УДК 629.783-022.513.2.05:004.4-024.35 **doi: 10.32620/aktt.2024.4.11**

О. В. ЛЮБІМОВ, І. Б. ТУРКІН, О. Б. ЛЕЩЕНКО, В. С. ВАЛКОВИЙ

Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна

ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА ПРОДУКТИВНОСТІ МЕТОДУ КОНТЕЙНЕРИЗАЦІЇ В БОРТОВОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ НАНОСУПУТНИКА CUBESAT

Об'єкт дослідження *– обчислювальна ефективність різних системних архітектур бортового програмного забезпечення бортових обчислювачів наносупутників CubeSat.* ***Предметом дослідження*** *є накладні витрати на обчислення при використанні методу контейнеризації в побудові бортового програмного забезпечення наносупутників CubeSat.* ***Мета роботи*** *– експериментально обґрунтувати висновки щодо можливості та доцільності використання методу контейнеризації в бортовому програмному забезпеченні наносупутників CubeSat.* ***Завдання****: обґрунтувати необхідність пошуку нових архітектурних рішень в бортовому програмному забезпеченні наносупутника CubeSat, виконати порівняльний аналіз переваг та недоліків використання монолітної (класичної) та мікросервісної архітектури в бортовому програмному забезпеченні наносупутника CubeSat, обґрунтувати вибір системного програмного середовища виконання контейнерів, визначити типову структуру програмного забезпечення CubeSat та стратегію адаптації контейнерного середовища WASM3 в операційну систему FreeRTOS, розробити план та провести експериментальні дослідження, за результатами яких сформулювати висновки щодо можливості та доцільності використання контейнерної архітектури в бортовому програмному забезпеченні наносупутника CubeSat.* ***Висновки****. У дослідженні продемонстрована актуальність розробки програмного забезпечення наносупутників на основі мікросервісів та контейнерів. Отримані експериментальні результати дозволяють порівняти продуктивність бортового обчислювача з виконання різних алгоритмів, реалізованих за допомогою мови програмування С (виконання на залізі, Bare-Metal) та архітектури, побудованої на підході мікросервісів, розгалужених між контейнерами середовища WASM3, що виконуються під керуванням ОСРЧ FreeRTOS та розроблених за мовами С та С++. Основним висновком роботи є знайдена можливість використання апаратної платформи «Falco SBC/CDHM» у якості доступної та потужної обчислювальної платформи для наносупутників CubeSat.*

Ключові слова: *CubeSat; наносупутник; COTS; бортовий обчислювач; контейнеризація; архітектура мікросервісів; програмне забезпечення; WASM3; Боривітер/Falco; обчислювальна ефективність; накладні витрати.*

Вступ

Двадцять п'ять років тому професори Роберт Твіггс (Стенфордський університет) і Джорді Пуч-Суарі (Cal Poly – Каліфорнійський політехнічний університет) здійснили важливу зміну в парадигмі космічних досліджень, розробивши концепт 10-сантиметрового наносупутника - CubeSat [1]. Це зробило космічні дослідження та науку доступнішими для студентів. За менше ніж десять років CubeSats перетворилися з освітніх інструментів у стандартну платформу для демонстрації технологій і наукових досліджень. Використання комерційних готових компонентів COTS (Commercial Off-The-Shelf) [2] і мініатюризація технологій призвели до створення багатьох місій із значною науковою цінністю [3].

Кожен блок CubeSat мінімального типорозміру 1U має розміри 10×10×10 $см^3$ і масу до 1,33 кг. Такі блоки можуть збиратись в більш великі збірки наносупутників. Вже реалізовано конфігурації CubeSat з 1, 2, 3, 6, 12 та навіть 27U для задоволення попиту на більші можливості наносупутників. Станом на 31 травня 2024 року найпоширенішими є конфігурації 3U (41%), 6U (15%) і 1U (12%) (рис. 1).

Основними перевагами наносупутників класу CubeSat, у порівнянні з традиційними супутниками, є низька вартість, модульна побудова з повторним використанням окремих підсистем і значно менший час розробки. Однак вони мають проблеми з продуктивністю та надійністю. Існуючі процеси проєктування, розробки, забезпечення якості та експлуатації потребують удосконалень. Наносупутники, розроблені в університетах [4], переслідують освітні, наукові та технологічні цілі, залучаючи студентів до командної роботи, випробовування нових компонентів і демонстрації інноваційних місій. CubeSat стали невід'ємним компонентом сучасної STEM-освіти [5, 6].

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

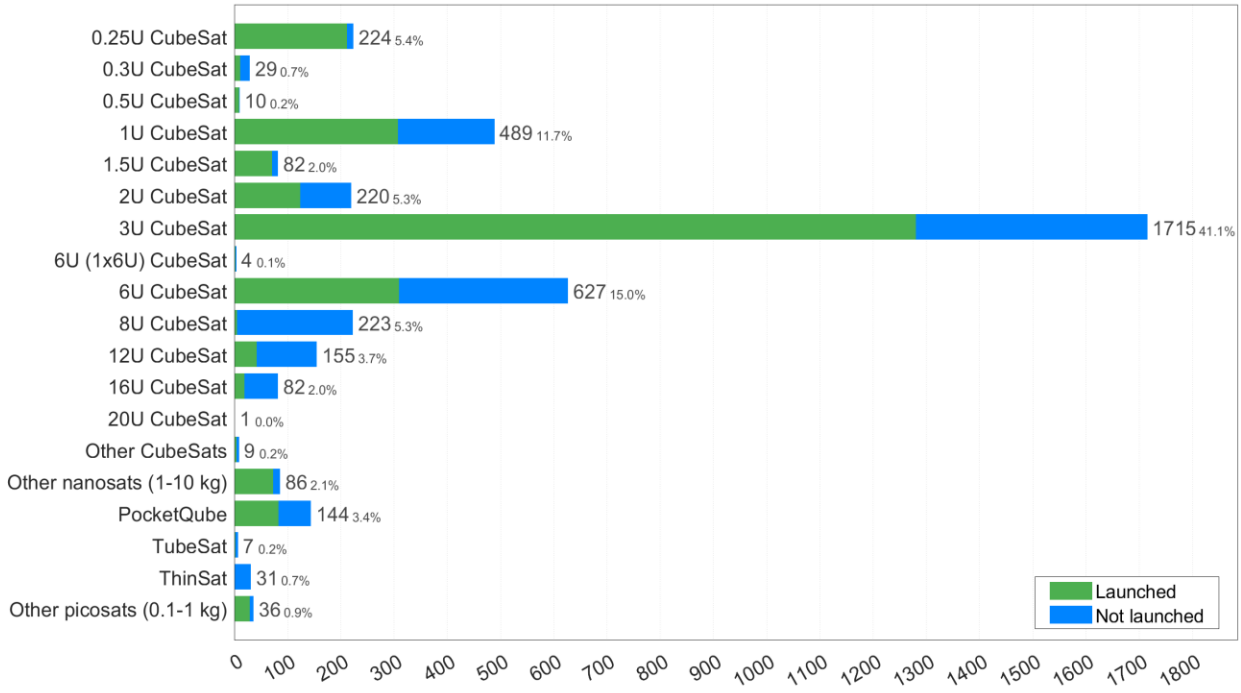


Рис. 1. Кількість запущених наносупутників в залежності від їх розміру (станом на травень 2024) [7]

Наносупутники CubeSat мають незалежне керування живленням, температурний контроль, керування даними, контроль орієнтації, систему зв'язку та навігацію. Наносупутники живляться від сонячних елементів на освітленій частині орбіти, та від хімічної батареї при нестачі сонячної енергії. Стандартизовані специфікації побудови CubeSat мінімізують ризик для ракети-носія та інших корисних навантажень, розміщених на ній. Модульна структура дозволяє легко комбінувати різні компоненти для різних місій, що спрощує їх проектування, інтеграцію та повторне використання.

Ключова перевага стандартизованих специфікацій для CubeSats полягає в тому, що вони мінімізують ризик для решти ракети-носія та інших її корисних навантажень [8]. Відділення CubeSat від носія здійснюються через диспенсери, наприклад P-POD (Poly – Picosatellite Orbital Deployer), розроблений CalPoly. CubeSats катапультуються з P-POD після досягнення орбіти і починають свої місії.

Близько половини місій CSLI (CubeSat Launch Initiative) [9] проводять наукові дослідження, включаючи космічну погоду, науки про Землю, біологію, вивчення навколоземних об'єктів, кліматичні зміни, снігове покриття, орбітальне сміття, планетологію, астрономію та геліофізику. Низька вартість розробки CubeSat дозволяє виконувати ризиковані проєкти, неможливі під час масштабних місій NASA.

Зазвичай до складу типових наносупутників CubeSat входять наступні електронні системи:

- бортовий обчислювач (OBC(D) – On Board Computer (and Data));
- система орієнтації (ADCS – Attitude Determination and Control System);
- система енергоживлення (EPS – Energy Power System);
- система зв'язку (Comm, або COM);
- корисне навантаження (Payload) , яке власне і створює комерційну користь від наносупутника.
- Система керування двигуном (Propulsion Control System), яка зазвичай використовується в достатньо великих (6U+) наносупутниках CubeSat.

До CubeSat також входять неелектронні компоненти, такі як сонячні панелі, батареї, антени, двигун або декілька двигунів та механічна конструкція, навколо якої збираються всі модулі (рис. 2).

У деяких досить складних збірках CubeSat OBC(D) та ADCS об'єднуються в єдиний модуль обробки команд (CDHM – Command and Data Handling Module).

З точки зору розподілу обчислювальних завдань на борту виділимо два підходи: централізований, коли OBC/CDHM виконує все, а всі інші компоненти мінімально активні (рис. 3), та розподілений, коли OBC/CDHM лише координує роботу інших компонентів системи, а кожен компонент виконує власну програму (рис. 4).

