ISSN 2071-1077(print)
ISSN 2663-2411(online

doi: 10.32620/oikit.2026.107.09

УДК 004.42:629.78

О. Б. Лимаренко,
Є. В. Соколова

Аналіз архітектурних методів інтеграції бортових систем наносупутників у хмарні обчислювальні середовища

Національний аерокосмічний університет «Харківський авіаційний інститут»

У роботі проведено комплексний аналіз архітектурних методів оптимізації обчислювальних ресурсів наносупутників стандарту CubeSat в умовах переходу до концепції NewSpace. Розглянуто фундаментальну проблему дисбалансу між експоненційно зростаючими вимогами до бортових обчислень та жорсткими апаратними обмеженнями платформ формату 1U-3U, що характеризуються критичним дефіцитом енергії, високою радіаційною вразливістю компонентів та обмеженим обсягом пам’яті. Виконано детальний порівняльний аналіз ефективності провідних операційних систем реального часу FreeRTOS та Zephyr OS за критеріями латентності перемикання контексту, енергоефективності та надійності, який продемонстрував переваги FreeRTOS для енергодефіцитних місій завдяки мінімальним накладним витратам ядра на рівні 223 машинних циклів. Формалізовано математичну модель енергоспоживання бортового комп’ютера, яка враховує динамічну та статичну потужність з огляду на температурний режим космосу та вплив радіації на зростання струмів витоку в напівпровідникових структурах. Досліджено перспективи впровадження гібридної архітектури «Edge-to-Cloud», що передбачає інтелектуальний офлоудинг ресурсомістких обчислювальних задач у наземні хмарні середовища через сервіси GSaaS (Ground Station as a Service), з детальним аналізом енергетичних та комунікаційних критеріїв доцільності розвантаження. Обґрунтовано необхідність використання програмно-апаратних методів забезпечення радіаційної стійкості, включаючи скрабінг пам’яті та багаторівневі системи сторожових таймерів, що споживають до 20% процесорних ресурсів. Доцільність застосування легковагових технологій контейнеризації на основі WebAssembly підтверджено результатами сучасних експериментальних досліджень, що засвідчують їх придатність для архітектур програмно-визначених супутників із забезпеченням безпечного оновлення програмних модулів безпосередньо на орбіті та ізоляції відмов.
Ключові слова: CubeSat; операційні системи реального часу; розвантаження обчислень; edge computing; хмарні обчислення; енергоефективність.

Вступ

Парадигма NewSpace та еволюція наносупутників

Наразі сучасна аерокосмічна галузь переживає значні зміни, яку фахівці називають «NewSpace». Нова ера характеризується переходом від контролю великих державних агентств з їхніми дорогими великими супутниками до використання менших та розподілених космічних апаратів, зокрема наносупутників стандарту CubeSat. Започаткований наприкінці 1990-х років як освітній проєкт професорів Боба Твіггса (Стенфордський університет) та Хорді Пуіг-Суарі (Каліфорнійський політехнічний університет), він забезпечив уніфікацію підходів до проєктування малих супутників і значно спростив доступ до навколоземної орбіти. Завдячуючи уніфікуванню форм-фактору (базовий блок 1U розміром 10×10×10 см і масою до 1,33 кг), інтерфейсів запуску (P-POD) та підходів до проєктування, воно дозволило у рази знизити вартість розробки та запуску.

Сьогодні CubeSat еволюціонував із суто освітнього інструменту в надійну

платформу для виконання складних комерційних, наукових та оборонних завдань. Аналітичні звіти прогнозують стабільне зростання ринку наносупутників із середньорічним темпом приросту (CAGR) на рівні 16,3% у період до 2030 року [1], що зумовлено попитом на дані дистанційного зондування Землі з високою частотою оновлення, глобальне покриття Інтернету речей (IoT) та проведення астрофізичних досліджень. Проте демократизація космосу приходить із серйозними інженерними викликами. Фізичні обмеження платформи 1U-3U встановлюють чіткі межі для енергоспоживання, тепловідведення і обчислювальної потужності.

