управління часом через свій планувальник завдань (scheduler)**,** який дуже схожий на Часове розділення (Time Division Multiplexing, TDM). Ми можемо порівняти результати роботи FreeRTOS за замовчуванням і результати роботи FreeRTOS із модифікованим планувальником задач на основі тензорної декомпозиції і машинного навчання.

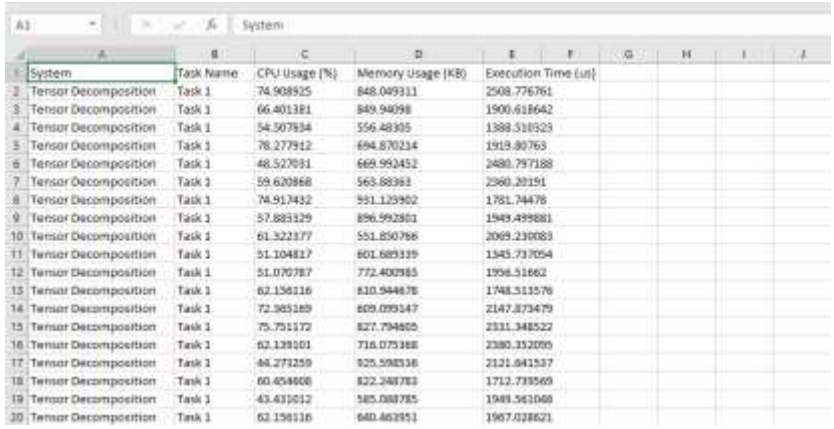
Для проведення порівняльного аналізу ефективності стандартної FreeRTOS та FreeRTOS із модифікованою архітектурою на основі тензорної декомпозиції було обрано кілька ключових метрик: тип задачі (обробка даних, введення/виведення, обчислювальні операції або фонові завдання), рівень використання процесорних ресурсів (CPU) під час виконання задачі, час її виконання, а також черговість або пріоритет задачі в системі.

На першому етапі експерименту було проведено запуск програм з використанням стандартного планувальника задач FreeRTOS; під час виконання завдань збиралися дані про час обробки, споживання ресурсів, пропускну здатність та затримки, що фіксувалися через системи логування. Після цього аналогічні вимірювання були здійснені для FreeRTOS із модифікованою архітектурою на основі тензорної декомпозиції. Для забезпечення статистичної надійності дослідження було сформовано набір із трьох типів задач, по кожній з яких було проведено 100 вимірювань. Отримані результати було збережено у форматі csv у файлах freertos_classic_results.csv та tensor_decomposition_results.csv для подальшого аналізу.



System	Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (us)
Classic FreeRTOS	Task 1	88.05	919.03	2871.06
Classic FreeRTOS	Task 1	78.05	921.08	2175.08
Classic FreeRTOS	Task 1	64.07	603.06	1589.02
Classic FreeRTOS	Task 1	92.01	753.03	2197.04
Classic FreeRTOS	Task 1	57.04	726.07	2839.04
Classic FreeRTOS	Task 1	70.08	611.08	2701.03
Classic FreeRTOS	Task 1	88.06	1009.06	2039.04
Classic FreeRTOS	Task 1	68.04	972.07	2231.02
Classic FreeRTOS	Task 1	72.08	598.04	2368.04
Classic FreeRTOS	Task 1	60.07	652.05	1540.07
Classic FreeRTOS	Task 1	60.03	837.05	2239.05
Classic FreeRTOS	Task 1	73.06	662.08	2001.01
Classic FreeRTOS	Task 1	85.06	660.08	2458.04
Classic FreeRTOS	Task 1	89.04	897.08	2668.01
Classic FreeRTOS	Task 1	73.04	776.01	2724.09
Classic FreeRTOS	Task 1	52.04	1003.07	2428.02
Classic FreeRTOS	Task 1	71.06	891.07	1960.07
Classic FreeRTOS	Task 1	51.05	634.06	2231.09
Classic FreeRTOS	Task 1	73.06	694.07	2251.08