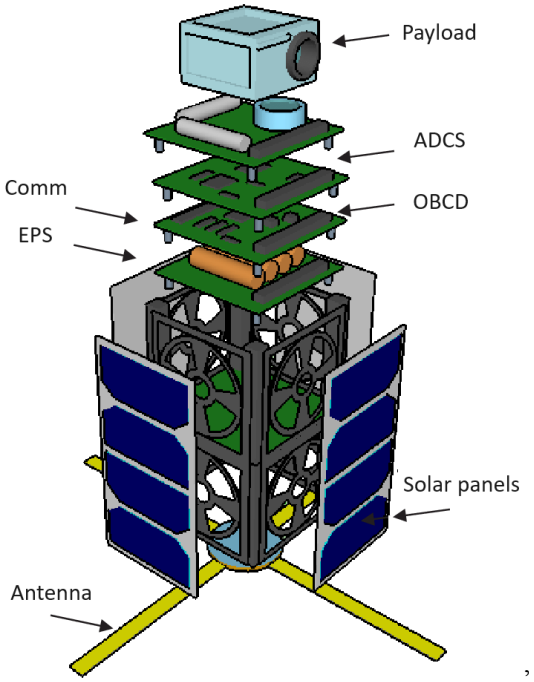


Рис. 2. Типова структура та компоненти наносупутника CubeSat розміру 3U

1. Огляд сучасного стану та вимог до бортового ПЗ

Надалі основну увагу буде приділено бортовому програмному забезпеченню (ПЗ), оскільки бортове ПЗ зазвичай є складнішим для розробки, підтримки та експлуатації, ніж апаратна частина з наступних причин.

1. Складність розробки та тестування: ПЗ повинно взаємодіяти з численними іншими програмами та системами та враховувати безліч можливих сценаріїв використання, що ускладнює його розробку і тестування.

2. Постійні оновлення та підтримка необхідні для виправлення помилок, покращення функціональності і забезпечення безпеки.

3. ПЗ має бути сумісним з різними апаратними компонентами, що може бути складним через варіації у технічних характеристиках та можливостях обладнання.

4. Бортове ПЗ працює на енергетично обмеженій платформі в режимі реального часу, що накладає додаткові вимоги до енергетичної ефективності програмної реалізації алгоритмів, швидкості та надійності обробки даних.

Стаття [10] виділяє наступні ключові нефункціональні вимоги до ПЗ наносупутників:

– ***надійність:*** захист від збоїв та відмов, спричинених зовнішніми факторами, такими як випромінювання, екстремальні температури тощо, здатність виявляти, ізолювати та відновлюватися після збоїв, реалізація безпечних режимів для запобігання критичним відмовам; використання надійних комунікаційних протоколів з перевіркою помилок, можливість програмного резервування критичних даних та завантажувачів;

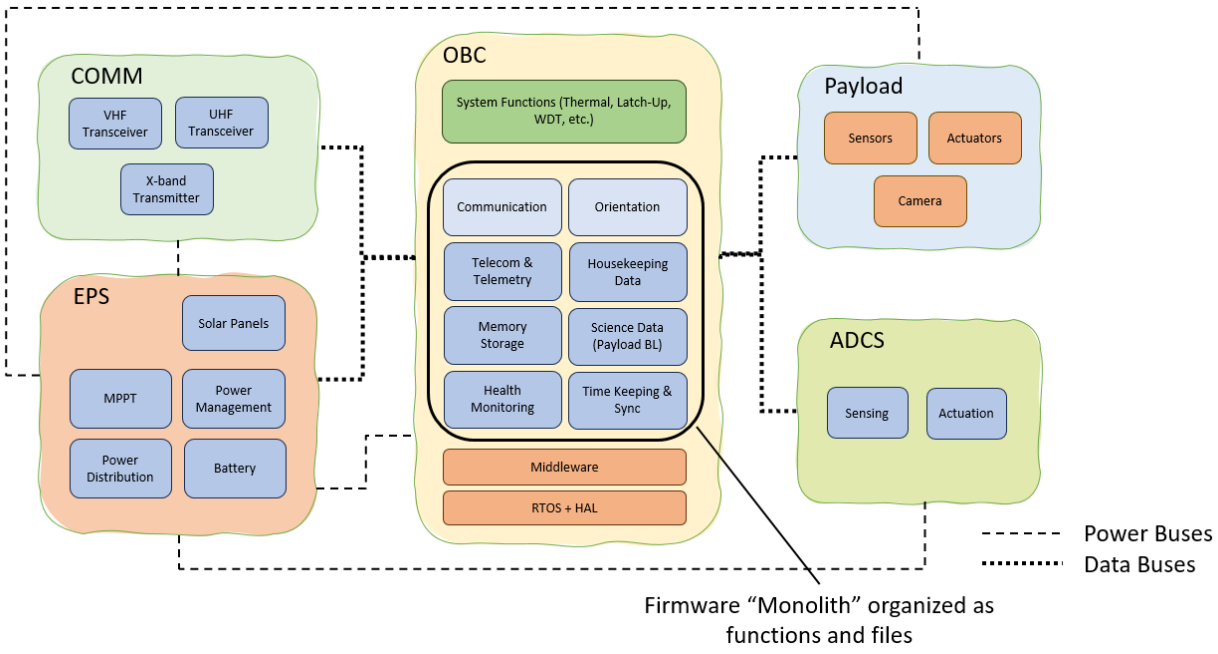


Рис. 3. Централізована організація обчислень в CubeSat

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

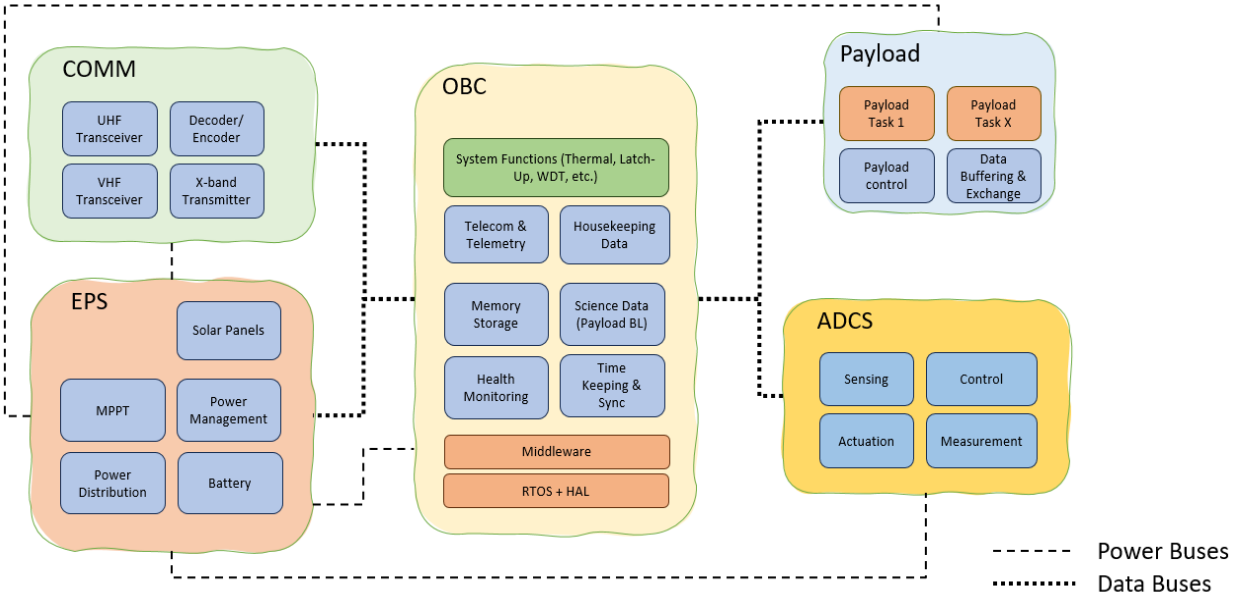


Рис. 4. Децентралізована (розподілена) організація обчислень в CubeSat

– ***модульність:*** чітка інкапсуляція та інтерфейси модулів, пов'язаних з корисним навантаженням та підсистемами, легка замінність та розширюваність цих модулів для різних місій, підтримка масштабованості для додавання/видалення модулів, забезпечення здатності оновлювати окремі модулі на орбіті;

– ***автономність:*** можливість автономного планування завдань та діяльності супутника, генерація розкладів виконання на борту на основі цілей місії та обмежень системи, точне та передбачуване виконання згенерованих планів з виявленням порушень.

Архітектурні рішення бортового ПЗ наносупутників достатньо різноманітні.

ПЗ канадського проекту CubeSat, яке працює під операційною системою FreeRTOS на бортовому обчислювачі (OBC) на базі STM32, має багаторівневу архітектуру та розділене на п’ять рівнів: від нижнього рівня, призначеного для периферійних пристроїв, до верхнього рівня, присвяченого застосункам місії. Протокол CubeSat (CSP – CubeSat Space Protocol) використано для забезпечення зв’язку між підсистемами та для зв’язку з наземним сегментом [11].

Для забезпечення надійності розробки програмного забезпечення, модульності, багаторазового використання та масштабованості, в супутнику Masat-1 [12] було використано багаторівневий сервіс-орієнтований архітектурний підхід, який базується на кінцевому автоматі для виконання функцій у детермінований спосіб. Модель клієнт-сервер була обрана для міжпроцесного зв’язку та доступу до ресурсів, використовуючи єдині API для покращення міжплатформного обміну даними. Механізм виявлення та усунення несправностей, та відновлення роботи ПЗ (Fault detection, isolation and recovery – FDIR), розглянутий в публікації, базується на ієрархічних рівнях, що дозволяє відновлювати несправності на нижчих рівнях, гарантує ізоляцію і мінімізує обсяг впливу.

При децентралізованому підході реалізується вбудовування кількох порівняно потужних процесорів/мікроконтролерів на кожній системній платі. Тому споживання енергії, складність, вартість і, нарешті, ймовірність прихованої помилки досягає максимуму. Велика проблема в таких розподілених, але монолітних системах виникає зі складністю тестування, коли зусилля та час, витрачені на пошук помилок, є дуже витратними, а проект загалом є дуже обмежений з точки зору часу та ресурсів.

Сучасний підхід до використання операційних систем реального часу (ОСРЧ), таких як FreeRTOS, embOS та Salvo, дозволяє належним чином абстрагувати складність та модульність складної системи бортового обчислювача (БО). Однак основна проблема монолітного вбудованого програмного забезпечення, яка все ще є загальною практикою, залишається актуальною. Монолітна архітектура при розробці призводить до викликів, які зазвичай включають у себе складність та час інтеграції, багато ітерацій перепроектування обміну даними та API, складне виявлення та виправлення помилок, складне повторне використання тощо.