Проблема SWaP та обчислювальний дисбаланс

SWaP (Size, Weight, and Power – Розмір, Вага та Потужність) можна вважати ключовою метрикою в проєктуванні будь-якої аерокосмічної системи. Для наносупутників ці обмеження є критичними, бо:

- енергоспоживання: поверхня CubeSat 1U дозволяє розмістити фотоелектричні перетворювачі, що генерують у середньому 1-2 Вт електроенергії. Це змушує інженерів балансувати між живленням системи зв’язку, корисного навантаження та бортового комп’ютера.
- обчислення: зростаючі вимоги до автономності місій, обробки гіперспектральних зображень на борту (On-board Processing) та впровадження алгоритмів штучного інтелекту вимагають чималих потужностей, які важко забезпечити в межах доступного обмеженого енергоспоживання.

Традиційна архітектура «store-and-forward», коли супутник збирає «сирі» дані та передає їх на Землю для обробки, стає неефективною через обмежену пропускну здатність радіоканалів та експоненційне зростання обсягів даних, що у свою чергу створює «пляшкове горлечко», де цінність отриманої інформації знижується через затримку її доставки та обробки [2].

Мета дослідження

Дана оглядова стаття має на меті не лише систематизувати існуючі дані, але й пропонує подивитися на поєднання наносупутників та хмарних екосистем, а також викликів, до яких це призводить.

Основні аспекти наукової новизни роботи полягають у формалізації енергетичної моделі з урахуванням накопиченої іонізуючої дози (TID) та температурного режиму; кількісному порівняльному аналізі операційних систем реального часу в контексті обмежень SWaP; визначенні критерію доцільності обчислювального офлоудингу.

Практичне значення отриманих результатів полягає у формуванні рекомендацій щодо вибору RTOS, обґрунтуванні архітектури Edge-to-Cloud та застосуванні WebAssembly для реалізації концепції програмно-визначеного супутника.

Також оглядова стаття має на меті проведення аналізу архітектурних методів оптимізації ресурсів наносупутників для вирішення суперечності між зростаючими обчислювальними потребами та обмеженнями платформи. У роботі детально розглядається інтеграція бортових систем реального часу (RTOS) з хмарними обчислювальними потужностями через модель «Edge-to-Cloud».

Апаратна архітектура бортових комп’ютерів

Бортовий комп’ютер (ОВС) є «мозком» наносупутника, що відповідає за керування всіма підсистемами, збір телеметрії, обробку команд та виконання наукової місії. Еволюція ОВС демонструє тренд відходу від спеціалізованих радіаційно-стійких компонентів до використання високопродуктивних комерційних рішень (COTS). Також з’являються змішані, тобто гібридні архітектури.

Перехід до COTS-компонентів: Переваги та ризики

Використання COTS-компонентів (Commercial Off-The-Shelf) стало де-факто стандартом для супутників класу CubeSat. Це зумовлено кількома факторами:

- продуктивність: сучасні споживчі SoC випереджають спеціалізовані космічні процесори на декілька поколінь. Наприклад, Cortex-M7 працює на частотах до 480 МГц, тоді як перевірені часом радіаційно-стійкі процесори часто обмежені десятками МГц.
- вартість та доступність: ціна COTS-компонентів суттєво нижча, а доступність засобів розробки (SDK, документація, спільнота) значно спрощує процес створення ПЗ навіть для нещодавніх студентів.
- інтеграція: високий рівень інтеграції периферії (ADC, DAC, PWM, комунікаційні інтерфейси) зменшує кількість компонентів на платі, що критично для обмеженого об’єму 1U.

Однак є суттєве «але» – COTS-компоненти не призначені для роботи в умовах космічного простору, що вимагає впровадження комплексних заходів захисту на системному рівні.

Мікроконтролерні архітектури: Cortex-M7, RISC-V та Rad-Hard рішення

Аналіз попередніх досліджень дозволяє виділити дві домінуючі архітектури для сучасних ОВС:

ARM Cortex-M7: Лідер продуктивності

Мікроконтролери на базі ядра ARM Cortex-M7 (наприклад, Microchip ATSAMV71, STM32H7) є найпопулярнішими для ОВС.

Характеристики: тактова частота 300-480 МГц, суперскалярна архітектура (подвійна видача команд), модуль FPU подвійної точності (DP). Розвинена система кешування (L1 I/D Cache) та тісно пов’язана пам’ять (TCM) забезпечують детермінованість [3].

До прикладу вона використовується в платформі «Boryviter 0.1» [4] та платах ATSAMV71-XULT, демонструючи здатність виконувати складні алгоритми фільтрації в реальному часі.

Vorago VA41630 (Rad-Hard Cortex-M4): Апаратна надійність

Компанія Vorago Technologies в свою чергу пропонує унікальне рішення, що поєднує популярну архітектуру ARM Cortex-M4 з технологією радіаційного зміцнення HARDSIL.