Рис. 2: скріншот таблиці фрагменту результатів збору даних класичною FreeRTOS



System	Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (us)
Tensor Decomposition	Task 1	74.908925	848.049311	2508.776761
Tensor Decomposition	Task 1	66.401381	849.94098	1900.618642
Tensor Decomposition	Task 1	54.507834	556.48305	1388.510323
Tensor Decomposition	Task 1	78.277912	694.870234	1919.80763
Tensor Decomposition	Task 1	48.527031	669.992452	2480.797188
Tensor Decomposition	Task 1	59.620868	563.88363	2360.20191
Tensor Decomposition	Task 1	74.917432	951.125902	1781.74478
Tensor Decomposition	Task 1	57.683129	896.992801	1949.499881
Tensor Decomposition	Task 1	61.322177	551.850766	2069.230083
Tensor Decomposition	Task 1	51.104817	801.685319	1345.737054
Tensor Decomposition	Task 1	51.070787	772.400985	1956.51662
Tensor Decomposition	Task 1	62.156116	810.944678	1748.511576
Tensor Decomposition	Task 1	72.365189	809.099147	2147.873479
Tensor Decomposition	Task 1	75.751172	827.794665	2331.348522
Tensor Decomposition	Task 1	62.139101	716.075388	2380.352095
Tensor Decomposition	Task 1	46.271259	925.598536	2121.641537
Tensor Decomposition	Task 1	60.454608	822.248783	1712.739569
Tensor Decomposition	Task 1	43.431012	585.088785	1949.561068
Tensor Decomposition	Task 1	62.156116	640.463951	1967.028621

Рис. 3: скріншот таблиці фрагменту результатів збору даних з модифікованої FreeRTOS на основі тензорної декомпозиції

Після обчислення середніх значень було отримано такі результати: використання CPU у класичній FreeRTOS становить 74,03%, тоді як у системі з тензорною декомпозицією — 62,98%, що демонструє зменшення навантаження на процесор на 14,92%. Споживання пам'яті знизилося з 759,10 KB до 700,47 KB, тобто на 7,72%. Час виконання задач скоротився з 2224,29 мкс до 1943,62 мкс, що означає поліпшення на 12,62%. Ці дані підтверджують, що застосування тензорної декомпозиції забезпечує суттєве підвищення ефективності використання ресурсів порівняно з класичною FreeRTOS. Водночас варто зазначити, що очікуваний ефект від впровадження кастомного планувальника на основі TensorFlow Lite залежить від кількох факторів, зокрема від складності задач, характеру навантаження та якості реалізації самої машинної моделі.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Запропонований метод тензорної декомпозиції для адаптивного розподілу ресурсів, інтегрований у ядро FreeRTOS, продемонстрував вагоме підвищення ефективності керування ресурсами в режимі реального часу. Порівняльні експерименти засвідчили, що комплексний аналіз багатовимірних залежностей між задачами та апаратними ресурсами забезпечує зменшення завантаження CPU на 14,92 %, скорочення споживання оперативної пам'яті на 7,72 % та прискорення виконання задач на 12,62 % відносно класичної конфігурації. Тензорне представлення дозволяє динамічно класифікувати процеси за пріоритетами й ресурсними вимогами, оптимізуючи розподіл обчислювальних потужностей без втручання в логіку прикладових програм. Отримані результати особливо цінні для вбудованих і кіберфізичних систем, де суворі обмеження на час відгуку та пам'ять критичні для безпеки й надійності.

Методологія є прозорою щодо транспорту користувацьких даних, сумісна з усіма схемами керування пам'яттю, підтримуваними сучасними процесорами, і легко масштабується від мінімальних однокристальних платформ до комплексних багатоядерних обчислювальних середовищ. Закладений потенціал подальшого розвитку охоплює інтеграцію прогнозних моделей машинного навчання, здатних адаптивно корегувати параметри розподілу ресурсів залежно від поточної телеметрії, що відкриває шлях до формування нових архітектур високопродуктивних та гнучких систем реального часу. Ця розробка не тільки розширює

можливості систем FreeRTOS, але й забезпечує основу для створення нових архітектур управління ресурсами, які можуть бути критично важливими для майбутніх кіберфізичних систем.

Література

1. Mosqueda-Arvizu, C.-A., Romero-González, J.-A., Córdova-Esparza, D.-M., Terven, J., Chaparro-Sánchez, R., & Rodríguez-Reséndiz, J. (2024). Logical Execution Time and Time-Division Multiple Access in Multicore Embedded Systems: A Case Study. Algorithms, 17(7), 294. https://doi.org/10.3390/a17070294.