Актуальні виклики при розробці монолітного програмного забезпечення для CubeSat полягають у

складності етапів розробки, відпрацювання, супроводження та експлуатації, особливо в випадку декількох географічно або культурно розгалужених команд. У монолітних програмах весь код зосереджений в одному блоці, що може призводити до конфліктів між розробниками. Крім того, наявність значної кількості взаємозалежних компонентів у монолітних рішеннях ускладнює внесення змін. Масштабованість часто стає проблемою, особливо при додаванні нових функцій або необхідності підтримувати зростання користувацької бази. Це також ускладнює процес тестування через великий обсяг коду та його взаємозалежності, що може призвести до неповного покриття тестами та появи дефектів. Таким чином, великі монолітні системи потребують значних зусиль для впровадження та підтримки, оскільки будь-які зміни часто вимагають повного перезапуску системи.

2. Мікросервісний підхід до побудови бортового ПЗ

Для вирішення цих проблем ми пропонуємо застосувати порівняно нові принципи побудови вбудованого програмного забезпечення, такі як мікросервісна архітектура (рис. 5) та концепція контейнеризації.

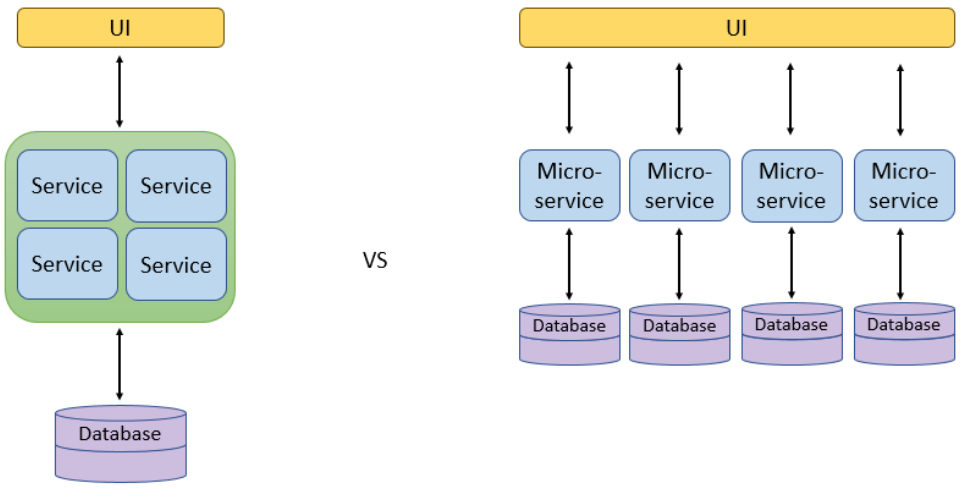
Архітектура мікросервісів – це підхід до розробки програмного забезпечення, при якому програмний застосунок розбивається на невеликі, незалежні та слабко пов'язані між собою сервіси, що взаємодіють через API. Зв'язок між цими мікросервісами може бути реалізований за допомогою протоколів HTTP, WebSockets, AMQP або навіть MQTT.

Стаття [13] систематизує поточний стан досліджень у сфері перетворення монолітних застосунків на мікросервісну архітектуру, визначає ключові підходи та інструменти, а також окреслює майбутні напрямки для вдосконалення цього процесу. Серед наведених у статті метрик, які використовуються для оцінки продуктивності генерованих мікросервісів у порівнянні з вихідним монолітним застосунком, для подальшого дослідження виділимо наступні:

- час виконання (execution time) – час, необхідний для виконання певних операцій або завдань;
- ефективність (efficiency) – порівняння часу виконання монолітного застосунку та мікросервісної версії для оцінки продуктивності перетвореної системи.

Мікросервіс є окремим сервісом, створеним для забезпечення функціональності застосунку та виконання відокремлених завдань у мікросервісній архітектурі. Кожен мікросервіс обмінюється обчислювальним контекстом з іншими сервісами через прості інтерфейси для вирішення бізнес-задач. Основні переваги мікросервісів полягають у тому, що вони незалежно встановлювані, слабко зв'язані, організовані навколо бізнес-завдань (функцій) та розроблені та підтримуються невеликою окремою командою розробників ПЗ.

Ідея використання мікросервісної архітектури відкриває можливості міграції для спочатку монолітного програмного забезпечення за допомогою шаблону проектування застосунку Strangler. Процес такої міграції називається «удушенням моноліту». Галузь розробки програмного забезпечення швидко розвивається в напрямку використання такого підходу, і існує ціла низка метрик та інструментів, пов'язаних з цією темою [14].



а) б)

Рис. 5. Порівняння монолітної (а) та мікросервісної (б) архітектур побудови ПЗ

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

3. Декомпозиція та контейнерізація як наступний крок до побудови надійного бортового ПЗ

Контейнеризація, згідно з компанією IBM [15], яка є одним з головних ініціаторів розробки концепції та її впровадження в промисловість, полягає в упакуванні певного функціонального додатку ПЗ в окремий файл – контейнер, який стабільно розгортається та працює на будь-якій інфраструктурі [16]. Контейнери є більш переносними та ефективними з точки зору використання ресурсів, ніж віртуальні машини (ВМ), і вони стали «де факто» головними обчислювальними одиницями сучасних хмарних програм, успішно переходячи й на менші апаратні платформи.

Основна перевага контейнеризації полягає в тому, що такий підхід дозволяє «написати один раз (на одній платформі) і запустити будь-де». Контейнерізація також пропонує інші переваги, такі як ізоляція та стійкість до помилок [17], легке керування, спрощена безпека тощо.

Оскільки значення вбудованих систем та відповідних бізнес-застосунків зростає, а потужність апаратури збільшується, розвиток мікросервісної архітектури та її упакування в контейнери на вбудованих системах просувається вперед. Автори статті [18] пропонують використовувати підхід контейнеризації та розподіляти типові завдання бортового програмного забезпечення наносупутника серед контейнерів (рис. 6).

Замість монолітної системної архітектури, з використанням контейнеризації програмне забезпечення можна реалізувати по-іншому. Замість багатьох різних монолітних компонентів у системі – бізнес та керуюча логіка цих модулів переносяться в контейнери. Кожен контейнер буде представляти певний компонент системи, і таким чином, строго розбивати функції між контейнерами (рис. 7).

У цьому підході застосування централізованих методів побудови ПЗ «Event Hub» та «Event Bus», які використовуються для обміну даними та командами, може бути покращено шляхом введення патерна проєктування «відкритий Event Bus». Такий патерн буде використовуватись у модулі зв'язку інших компонентів CubeSat. Таким чином, управління всією системою може бути синхронізовано та оркестровано за допомогою шаблону «Сага» або йому подібного. У той же час такий підхід вирішує проблему складних та неузгоджених між собою багаторазових додатків вбудованого програмного забезпечення для різних модулів і, отже, зменшує загальну складність проекту.

Для цього першим кроком необхідно обрати належне контейнерне середовище виконання (фреймворк). Після проведення аналізу на системному рівні, було з'ясовано, що більшість контейнерних середовищ базується на принципах WebAssembly, які виконують інтерпретацію програмного коду в реальному часі. WebAssembly спочатку був розроблений організацією W3C як платформа для веб-технологій та застосунків. Пізніше, через його популярність, він став хорошим вибором для кросплатформного середовища виконання контейнерів для системних та бізнес застосунків. Частково його популярність обумовлена підтримкою типових для вбудованих систем мов програмування C, C++ та Rust.

На сьогодні існує понад 35 реалізацій високопродуктивних машин WebAssembly, приблизно половина з яких активно підтримується та продовжує свій життєвий цикл. Під час аналізу WebAssembly-реалізацій контейнерних середовищ виконання контейнерів, вказаних вище, основна увага була зосереджена на двох аспектах: простоті міграції та сумісності з найбільш поширеними апаратними платформами, наявності вже готових портів для легко доступних апаратних платформ COTS, а також відкритості ПЗ середовища, щоб зберегти концепцію компонентів COTS для наносупутників CubeSat.

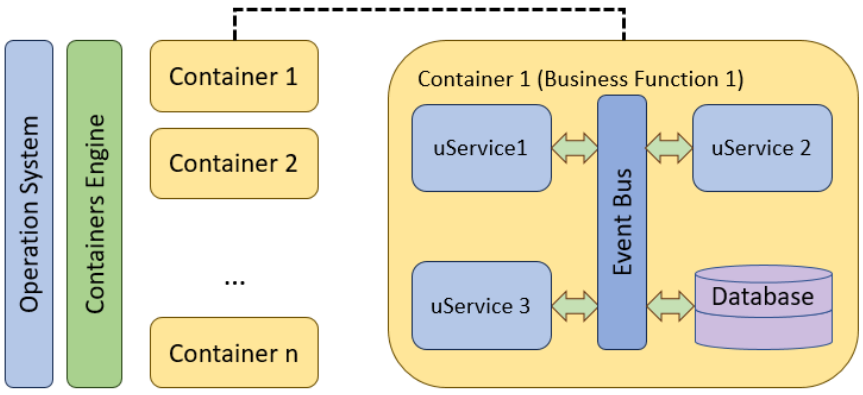


Рис. 6. Мікросервіси, які розподілені у контейнери відповідно до бізнес-функцій

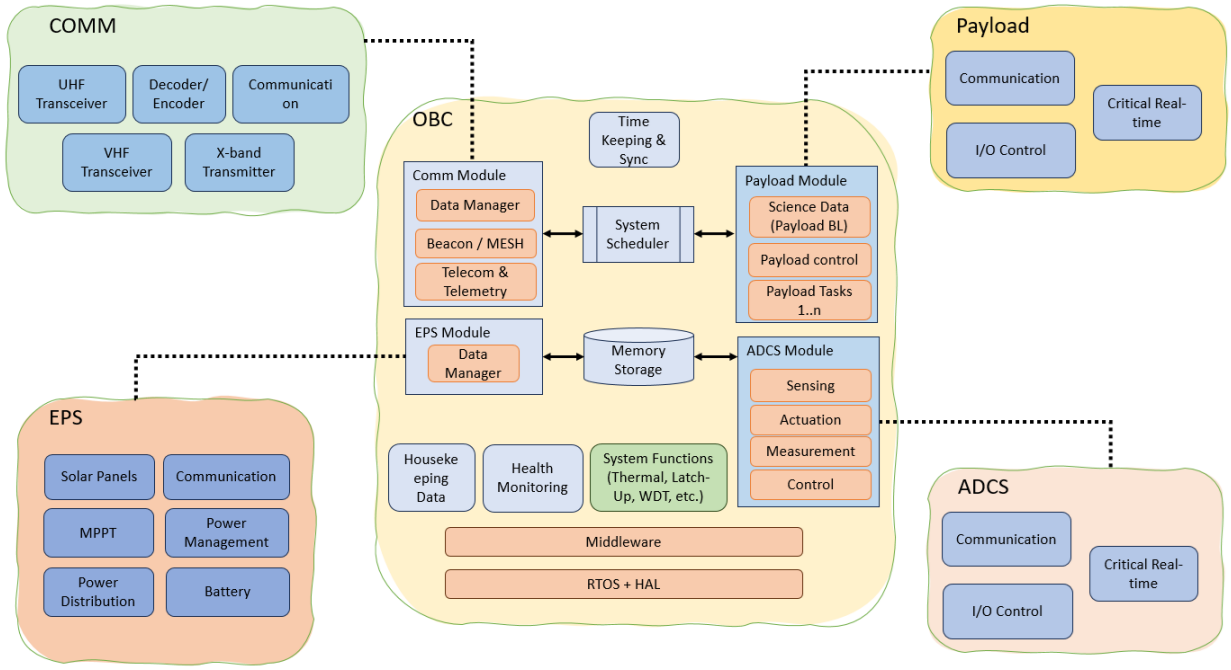


Рис. 7. Запропонована контейнерна структура програмного забезпечення бортового програмного забезпечення CubeSat