Особливості: VA41630 має вбудовану апаратну підтримку потрійного модульного резервування (TMR) для всіх внутрішніх регістрів та тригерів. Це

означає, що кожен біт зберігається у трьох фізичних комірках, і мажоритарний елемент (voter) автоматично виправляє помилку, якщо одна з комірок змінить стан через влучання частинки.

Пам’ять: містить 256 КБ FRAM (фероелектрична пам’ять), яка є стійкою до радіації та енергонезалежною, а також апаратний EDAC та скрабер.

Cobham Gaisler NOEL-V (RISC-V): Відкрита архітектура

NOEL-V – це м’яке процесорне ядро (Soft Core) на базі RISC-V, розроблене для розгортання на FPGA (наприклад, RTG4 або Kintex UltraScale).

До переваг можна віднести підтримку розширень віртуалізації (Hypervisor), вбудована корекція помилок у кеш-пам’яті, можливість конфігурації Dual-Core Lockstep. Це дозволяє створювати гнучкі системи з високою відмовостійкістю, але ціною значного збільшення енергоспоживання.

Радіаційна стійкість та надійність пам’яті

Космічне середовище характеризується наявністю високоенергетичних частинок (наприклад, протони, важкі іони), які спричиняють поодинокі ефекти (SEE – Single Event Effects) у напівпровідниках.

SEU (Single Event Upset). У SRAM зміна стану біта пам’яті («bit flip»), яка займає значну площу кристала мікроконтролера, це найбільш часта проблема.

SEL (Single Event Latch-up). Паразитне тиристорне замикання, що може призвести до фізичного руйнування чіпа через надмірний струм.

Для забезпечення надійності використовуються програмно-апаратні методи:

1. EDAC (Error Detection and Correction): програмна реалізація кодів корекції помилок (наприклад, коди Ріда-Соломона або Хемінга) для захисту даних у пам’яті. Це критично важливо, оскільки вбудована SRAM часто не має апаратного захисту. Дослідження показують, що програмний EDAC може забирати до 15-20% процесорного часу, тому це критично враховувати при плануванні ресурсів.
2. Скрабінг пам’яті (Scrubbing): періодичне зчитування та перезапис комірок пам’яті для виправлення накопичених помилок до того, як вони стануть невиправними.
3. Watchdog Timers: використання багаторівневої системи сторожових таймерів (внутрішніх та зовнішніх) для перезавантаження системи у разі зависання, спричиненого SEU в регістрах процесора [5].
4. Потрійне резервування: у критичних вузлах може застосовуватися архітектура з трьома паралельними обчислювачами та схемою голосування («majority voting»), що значно підвищує надійність, але втричі збільшує енергоспоживання.

Операційні системи реального часу (RTOS): FreeRTOS vs Zephyr

Вибір операційної системи є одним із найважливіших архітектурних рішень, оскільки на відміну від наземних IoT-пристроїв, де пріоритетом часто є швидкість розробки, у космічних місіях на перший план виходять детермінованість, надійність та мінімальні накладні витрати.

 ISSN 2071-1077(print)
ISSN 2663-2411(online

FreeRTOS: Еталон ефективності

FreeRTOS є найбільш поширеною операційною системою для вбудованих систем класу мікроконтролерів. Її архітектура базується на мінімалістичному мікроядрі, яке забезпечує лише базові примітиви: планувальник, мютекси, семафори та черги повідомлень.

Експериментальні дані, отримані на платформі Cortex-M7, демонструють виняткову продуктивність цієї системи, зокрема, операція перемикання контексту займає близько 223 машинних циклів [6]. Така швидкодія забезпечує мінімальну затримку реакції на зовнішні події, що є критичним для задач жорсткого реального часу (Hard Real-Time). Наприклад, це є критичним при керуванні двигунами маховиків або обробці сигналів GPS.

З точки зору ресурсоємності, FreeRTOS також демонструє високі показники. Так розмір ядра у бінарному вигляді становить приблизно 4-9 КБ, що дозволяє використовувати її на мікроконтролерах із вкрай обмеженою пам’яттю. Однак система має певні недоліки, ключовим з яких є відсутність чіткого розмежування прав доступу до пам’яті (MPU не використовується за замовчуванням). Це робить систему потенційно вразливою до помилок переповнення буфера.