2. Schwäricke, G., Kloda, T., Gracioli, G., Bertogna, M., & Caccamo, M. (2020). Fixed-priority memory-centric scheduler for COTS-based multiprocessors. Leibniz International Proceedings in Informatics (LIPIcs), 165, 1–24. https://doi.org/10.4230/LIPIcs.ECRTS.2020.1.

3. Sharma, S., Chmaj, G., & Selvaraj, H. (2023). Applying machine learning to minimize the impact of sensor failures in RTOS-based Internet of Things systems. In H. Selvaraj, G. Chmaj, & D. Zydek (Eds.), Advances in systems engineering (Vol. 761, pp. 135–146). Springer. https://doi.org/10.1007/978-3-031-40579-2_14.

4. Cho, C., Seong, Y., & Won, Y. (2021). Mandatory access control method for Windows Embedded OS security. Electronics, 10(24), 2478. https://doi.org/10.3390/electronics10202478.

5. Mohammad, A., Das, R., Islam, M. A., & Mahjabeen, F. (2024). Real-time operating systems (RTOS) for embedded systems. Asian Journal of Mechatronics and Electrical Engineering, 2(2), 95–104. https://doi.org/10.55927/ajmee.v2i2.7761.

6. Delgadillo, O., Blieninger, B., Kuhn, J., & Baumgarten, U. (2021). An architecture to enable machine-learning-based task migration for multi-core real-time systems. In Proceedings of the 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (pp. 405–412). IEEE. https://doi.org/10.1109/MCSoC51149.2021.00066.

7. Park, S., Kwon, M.-Y., Kim, H.-K., & Kim, H. (2020). Execution model to reduce the interference of shared memory in ARINC 653 compliant multicore RTOS. Applied Sciences, 10(7), 2464. https://doi.org/10.3390/app10072464.

8. Kabilesh, S. K., et al. (2020). Resemblance of real-time scheduling algorithms for real-time embedded systems. Journal of Optoelectronics and Communication, 2(3).

9. Gahtan, B., Cohen, R., Bronstein, A. M., & Kedar, G. (2023, February 18). Using deep reinforcement learning for mmWave real-time scheduling (v2). arXiv. https://doi.org/10.48550/arXiv.2210.01423.

10. Hassan, H. E., Ibrahiem, K. H., & Madian, A. H. (2025). Optimizing multiprocessor performance in real-time systems using an innovative genetic algorithm approach. Scientific Reports, 15, Article 3842. https://doi.org/10.1038/s41598-024-80910-4

11. Rashid, S. M., Aliyu, I., Jeong, I.-K., Um, T.-W., & Kim, J. (2025, January 19). Graph neural network for in-network placement of real-time metaverse tasks in next-generation network (v2). arXiv. https://doi.org/10.48550/arXiv.2403.01780

12. Fischer, D., Hüsener, H. M., Grumbach, F., Vollenkemper, L., Müller, A., & Reusch, P. (2024, August 30). Demystifying reinforcement learning in production scheduling via explainable AI (v2). arXiv. https://doi.org/10.48550/arXiv.2408.09841

13. Amin, M. R., Hasan, M., Arnab, S. P., & DeGiorgio, M. (2023). Tensor decomposition-based feature extraction and classification to detect natural selection from genomic data. Molecular Biology and Evolution, 40(10), msad216. https://doi.org/10.1093/molbev/msad216

14. Gahtan, B., Cohen, R., Bronstein, A. M., & Kedar, G. (2022, October 4). Using deep reinforcement learning for mmWave real-time scheduling. arXiv. https://arxiv.org/abs/2210.01423

15. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing // CEUR Workshop Proceedings. – 2025. – Т. 3899. – С. 234–243. – Режим доступу: https://ceur-ws.org/Vol-3899/paper21.pdf

16. Boyd, S., & Vandenberghe, L. (2004). Convex optimization [PDF]. Cambridge University Press. – Режим доступу: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

17. Bertsekas, D. P. (1999). Nonlinear programming (2nd ed.). Athena Scientific. ISBN 1-886529-00-0

References

1. Mosqueda-Arvizu, C.-A., Romero-González, J.-A., Córdova-Esparza, D.-M., Terven, J., Chaparro-Sánchez, R., & Rodríguez-Reséndiz, J. (2024). Logical Execution Time and Time-Division Multiple Access in Multicore Embedded Systems: A Case Study. Algorithms, 17(7), 294. https://doi.org/10.3390/a17070294.