Як основу для реалізації середовища виконання контейнерів на типовій апаратній платформі CubeSat був обраний інтерпретаторний двигун WASM3 [19]. WASM3 повністю підтримує енергоефективне та низькопродуктивне обладнання БО CubeSat. Важливим аргументом для вибору є те, що WASM3 розроблено та активно підтримується в Україні, що дозволить авторам встановити прямий зв'язок з командою розробників WASM3 та забезпечити подальшу роботу з використання контейнерів для застосунків CubeSat.

Поєднання переваг мікросервісів і контейнеризації – це вирішення проблеми монолітного коду, шляхом розподілу мікросервісів у набір контейнерів відповідно до бізнес-функцій. Важливими перевагами контейнеризації є можливість розробки та тестування контейнера на звичайному персональному комп'ютері, ізоляція від інших розробників для уникнення конфліктів та спрощене профілювання продуктивності контейнера. Кожен контейнер буде складатися з набагато меншого піднабору мікросервісів, пов'язаних із високорівневими типовими завданнями наносупутника CubeSat, наприклад, підсистема комунікації – COM, або підсистеми визначення та керування орієнтацією – ADCS.

4. Постановка задачі

Головна мета цього дослідження полягає у практичному обґрунтуванні доцільності застосування архітектури мікросервісів та методу контейнеризації у бортовому програмному забезпеченні наносупутника CubeSat. Мікросервісний підхід відкриває ширші можливості для самостійної розробки окремих студентських проектів та їхнього подальшого об'єднання в цілісну систему. Втім, постає нагальна потреба оцінити додаткове навантаження на обчислювальні ресурси, спричинене функціонуванням контейнерів. Адже незаперечним є той факт, що монолітна програма, скомпільована та оптимізована під конкретний процесор, завжди демонструватиме вищу швидкодію порівняно з програмою, яка виконується через середовище виконання контейнерів - інтерпретатор.

Дослідження ставить для вирішення наступні завдання:

– обґрунтувати необхідність пошуку інноваційних архітектурних підходів до розробки бортового програмного забезпечення наносупутника CubeSat, здійснити ґрунтовний порівняльний аналіз переваг і недоліків монолітної та мікросервісної архітектур у контексті бортового ПЗ, зробити вибір програмного середовища виконання контейнерів для реалізації методу контейнеризації та окреслити типову структуру програмного забезпечення CubeSat;

– розглянути процес перенесення (портування) контейнерного середовища виконання WASM3 під керування операційної системи FreeRTOS, розробити план проведення експериментального дослідження для порівняння продуктивності різних апаратних платформ, визначити модель та метод обробки експериментальних результатів;

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

– представити результати досліджень та їх обговорення;

– розглянути питання технічної можливості та практичної доцільності впровадження методу контейнеризації у бортовому програмному забезпеченні наносупутника CubeSat, запланувати напрями майбутніх досліджень.

5. Методи досліджень

5.1. Стратегія портування контейнерного середовища виконання WASM3 до керування операційною системою FreeRTOS

Концепція реалізації контейнеризації в БО CubeSat (рис. 8) базується на наступних положеннях:

– кожен контейнер представляє або функціональний блок CubeSat (система орієнтації та стабілізації, система електропостачання тощо), або/та окрему функцію бортового комп'ютера чи корисного навантаження;

– під час розробки і процесів верифікації і валідації кожного окремого контейнера – окрема команда або член команди розробляє кожний окремий контейнер;

– завдяки крос-платформній підтримці контейнерів, розробка та верифікація кожного окремого контейнера може здійснюватися на ПК, а не обов'язково на апаратному забезпеченні CubeSat.

Процес так званого «портування» – це процес розробки та адаптації обраної частини програмного забезпечення до конкретної операційної системи реального часу та певної апаратної платформи. Для забезпечення функціонування WASM3 на вбудованих системах різних типів, необхідно адаптувати наступну базову інфраструктуру:

– файлову систему для зберігання контейнерів та забезпечення можливості їх завантаження для подальшого виконання;

– командний інтерфейс (CLI), який надає засоби керування контейнерами, зокрема їх запуск, зупинку та завантаження, а також дозволяє користувачеві відстежувати системні параметри завантажених контейнерів в режимі реального часу;

– HTTP сервер, який забезпечує простий та ефективний механізм бездротової передачі (Over-The-Air) образів контейнерів з ПК на вбудовану платформу. Для зв'язку з земними станціями передбачено використання протоколу AX.25, інкапсульованого в HTTP.

Комплексна контейнерна архітектура програмного забезпечення БО, що враховує взаємодію з наземною станцією та використовує широко поширену операційну систему реального часу FreeRTOS від Amazon, представлена на рисунку 9.

У нашому випадку вже існує портований код для вбудованих систем, наданий спільнотою через сховище GitHub. Найбільш актуальним існуючим портованим кодом для наших завдань виявився ESP32-IDF, тому його обрано як основу для подальших експериментальних робіт у цій статті. Під час портування були зроблені наступні припущення:

– кожен контейнер працює як окреме завдання FreeRTOS;

– контейнери працюють з пріоритетом і плануванням за принципом кругового розподілу часу (round robin) – тобто з однаковим пріоритетом;

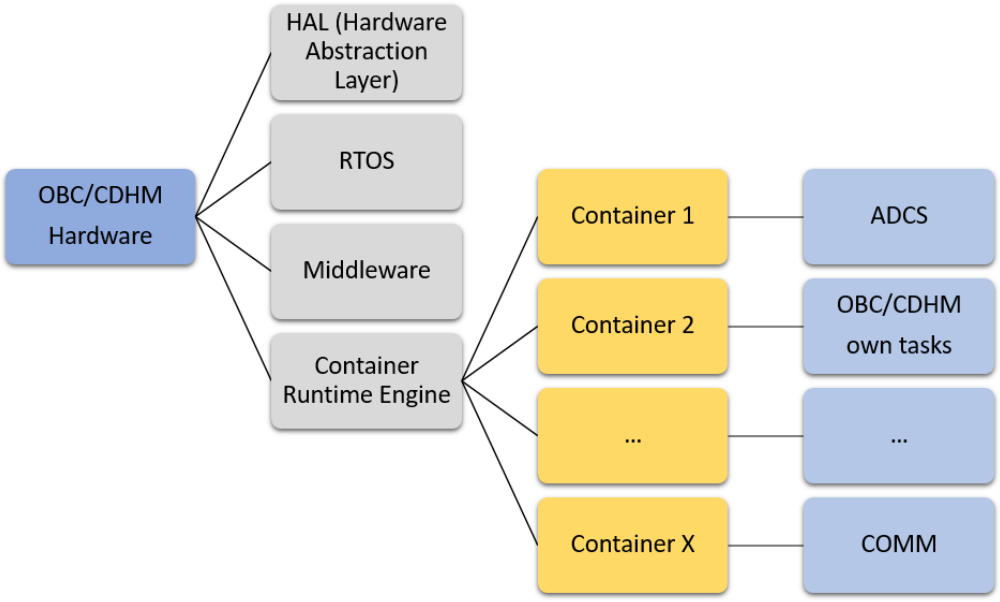


Рис. 8. Загальна ієрархічна контейнерна архітектура ПЗ БО наносупутника CubeSat

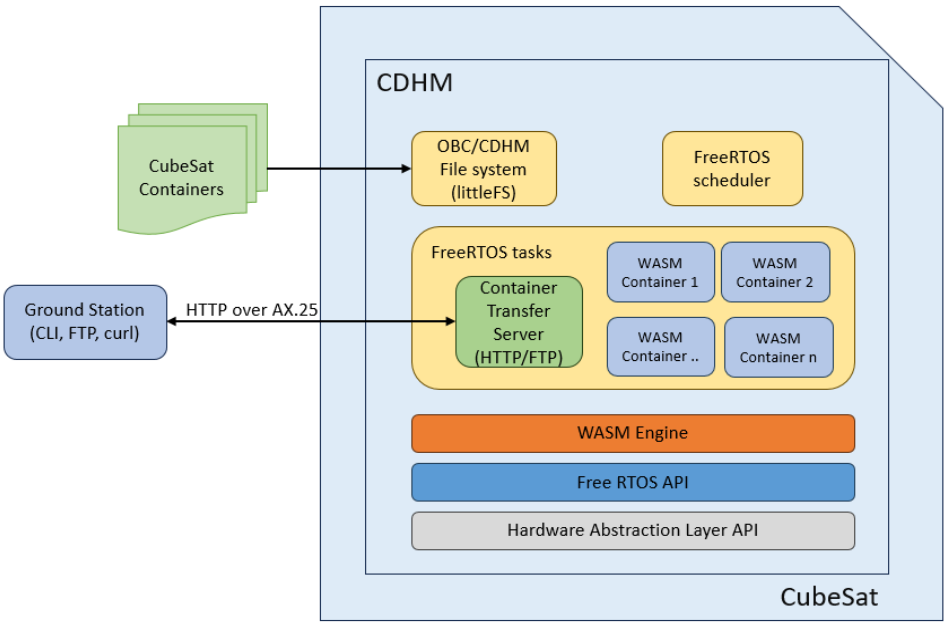


Рис. 9. Комплексна контейнерна архітектура ПЗ ОВС

– файлова система використовується для зберігання контейнерів, і кожен контейнер завантажується перед його використанням;

– загальне портування виконується за допомогою API, специфічних для мікроконтролера та ОСРЧ FreeRTOS.