Zephyr OS: Сучасний підхід до безпеки

Zephyr OS – це більш сучасна операційна система, що розвивається під егідою Linux Foundation і пропонує багату екосистему драйверів, підтримку мережевих протоколів та вбудовані механізми безпеки. Архітектурно Zephyr OS використовує модель єдиного адресного простору, проте забезпечує можливість захисту пам’яті для окремих потоків. Важливою особливістю є система конфігурації Kconfig, запозичена з Linux, яка дозволяє гнучко налаштовувати ядро під конкретні потреби місії.

Водночас, вищий рівень абстракції та функціональності впливає на продуктивність системи. Експериментальні дослідження показують, що перемикання контексту в Zephyr OS на аналогічній апаратній платформі займає 524 цикли [6], що у 2,3 рази більше, ніж у FreeRTOS. Латентність обробки переривань також вища і становить 143 цикли проти 101 у FreeRTOS.

Попри зниження «сирої» продуктивності, Zephyr OS забезпечує вищий рівень надійності та безпеки коду завдяки суворим стандартам розробки. Ця операційна система ідеально підходить для складних місій, де супутник виступає повноцінним вузлом мережі (концепція IoT in Space) і вимагає підтримки розвинених стеків протоколів, таких як IPv6, LwM2M та CoAP.

Результати порівняльного аналізу

Зведені характеристики продуктивності RTOS на основі даних UL Solution [6] наведено в таблиці 1.

Таблиця 1

Порівняльний аналіз показників продуктивності RTOS FreeRTOS та Zephyr OS

Характеристика	FreeRTOS	Zephyr OS	Інтерпретація для CubeSat
Перемикання контексту (цикли)	223	524	FreeRTOS краща для високочастотних контурів керування (ADCS)

 ISSN 2071-1077(print)
ISSN 2663-2411(online

Продовження таблиці 1

Характеристика	**FreeRTOS**	**Zephyr OS**	**Інтерпретація для CubeSat**
Латентність переривань (цикли)	101	143	FreeRTOS швидше реагує на події від датчиків
Синхронізація (Mutex Unblock)	1212	1309	Показники близькі, незначна перевага FreeRTOS
Повідомлення (Queue Block)	1660	710	Zephyr демонструє кращу оптимізацію роботи з чергами
Повідомлення (Queue Unblock)	1058	923	Zephyr ефективніший у передачі даних між задачами

Для більш детального аналізу архітектурного впливу механізмів безпеки на ресурси бортового комп’ютера, у таблиці 2 наведено порівняння конфігурацій RTOS із використанням модуля захисту пам’яті (MPU) та без нього.

Таблиця 2

Вплив механізмів захисту пам’яті на ресурсоємність операційних систем реального часу

Характеристика	**FreeRTOS (MPU Off)**	**FreeRTOS (MPU On)**	**Zephyr OS (Userspace Off)**	**Zephyr OS (Userspace On)**
Перемикання контексту (цикли)	223	~410	524	~1150
Обсяг Flash (ядро)	4–9 КБ	12–18 КБ	20–45 КБ	60–100 КБ
Обсяг RAM (мінімальний)	< 1 КБ	~4 КБ	~8 КБ	~16 КБ
Tick Rate (типовий)	100–1000 Гц	100–1000 Гц	100–10000 Гц	100–10000 Гц
Ізоляція задач	Відсутня	Програмна (MPU)	Рівень потоків	Повна (Userspace)

Висновок: для місій з обмеженим енергоспоживанням та критичними вимогами до часу реакції (наприклад, CubeSat 1U з активною орієнтацією) FreeRTOS залишається оптимальним вибором. Для складніших платформ (3U+), що виконують роль edge-вузлів та потребують розвинених мережевих функцій і захисту пам’яті, доцільним є перехід на Zephyr OS, попри її більшу ресурсоємність.

Математичне моделювання енергоспоживання

Енергія є найціннішим ресурсом на орбіті. Для оптимізації роботи ОВС необхідна точна модель, що враховує фізику напівпровідників у космосі.

Формалізація задачі енергоспоживання

Загальне енергоспоживання E_{total} за один орбітальний період T_{orbit} визначається як інтеграл від миттєвої потужності $P(t)$:

$$E_{total} = \int_0^{T_{orbit}} (P_{dyn}(t) + P_{static}(t))dt \quad (1)$$

Динамічна та Статична потужність

Динамічна потужність (P_{dyn}): Залежить від активності перемикання транзисторів при виконанні коду

$$P_{dyn} \approx C_{eff} \cdot V^2 \cdot f \cdot \alpha, \quad (2)$$

де C_{eff} – ефективна ємність, V – напруга живлення ядра (наприклад, 1.2 В для STM32H7), f – тактова частота (до 480 МГц),

α – коефіцієнт активності.