2. Schwäricke, G., Kloda, T., Gracioli, G., Bertogna, M., & Caccamo, M. (2020). Fixed-priority memory-centric scheduler for COTS-based multiprocessors. Leibniz International Proceedings in Informatics (LIPIcs), 165, 1–24. https://doi.org/10.4230/LIPIcs.ECRTS.2020.1.

3. Sharma, S., Chmaj, G., & Selvaraj, H. (2023). Applying machine learning to minimize the impact of sensor failures in RTOS-based Internet of Things systems. In H. Selvaraj, G. Chmaj, & D. Zydek (Eds.), Advances in systems engineering (Vol. 761, pp. 135–146). Springer. https://doi.org/10.1007/978-3-031-40579-2_14.

4. Cho, C., Seong, Y., & Won, Y. (2021). Mandatory access control method for Windows Embedded OS security. Electronics, 10(24), 2478. https://doi.org/10.3390/electronics10202478.

5. Mohammad, A., Das, R., Islam, M. A., & Mahjabeen, F. (2024). Real-time operating systems (RTOS) for embedded systems. Asian Journal of Mechatronics and Electrical Engineering, 2(2), 95–104. https://doi.org/10.55927/ajmee.v2i2.7761.

6. Delgadillo, O., Blieninger, B., Kuhn, J., & Baumgarten, U. (2021). An architecture to enable machine-learning-based task migration for multi-core real-time systems. In Proceedings of the 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (pp. 405–412). IEEE. https://doi.org/10.1109/MCSoC51149.2021.00066.

7. Park, S., Kwon, M.-Y., Kim, H.-K., & Kim, H. (2020). Execution model to reduce the interference of shared memory in ARINC 653 compliant multicore RTOS. Applied Sciences, 10(7), 2464. https://doi.org/10.3390/app10072464.

8. Kabilesh, S. K., et al. (2020). Resemblance of real-time scheduling algorithms for real-time embedded systems. Journal of Optoelectronics and Communication, 2(3).

9. Gahtan, B., Cohen, R., Bronstein, A. M., & Kedar, G. (2023, February 18). Using deep reinforcement learning for mmWave real-time scheduling (v2). arXiv. https://doi.org/10.48550/arXiv.2210.01423.

10. Hassan, H. E., Ibrahiem, K. H., & Madian, A. H. (2025). Optimizing multiprocessor performance in real-time systems using an innovative genetic algorithm approach. Scientific Reports, 15, Article 3842. https://doi.org/10.1038/s41598-024-80910-4

11. Rashid, S. M., Aliyu, I., Jeong, I.-K., Um, T.-W., & Kim, J. (2025, January 19). Graph neural network for in-network placement of real-time metaverse tasks in next-generation network (v2). arXiv. https://doi.org/10.48550/arXiv.2403.01780

12. Fischer, D., Hüsener, H. M., Grumbach, F., Vollenkemper, L., Müller, A., & Reusch, P. (2024, August 30). Demystifying reinforcement learning in production scheduling via explainable AI (v2). arXiv. https://doi.org/10.48550/arXiv.2408.09841

13. Amin, M. R., Hasan, M., Arnab, S. P., & DeGiorgio, M. (2023). Tensor decomposition-based feature extraction and classification to detect natural selection from genomic data. Molecular Biology and Evolution, 40(10), msad216. https://doi.org/10.1093/molbev/msad216

14. Gahtan, B., Cohen, R., Bronstein, A. M., & Kedar, G. (2022, October 4). Using deep reinforcement learning for mmWave real-time scheduling. arXiv. https://arxiv.org/abs/2210.01423.

15. Kozelskiy, O., Kashtalian, A., Stetsyuk, V., Martiniuk, D., & Sachenko, A. (2025). A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. CEUR Workshop Proceedings, 3899, 234–243. Retrieved from https://ceur-ws.org/Vol-3899/paper21.pdf

16. Boyd, S., & Vandenberghe, L. (2004). Convex optimization [PDF]. Cambridge University Press. Retrieved from https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

17. Bertsekas, D. P. (1999). Nonlinear programming (2nd ed.). Athena Scientific. ISBN 1-886529-00-0