Для реалізації файлової системи було обрано вбудовану файлову систему з низьким рівнем навантаження littleFS, яка широко використовується в індустрії і є основним кандидатом для використання у CubeSat. Усі бінарні додатки контейнерів постійно зберігаються в файловій системі, тому вони готові до роботи відразу після завантаження системи. Для віддаленого завантаження контейнерів з наземної станції можна розглянути використання спрощеного HTTP/FTP сервера. Схематично процес завантаження контейнера до ОВС/CDHM «Боривітер» представлено на рис. 10, а процес запуску/зупинки завдань ОСРЧ на рис. 11. UML діаграма послідовностей (рис. 11) передбачає, що в один момент часу працює лише один контейнер. Якщо контейнер вже існує в файловій системі, потрібно лише виконати команду «START». У випадку реалізації на CubeSat така команда може надходити через будь-який з каналів телеметричного зв'язку або від системного планувальника, а не лише через інтерфейс завантаження контейнера (наприклад, через HTTP/FTP-сервер на наземній станції).

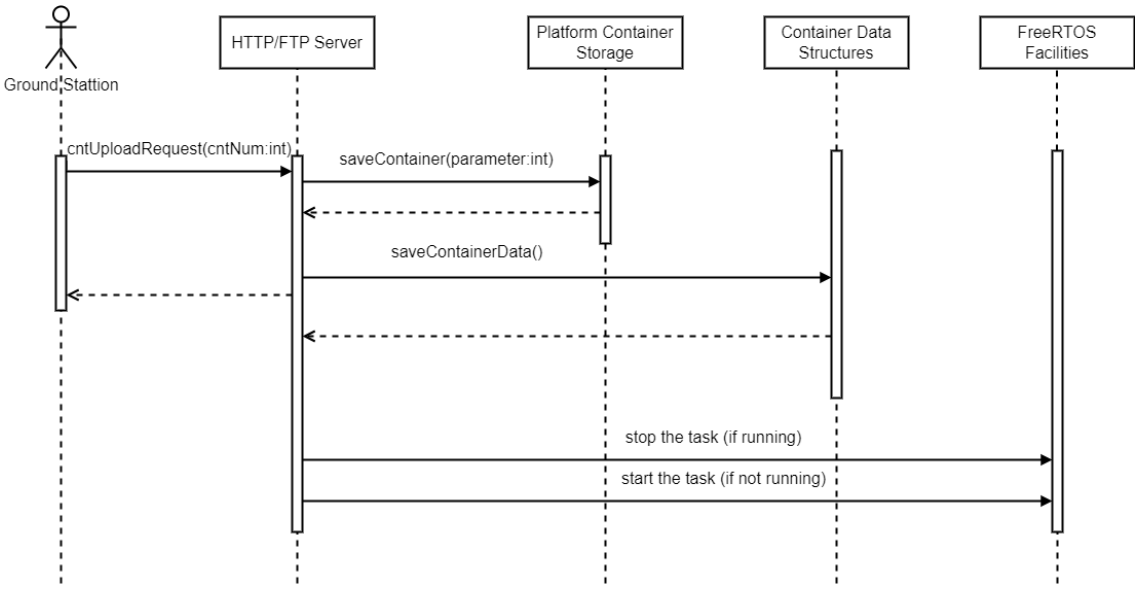


Рис. 10. Діаграма послідовності завантаження контейнера WASM3 до ОВС/CDHM

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

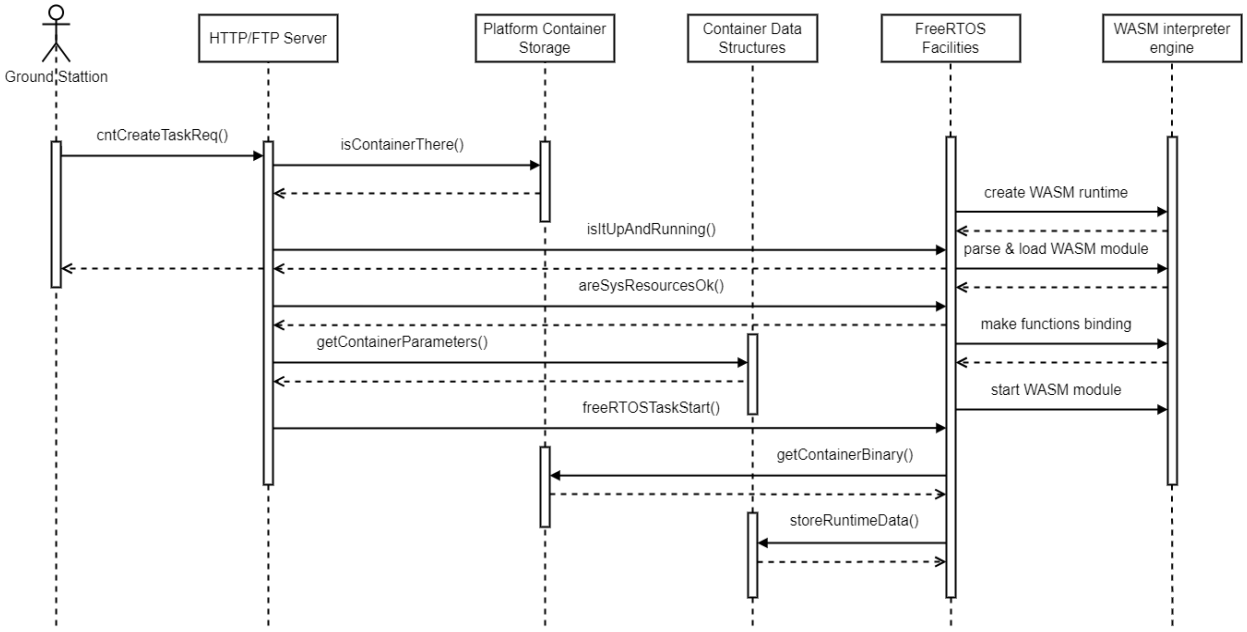


Рис. 11. Діаграма послідовності дистанційного керування контейнером WASM3 на CDHM (*START/STOP*)

5.2. Планування експерименту

Щоб порівняти продуктивність різних апаратних платформ і оцінити додаткові витрати процесорних ресурсів при переході на мікросервісну архітектуру на основі інтерпретатора WASM3, були виконані наступні кроки:

- дві апаратні платформи: відкрита апаратна платформа, подібна до STEM – Cortex-M0+ RP2040 від Raspberry та нова універсальна платформа БО CubeSat – «Боривітер»/«Falco» [20], розроблена авторами статті на базі ARM Cortex-M7 Microchip (Atmel) ATSAM V71Q21 (рис. 12, а, б, в, г);
- дві програмні архітектури: нативний застосунок «bare metal», розроблений за допомогою мови C без операційної системи, та застосунок, розроблений на мовах C та C++ у вигляді мікросервісу та кроскомпільований для інтерпретатору WASM3, який визначає портативний формат двійкового коду та відповідний текстовий формат для виконуваних програм;
- чотири алгоритми з різною обчислювальною складністю (таблиця 2), серед яких тільки FFT має велику кількість операцій з даними у форматі float;
- шість різних розмірів вхідного масиву, які визначаються як ряд $N = 2^n$, оскільки алгоритм швидкого перетворення Фур'є має саме таке обмеження: N = 64, 128, 256, 512, 1024, 2048.

Як відомо, кількість операцій порівняння та перестановки визначає складність алгоритму сортування бульбашкою. Кількість операцій порівняння буде залежати тільки від розміру вхідних даних; кожен елемент порівнюється з усіма іншими:

$$N_{comparison} = \frac{(N-1)*N}{2}. \quad (1)$$

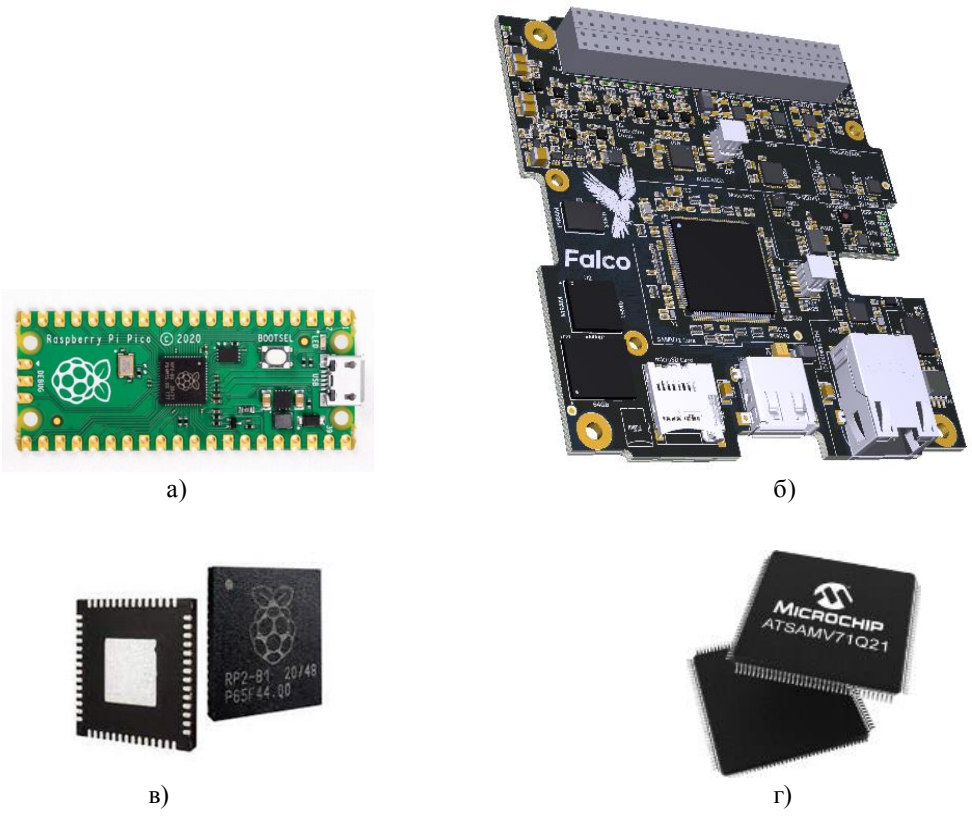
Кількість операцій перестановки залежить від вхідних даних. Якщо вхідний масив вже правильно відсортований, кількість перестановок буде 0; якщо він відсортований у зворотному напрямку, кількість перестановок дорівнюватиме кількості операцій порівняння. Якщо вхідний масив є випадковим, можна лише гарантувати, що кількість перестановок буде в діапазоні

$$N_{permutation} \in [0, \frac{(N-1)*N}{2}]. \quad (2)$$

Щоб оцінити повторюваність результатів вимірювань та основні статистичні показники, проводиться 100 вимірювань для кожного набору: платформа, архітектура, алгоритм і розмір вхідних даних. Таким чином, загальна кількість вимірювань дорівнює:

$$2 \text{ платформи} \times 2 \text{ архітектури} \times 4 \text{ алгоритми} \times 6 \text{ різних розмірів } N \times 100 \text{ повторень} = 9600.$$

Відповідно, ми розглянемо два варіанти вхідних даних для алгоритму бульбашкового сортування: вже правильно відсортований масив і масив, відсортований у зворотному напрямку.