Для сучасних контролерів (STM32H7, ATSAMV71) динамічне споживання складає приблизно 250–300 мА/МГц у режимі повного навантаження [3].

Статична потужність (P_{static}) зумовлена струмами витоку (I_{leak}), які протікають навіть коли процесор простоює. У нанометрових техпроцесах (40 нм і менше) ця складова стає значною. Критичним фактором для космосу є експоненційна залежність струму витоку від температури T (наближення Шоклі):

$$I_{leak}(T) \propto T^2 \cdot e^{\frac{-E_g}{k \cdot T}}, \quad (3)$$

де E_g – ширина забороненої зони, k – стала Больцмана.

Дослідження показують, що при нагріванні кристала з 25°C до 85°C (типовий діапазон на сонячній стороні орбіти) струм витоку може зростати у рази. Це створює ризик «теплового розгону» (thermal runaway), коли нагрів збільшує струм витоку, що, в свою чергу, ще більше нагріває кристал. Крім того, накопичення радіаційної дози (TID) призводить до утворення дефектів в оксиді затвора, що також збільшує струми витоку з часом.

Практичне значення: Модель показує, що просте зниження тактової частоти може бути неефективним, якщо температура кристала висока. Системи терморегуляції та планувальник задач RTOS повинні враховувати поточну температуру для вибору оптимального режиму роботи.

Архітектура «Edge-to-Cloud»: Гібридна модель обчислень

Оскільки фізичні можливості нарощування обчислювальної потужності на борту обмежені SWaP, тож подальший розвиток функціональності можливий лише через зміну архітектурної парадигми на користь розподілених обчислень.

Концепція Computation Offloading та Енергетичний Критерій

Концепція передбачає, що супутник виступає як edge-вузол, який може

динамічно вирішувати: обробити дані локально чи передати їх у наземну хмару. Рішення приймається на основі мінімізації вартості (енергії або часу).

Умова доцільності офлоудингу:

$$E_{tx} < E_{local} \quad (4)$$

де E_{tx} – енергія на передачу даних, E_{local} – енергія на локальну обробку.

Розглянемо задачу класифікації зображень (наприклад, визначення наявності хмар на знімку перед його збереженням).

Локальна обробка (On-board):

- Вхідні дані: Знімок 5 МБ.
- Алгоритм: MobileNetV2.
- Час виконання на Cortex-M7: 60 с
- Споживання CPU: 300 мВт
- $E_{local} = 60\ \text{с} \times 0.3\ \text{Вт} = 18\ \text{Дж}$

Офлоудинг (Передача на Землю):

- Канал зв’язку: S-band
- Ефективна швидкість 1 Мбіт/с
- Час передачі: $5\ \text{МБ} \times 8\ \text{біт} / 1\ \text{Мбіт/с} = 40\ \text{с}$
- Споживання передавача: 2 Вт.
- $E_{tx} = 40\ \text{с} \times 2\ \text{Вт} = 80\ \text{Дж}$

Аналіз. У даному сценарії $E_{local}(18\ \text{Дж}) < E_{tx}(80\ \text{Дж})$, отже, вигідніше обробити дані на борту. Це типова ситуація для задач, де обсяг вхідних даних великий, а алгоритм відносно ефективний. Однак, якщо передаються лише метадані, тоді енергія передачі стає мізерною, що робить офлоудинг доцільним для подальшої складної аналітики. Запропонована модель може бути використана як основа для задач динамічного планування навантаження RTOS та прийняття рішень щодо offloading.

Еволюція наземного сегмента: Федеративні GSaaS та SkyGS

Традиційна модель передбачала наявність власних наземних станцій. Впровадження сервісів *Ground Station as a Service* (GSaaS), таких як AWS Ground Station та Azure Orbital [7], дозволило замінити капітальні витрати (CAPEX) на операційні (OPEX). Використання одного провайдера залишається обмеженим через погодні умови та географічне покриття.