а) б)

в) г)

Рис. 12. Апаратні платформи (а,б) та процесори (в, г), що були використані для експериментальної роботи: а) – Raspberry Pi Pico Dev. Kit, б) – Falco SBC/CDHM 1.0, в) – RP2040, г) – ATSAM V71Q21

Таблиця 2

Алгоритми, які використані в експерименті

№	Клас часової складності	Алгоритм	Часова складність в О-нотації	Модель складності
1	Лінійний час	CRC – обчислення контрольної суми	O(N)	$\tau_1(N) = a_{1,0} + a_{1,1} \cdot N$
2	Лінійно-логарифмічний час	FFT – швидке перетворення Фур'є	O(N log N)	$\tau_2(N) = a_{2,0} + a_{2,1} \cdot N \cdot \log(N)$
3	Лінійний час	BS(P) Бульбашкове сортування з вже правильно відсортованим масивом	O(N)	$\tau_3(N) = a_{3,0} + a_{3,1} \cdot N$
4	Квадратичний час	BS(R) Бульбашкове сортування з масивом, відсортованим у зворотному напрямку	$O(N^2)$	$\tau_4(N) = a_{4,0} + a_{4,1} \cdot N^2$

5.3. Модель та метод параметричної ідентифікації

Модель витрат часу на виконання алгоритмів з різною часовою складністю, яка дозволяє з достатньою точністю оцінити додаткові витрати обчислювальних ресурсів при переході до моделі мікросервісів, які розгалужені в контейнери виконання, може бути отримана, якщо буде враховано початкову складність алгоритму та обсяг переданих вхідних та вихідних даних:

$$T_i(N) = b_0 + b_1 \cdot \tau_i(N) + b_2 \cdot N + b_3 \cdot N_{i,out}, \qquad (3)$$

де $T_i(N)$ – час виконання і-го алгоритму при розмірі вхідних даних N,

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

$\tau_i(N)$ – складність і-го алгоритму,
N – розмір вхідних даних,
$N_{i,out}$ – об'єм вихідних даних і-го алгоритму:

$$N_{1,out} = N_{2,out} = 1, \quad (4)$$

$$N_{3,out} = N_{4,out} = N, \quad (5)$$

де коефіцієнти b_0 ... b_3 є невідомі параметри.

Метод параметричної ідентифікації базується на припущенні, що у випадку лінійної або плоскої моделі пам'яті центральний процесор може прямо (і лінійно) адресувати всі доступні місця пам'яті без використання переключення банків, сегментації пам'яті або схем підкачки.

У вищезазначеному випадку формули (3) ... (5) можуть бути спрощені, оскільки дані передаються через зв'язки, які не вимагають додаткового часу для копіювання їх з одного адресного простору до іншого:

$$T_i(N) = b_0 + b_1 \cdot \tau_i(N), \quad (6)$$

де коефіцієнти b_0 , b_1 є невідомі параметри.

Ідентифікація параметрів моделі виконується в наступних кроках.

1. Для кожного набору:

$$\{\text{платформа}, \text{архітектура}, \text{алгоритм}, N\}$$

обчислюється середнє арифметичне ($\bar{\tau}$) та стандартне відхилення (SD) вимірювань:

$$\bar{\tau} = \frac{1}{N}\sum_{j=1}^{N} t_j, \qquad SD = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(t_j - \bar{\tau})^2}, \quad (7)$$

де t_j – результат вимірювання (час, необхідний для одноразового виконання алгоритму).

2. Результат вимірювання t_j вважається помилкою і відкидається, якщо:

$$t_j \notin [\bar{\tau} - 3 \cdot SD, \bar{\tau} + 3 \cdot SD]. \quad (8)$$

3. Визначення невідомих параметрів a_i з таблиці 2 виконується методом найменших квадратів.

4. Визначення невідомих коефіцієнтів $b_0,\ldots b_3$ рівняння (3) у випадку використання WASM3 виконується також методом найменших квадратів з відомими параметрами a_i, отриманими в пункті 3.

6. Результати експерименту

Експериментально отримані залежності середнього часу виконання чотирьох алгоритмів на двох платформах і двох архітектурах програмного забезпечення при різних розмірах вхідних даних (рис. 13) відповідають моделям, представленим у таблиці 2. В log-log координатах – це лінійні залежності, нахил яких співпадає з теоретично визначеною часовою складністю алгоритмів.

Застосування методу найменших квадратів для обробки експериментальних даних підтверджує гіпотезу, що витрати часу на передачу даних між різними обчислювальними контекстами і запуск підпрограми чи сервісу в моделі плоскої пам'яті можуть бути повністю проігноровані. Це пояснюється тим, що такі витрати не перевищують $3*10^{-6}$ мс, що є менше за роздільну здатність системного таймера (коефіцієнти $a_{j,0}$ в таблиці 3).

Порівняємо час, витрачений на роботу різних функцій у різних архітектурах та на різних платформах. Різниця у функціях полягає у різній алгоритмічній складності (O(N) для CRC і O(N log N) для FFT) та різних типах даних, з якими ці функції працюють (лише цілі для CRC і дійсні числа для FFT). З наведених алгоритмів тільки алгоритм перетворення FFT дозволяє оцінити ефективність операцій з числами з плаваючою комою, тому необхідно відокремити аналіз цих обчислень від роботи з цілими числами. Це важливо для розуміння різниці у часі, витраченому бортовим комп'ютером на операції з цілими числами та з числами з плаваючою комою.

Співвідношення витрат часу в різних архітектурах (рис. 14) є принциповим показником, оскільки він характеризує накладні витрати на перехід від попередньо скомпільованої програми до її інтерпретації в середовищі WASM3.

При роботі з числами з плаваючою комою (перетворення FFT) RP2040 показує найкращі показники співвідношення, а для нової платформи це найгірша можлива проблема, що пояснити достатньо важко. Можливим поясненням є те, що існуюча реалізація двигуна WASM3 неефективно використовує можливості модуля операцій з плаваючою комою (FPU) в складі ATSAMV71.

Висновки

В роботі надано експериментальну оцінку накладних витрат при переході від нативної архітектури програмного забезпечення до мікросервісної архітектури на базі контейнерів WASM3, побудовано математичну модель, яка дозволяє визначити відносні накладні витрати часу при використанні контейнеризації.

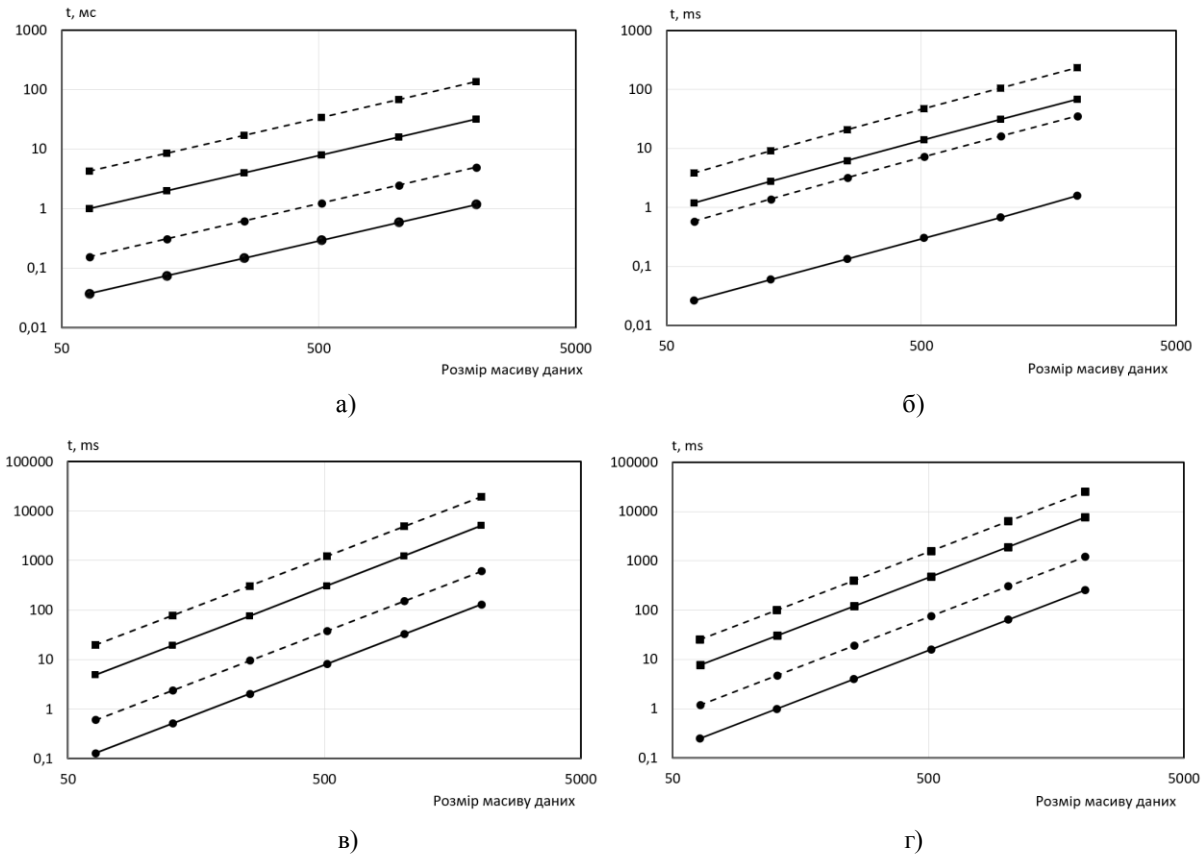


Рис. 13. Експериментальні результати та моделі залежності середнього часу виконання алгоритмів від розміру вхідного масиву:
а) – обчислення контрольної суми, б) – швидке перетворення Фур'є, в) – бульбашкове сортування попередньо відсортованого масиву, г) – бульбашкове сортування масиву, попередньо відсортованого в зворотному напрямку,
де – – RP2040, — ATSAMV71, ■ WASM3, ● Native

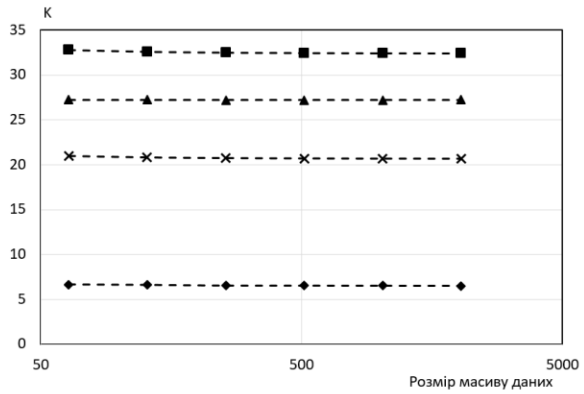
Таблиця 3

Експериментально визначені параметри моделі

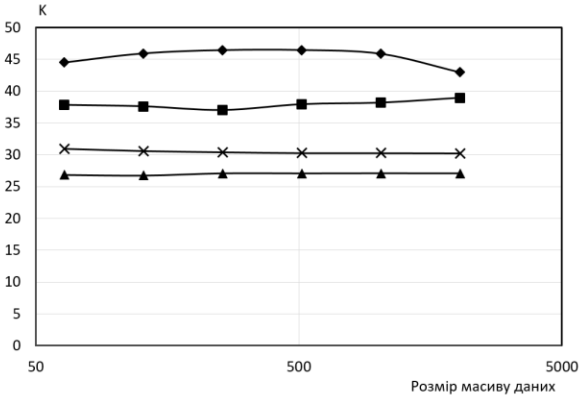
	$a_{j,0}$				$a_{j,1}$			
	ATSAMV71		RP2040		ATSAMV71		RP2040	
	Native	WASM3	Native	WASM3	Native	WASM3	Native	WASM3
CRC, j=1	3.22E-08	8.74E-07	1.37E-07	3.72E-06	2.56E-07	6.93E-06	1.08E-06	2.95E-05
FFT, j=2	3.19E-08	1.60E-06	8.24E-07	5.37E-06	3.61E-07	1.54E-05	8.03E-06	5.23E-05
BS – масив, відсортований в зворотньому напрямку, j=3	1.23E-10	2.03E-07	3.83E-09	2.71E-07	6.09E-05	1.84E-03	2.92E-04	6.04E-03
BS – відсортований масив, j=4	1.22E-09	2.89E-06	4.51E-09	3.50E-07	3.09E-05	1.20E-03	1.44E-04	4.67E-03

Мікросервісна архітектура спрощує процеси розробки та інтеграції, але суттєво збільшує час обчислень при використанні контейнерів WASM3. В умовах експлуатації це призводить не лише до порушення обмежень реального часу, але й до збільшення енергоспоживання бортового комп'ютера, що є дуже принциповим зауваженням, оскільки з раніше опублікованих робіт відомо, що в деяких промислових застосунках Інтернету речей витрати на роботу мікроконтролера можуть досягати 50% від загального енергоспоживання пристрою.

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)



а)



б)

Рис. 14. Співвідношення витрат часу при виконанні запланованих експериментом завдань – «пряма реалізація» до «контейнерної реалізації» (K – безрозмірні відносні накладні витрати (рази)) для двох платформ: а) – RP2040, б) – ATSAMV71, де ▲ – завдання – «обчислення контрольної суми», ◆ – завдання «швидке перетворення Фур'є», ■ – завдання «бульбашкове сортування попередньо відсортованого масиву», ✕ – завдання «бульбашкове сортування масиву, попередньо відсортованого в зворотному напрямку»

Недоліком роботи є те, що не досліджено питання організації сумісної та паралельної роботи окремих незалежних програмних модулів і мікросервісів у багатозадачній системі. Стаття обмежує розглянуті питання лише роботою одного контейнера під керівництвом багатозадачної операційної системи з її планувальником і диспетчером. Отримані результати слід сприймати як оптимістичну оцінку мінімального часу, який буде витрачено на роботу того чи іншого програмного застосунку/контейнеру.

Отримані результати є невтішними для прихильників впровадження WASM3 як платформи, що полегшує розробку бортового програмного забезпечення наносупутників. Однак, наші результати дозволяють шукати компроміс між ефективністю процесів розробки та обчислювальною ефективністю, а отже, енергоефективністю або енергетичними витратами на роботу бортового обчислювача.

Подальший розвиток цієї роботи буде полягати в напрямку аналізу відповідності вимогам двох класів систем реального часу та їх обмежень:

- жорсткі обмеження реального часу, коли кожне перевищення заздалегідь визначеного часу є помилкою;
- м'які обмеження, коли необхідно забезпечити лише середнє значення часу, витраченого на виконання всіх алгоритмів.

Також заплановано дослідити можливість поділу задач програмного забезпечення з критичним та некритичним часом виконання на два окремих процесори, таким чином вводячи «гібридну» обчислювальну платформу, що може бути перспективним напрямом подальшого розвитку.

Внесок авторів: розробив апаратне та програмне забезпечення, виконав дослідження – **Олександр Любімов**; запропонував концепцію, перевірив отримані дані, відредагував – **Ігор Туркін**; виконав пошук ресурсів, зробив візуалізацію та провів верифікацію тексту – **Вячеслав Валковий** та **Олександр Лещенко.**

Конфлікт інтересів

Автори заявляють, що немає конфлікту інтересів щодо цього дослідження, фінансового, особистого, авторського чи іншого, який міг би вплинути на дослідження та його результати, представлені в цій статті.

Фінансування

Дослідження проведено в рамках проєкту 2023.04/0143 «*Експериментальне відпрацювання бортового обчислювача безпілотного літального апарата подвійного призначення*» за фінансової підтримки Національного фонду досліджень України.

Доступність даних

Рукопис не має пов'язаних даних.

Використання засобів штучного інтелекту

Автори підтверджують, що при створенні представленої роботи використовували технології штучного інтелекту тільки для вирішення технічних питань, як то визначення УДК, або формування переліку літератури відповідно до вимог оформлення.

Подяки

Автори висловлюють вдячність інжиніринговій компанії “Falco Engineering” LLC за надання до проведення експерименту апаратних платформ та допомогу у налаштуванні та налагодженні інструментів для портування. Для детальнішої інформації https://falco.engineering/.

Усі автори прочитали та погодилися з опублікованою версією цього рукопису.

Література

1. Deepak, R. A. Thinking Out of the Box: Space Science Beyond the CubeSat [Electronics resource] / R. A. Deepak, & R. J. Twiggs // Journal of Small Satellites (JoSS). – 2012. – Vol. 1, No.1. – P. 3-7. – Available at: https://www.jossonline.com/wp-content/uploads/2014/12/0101-Thinking-Outside-the-Box-Space-Science-Beyond-the-CubeSat.pdf.(accessed: 20.05.2024).

2. Liubimov, O. Use of Open-Source Cots/Mots Hardware and Software Platforms for The Build Up of the CubeSat Nanosatellites [Text] / O. Liubimov, & M. Liubimov // Journal of Rocket-Space Technology. – 2023. – Vol. 31, iss. 4. – P. 138-147. DOI: 10.15421/452318.

3. Kulu, E. Nanosatellites Through 2020 and Beyond [Electronics Resource] / E. Kulu // CubeSat Developers Workshop 2021, April 9, 2021. DOI: 10.13140/RG.2.2.32735.59048.

4. Bouwmeester, J. Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology [Text] / J. Bouwmeester, & J. Guo // Acta Astronautica. – 2010. – Vol. 67, iss. 7-8. – P. 854-862. – DOI: 10.1016/j.actaastro.2010.06.004.

5. Istrate, O. The Use of Technology in STEM Education. An Empirical Research [Text] / O. Istrate, C. Mironov, A. Popovici // Journal of Pedagogy. – 2019. – Vol. 1. – P. 73-91. – DOI: 10.26755/RevPed/2019.1/73.

6. Birzina, R. Technology as a Tool in STEM Teaching and Learning [Electronic resource] / R. Birzina, & T. Pigozne // Rural environment. Education. Personality. – 2020. – Vol. 13. – P. 219-227. DOI: 10.22616/reep.2020.026.

7. Types of Nanosats [Electronic Resource] / Nanosats Database 2024. – Available at: https://www.nanosats.eu/img/fig/Nanosats_types_2024-05-31_large.png. (accessed: 20.05.2024).

8. Poghosyan, A. CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions [Text] / A. Poghosyan, A. Golkar // Progress in Aerospace Sciences. – 2017. – Vol. 88. – P. 59-83. DOI: 10.1016/j.paerosci.2016.11.002.

9. CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers [Electronic Resource] / NASA CubeSat Launch Initiative. – Available at: https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf.(accessed: 20.05.2024).

10. Design Guidelines for General-Purpose Payload-Oriented Nanosatellite Software Architectures [Text] / C. Araguz, M. Marí, E. Bou-Balust, E. Alarcon, & D. Selva // Journal of Aerospace Information Systems. – 2018. – Vol. 15, iss. 3. – P. 107-119. DOI: 10.2514/1.I010537.

11. Flight Software Development for a CubeSat Application [Text] / K. V. C. K. de Souza, Y. Bouslimani, & M. Ghribi // IEEE Journal on Miniaturization for Air and Space Systems. – 2022. – Vol. 3, iss. 4. – P. 184-196. DOI: 10.1109/JMASS.2022.3206713.

12. Reusable and Reliable Flight-Control Software for a Fail-Safe and Cost-Efficient CubeSat Mission: Design and Implementation [Text] / I. Latachi, T. Rachidi, M. Karim, & A. Hanafi // Aerospace. – 2020. – Vol. 7, iss. 10. – Article No. 146. DOI: 10.3390/aerospace7100146.

13. Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review [Text] / Y. Abgaz, A. McCarren, P. Elger, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yilmaz, J. Buckley, & P. Clarke // IEEE Transactions on Software Engineering. – 2023. – Vol. 49, iss. 8. – P. 4213-4242. DOI: 10.1109/TSE.2023.3287297.

14. Al-Debagy, O. Extracting Microservices’ Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach [Text] / O. Al-Debagy, P. Martinek // 15th International Conference of System of Systems Engineering (SoSE). – Budapest, Hungary, 2020. – P. 289-294. – DOI: 10.1109/SoSE50414.2020.9130466.

15. Susnjara, S. What Is Containerization? [Electronic Resource] / S. Susnjara, & I. Smalley // IBM. – Available at: https://www.ibm.com/topics/containerization. (accessed: 20.05.2024).

16. Containers in the Enterprise [Electronic Resource] / IBM Market Development & Insights. – 2020. – Available at: https://www.ibm.com/downloads/cas/VG8KRPRM. (accessed: 20.05.2024).

17. Tamanaka, G. T. B. Fault-tolerant architecture and implementation of a distributed control system using containers [Text] / G. T. B. Tamanaka, R. V. Aroca, G. A. De Paula Caurin // Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE). – São Bernardo do Campo, Brazil, 2022. – P. 1-6. DOI: LARS/SBR/WRE56824.2022.9995745.

18. Microservice Architecture for Embedded Systems [Text] / S. Wang, C. Du, J. Chen, Y. Zhang, & M. Yang // 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). – Xi’an, China, 2021. – Vol. 5. – P. 544-549. – DOI: 10.1109/ITNEC52019.2021.9587154.

ISSN 1727-7337 (print)
ISSN 2663-2217 (online)

19. Shymanskyy, V. WASM3. GitHub Page [Electronic Resource] / V. Shymanskyy // Github. – Available at: https://github.com/wasm3/wasm3. (accessed 28.04.2024).

20. Сторінка-презентація ДКР авторів статті з розробки БО «Боривітер»/«Falco» [Електронний ресурс] – Режим доступу: https://www.falco.engineering/ (дата звернення: 15.03.2024).

References

1. Deepak, R. A., & Twiggs, R. J. Thinking Out of the Box: Space Science Beyond the CubeSat'. *Journal of Small Satellites (JoSS)*, 2012, vol. 1, iss.1, pp. 3-7. Available at: https://www.jossonline.com/wp-content/uploads/2014/12/0101-Thinking-Outside-the-Box-Space-Science-Beyond-the-CubeSat.pdf (Accessed 20 May 2024).

2. Liubimov, O., & Liubimov, M. Use of open-source Cots/Mots hardware and software platforms for the build up of the CubeSat nanosatellites. *Journal of Rocket-Space Technology*, 2023, vol. 31, iss. 4, pp. 138-147. DOI: 10.15421/452318.

3. Kulu, E. Nanosatellites Through 2020 and Beyond'. *CubeSat Developers Workshop 2021, April 9, 2021*. DOI: 10.13140/RG.2.2.32735.59048.

4. Bouwmeester, J. & Guo, J. Survey of worldwide pico- and nanosatellite missions, distributions and subsystemtechnology. *Acta Astronautica*, 2010, vol. 67, iss. 7-8, pp. 854-862. DOI: 10.1016/j.actaastro.2010.06.004.

5. Istrate, O., Mironov, C., & Popovici, A. The Use of Technology in STEM Education. An Empirical Research. *Journal of Pedagogy*, 2019, vol. 1., pp. 73-91. DOI: 10.26755/RevPed/2019.1/73.

6. Birzina, R., & Pigozne, T. Technology as a Tool in STEM Teaching and Learning. *Rural environment. Education. Personality,* 2020, vol. 13, pp. 219-227. DOI: 10.22616/reep.2020.026.

7. *Types of Nanosats. Nanosats Database 2024*. Available at: https://www.nanosats.eu/img/fig/Nanosats_types_2024-05-31_large.png (Accessed 20 May 2024).

8. Poghosyan, A., & Golkar, A. CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions. *Progress in Aerospace Sciences*, 2017, vol. 88, pp. 59-83. DOI: 10.1016/j.paerosci.2016.11.002.

9. *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers. NASA CubeSat Launch Initiative*. Available at: https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf (Accessed 20 May 2024).

10. Araguz, C., Marí, M., Bou-Balust, E., Alarcon, E., & Selva, D. Design Guidelines for General-Purpose Payload-Oriented Nanosatellite Software Architectures. *Journal of Aerospace Information Systems*, 2018, vol. 15, iss. 3, pp. 107-119. DOI: 10.2514/1.I010537.

11. De Souza, K. V. C. K., Bouslimani, Y., & Ghribi, M. Flight Software Development for a CubeSat Application. *IEEE Journal on Miniaturization for Air and Space Systems*, 2022, vol. 3, iss. 4, pp. 184-196. DOI: 10.1109/JMASS.2022.3206713.

12. Latachi, I., Rachidi, T., Karim, M., & Hanafi, A. Reusable and Reliable Flight-Control Software for a Fail-Safe and Cost-Efficient CubeSat Mission: Design and Implementation. *Aerospace*, 2020, vol. 7, iss. 10, article no. 146. DOI: 10.3390/aerospace7100146.

13. Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering*, 2023, vol. 49, iss. 8, pp. 4213-4242. DOI: 10.1109/TSE.2023.3287297.

14. Al-Debagy, O., & Martinek, P. Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach. *15th International Conference of System of Systems Engineering (SoSE)*, 2020, pp. 289-294. DOI: 10.1109/SoSE50414. 2020.9130466.

15. Susnjara, S., & Smalley, I. *What Is Containerization?* Available at: https://www.ibm.com/topics/containerization (Accessed 12 May 2024).

16. *Containers in the Enterprise. IBM Market Development & Insights*. Available at: https://www.ibm.com/downloads/cas/VG8KRPRM (Accessed 12 May 2024).

17. Tamanaka, G. T. B., Aroca, R. V., & De Paula Caurin, G. A. Fault-tolerant architecture and implementation of a distributed control system using containers. *Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*, pp. 1-6. DOI: 10.1109/LARS/SBR/WRE56824.2022.9995745.

18. Wang, S., Du, C., Chen, J., Zhang, Y., & Yang, M. Microservice Architecture for Embedded Systems. *5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2021, vol. 5, pp. 544-549. DOI: 10.1109/ITNEC52019.2021.9587154.

19. Shymanskyy, V. *WASM3. GitHub Page*. Available at: https://github.com/wasm3/wasm3 (Accessed 28 April 2024).

20. *Storinka-prezentatsiya DKR avtoriv statti z rozrobky BO «Boryviter»/«Falco»* [Page-presentation of the R&D work of the authors of the article on the development of the on-board computer "Boryviter"/"Falco"]. Available at: https://www.falco.engineering/. (Accessed 15 March 2024).

Надійшла до редакції 20.05.2024, розглянута на редколегії 00.08.2024

EXPERIMENTAL EVALUATION OF THE EFFICIENCY OF CONTAINERIZATION TECHNOLOGY IN THE ONBOARD SOFTWARE OF A NANOSATELLITE CUBESAT

Oleksandr Liubimov, Ihor Turkin, Oleksandr Leshchenko, Viacheslav Valkovyi

The **objective of this research** is to investigate the computational efficiency of various system architectures of onboard software for the onboard computers of CubeSat nanosatellites. The **subject** of this study is the computing overhead associated with using the containerization method in the construction of the onboard software for CubeSat nanosatellites. **Purpose**: To experimentally investigate the possibility and feasibility of using the containerization method in the onboard software of CubeSat nanosatellites. **Objectives**: to define the need to find new architectural solutions in the onboard software of the CubeSat nanosatellite; to perform a comparative analysis of the advantages and disadvantages of using monolithic (classical) and microservice architecture in the onboard software of the CubeSat nanosatellite; to justify the choice of the system software environment for executing containers; to determine the typical structure of CubeSat software and the strategy for adapting the WASM3 container environment to the FreeRTOS operating system; to develop a plan; and to conduct an expert evaluation based on the results of the evaluation to formulate conclusions about the possibility and feasibility of using container architecture in the onboard software of the CubeSat nanosatellite. **Conclusions**. This study demonstrates the relevance of developing software for satellites based on microservices and containers. The experimental results allow us to compare the performance of the onboard computer when executing various algorithms implemented using the C programming language ("Bare-Metal" approach) and the architecture based on the microservices approach, branched between containers of the WASM3 environment running under the FreeRTOS, and developed in C and C++ programming languages. The main conclusion of this work is the potential of using the Falco SBC/CDHM hardware platform as an affordable and powerful computing platform for CubeSat nanosatellites.

Keywords: CubeSat; nanosatellite; COTS; onboard computer; containerization; microservices architecture; Software; WASM3; Boryviter/Falco; computation efficiency; computing overhead.

Любімов Олександр Вікторович – асп. каф. інженерії програмного забезпечення, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Туркін Ігор Борисович – д-р техн. наук, проф., зав. каф. інженерії програмного забезпечення, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Лещенко Олександр Борисович – канд. техн. наук, проф., проф. каф. комп'ютерних наук та інформаційних технологій, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Валковий В'ячеслав Сергійович – асп. каф. інженерії програмного забезпечення, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Oleksandr Liubimov – Ph.D. Student of the Department of Software Engineering, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: oleksandr.liubimov@gmail.com, ORCID: 0000-0003-3636-6939, Scopus Author ID: 58099287400.

Ihor Turkin – Doctor of Technical Science, Professor, Head of the Department of Software Engineering, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: i.turkin@khai.edu, ORCID: 0000-0002-3986-4186, Scopus Author ID: 57203145725.

Oleksandr Leshchenko – Candidate of Technical Science, Professor, Professor at the Department of Computer Science and Information Technology, National Aerospace University «Kharkiv Aviation Institute», Kharkiv, Ukraine,
e-mail: o.leshchenko@khai.edu, ORCID: 0000-0001-9405-4904.

Viacheslav Valkovyi – Ph.D. Student of the Department of Software Engineering, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: v.s.valkovyi@khai.edu, ORCID: 0009-0008-7712-4112.