Архітектура SkyGS (досліджена у 2024-2025 роках) забезпечує федерацію ресурсів різних провайдерів GSaaS. Алгоритми планування на основі методу Ляпунова дозволяють динамічно резервувати вікна зв’язку з оптимальною станцією будь-якого провайдера.

Дослідження показують, що застосування SkyGS зменшує середню затримку передачі даних на 95% та скорочує операційні витрати на 63% порівняно з використанням одного провайдера [8]. Підвищена ефективність досягається завдяки щільнішій мережі контактів, що дозволяє супутнику надсилати дані без очікування прольоту над конкретною точкою.

Ефективність SkyGS значною мірою визначається політикою доступу провайдерів та регуляторними обмеженнями частотного ресурсу.

Software-Defined Satellite та мікросервіси

Для забезпечення ефективної взаємодії з хмарним середовищем бортове програмне забезпечення супутника повинно відповідати критеріям гнучкості та модульності. Враховуючи, що класичні інструменти контейнеризації, такі як Docker, є занадто ресурсоємними для мікроконтролерів, доцільним є використання легковагових середовищ виконання.

Зокрема, експериментальні дослідження підтверджують ефективність використання інтерпретатора WebAssembly (WASM3) [9] на борту наносупутника.

WebAssembly (WASM): Високопродуктивна «пісочниця»

WASM – це компактний бінарний формат інструкцій. Інтерпретатори, такі як WASM3, дозволяють запускати скомпільовані модулі (написані на Rust, C++, Go, тощо) у безпечному ізольованому середовищі («пісочниці») на борту MCU [9]. Переваги:

- ізоляція: збій у модулі (наприклад, ділення на нуль) не призводить до краху ядра RTOS.
- компактність: оновлення (новий алгоритм) може займати лише кілька кілобайт, що економить трафік.
- безпека: модуль не має прямого доступу до пам’яті системи, лише до дозволених ресурсів.

Це дозволяє реалізувати концепцію «store apps, not firmware», тобто супутник отримує нові функції як додатки, без ризикованого процесу перепрошивки всього образу системи.

Порівняння зі скриптовими мовами (MicroPython, Lua)

MicroPython та Lua популярні завдяки простоті розробки, але мають суттєві недоліки для критичних застосувань.

Так MicroPython використовує значний обсяг RAM, а головною проблемою є збірка сміття (Garbage Collection), яка може викликати недетерміновані затримки, що неприпустимо для систем реального часу. Також присутня схильність до фрагментації кучі (heap fragmentation).

В той самий час Lua є більш компактним та швидшим за Python, на додаток ще й легко інтегрується з C. Проте стандартний інтерпретатор все ще значно повільніший за нативний код, а JIT-компіляція (LuaJIT) на мікроконтролерах зазвичай неможлива через обмеження пам’яті та захист від виконання коду в RAM.

Бенчмаркінг віртуальних середовищ

На основі аналізу наукових джерел [10] складено таблицю 3 порівняння ефективності цих технологій на вбудованих системах.

Висновок. WebAssembly (у реалізації Wasm3) є найбільш збалансованим підходом серед існуючих легковагових середовищ виконання лідером для архітектури Software-Defined Satellite. Він забезпечує продуктивність, достатню для запуску легковагових алгоритмів, одночасно гарантуючи безпеку пам’яті, яку не можуть забезпечити ні нативний C (ризик помилок), ні MicroPython (через ненадійність управління пам’яттю та низьку швидкість виконання).

ISSN 2071-1077(print)
ISSN 2663-2411(online

Таблиця 3

Порівняльна характеристика ефективності технологій віртуалізації та скриптових середовищ для наносупутників

Характе-ристика	**WebAssembly (Wasm3)**	**Lua (Standard/eLua)**	**MicroPython**	**Інтерпретація для наносупутників**
Швидкість виконання (нормалізована до C=1.0)	0.1 – 0.3x (повільніше в 3-10 разів)	~0.02 – 0.1x (повільніше в 10-50 разів)	~0.01 – 0.05x (повільніше в 20-100 разів)	Критично: WASM на порядок швидший за скриптові мови, що робить його єдиним варіантом для ОВР (стиснення, обробка сигналів).
Розмір інтерпрета-тора	~64 КБ	~100 – 200 КБ	~256 КБ+	Wasm3 є найкомпактнішим, залишаючи більше місця для корисного навантаження.
Вимоги до RAM (Min)	~10 – 64 КБ	~64 КБ	~16 КБ (мінімум), але високий ризик фрагментації	WASM має передбачуване споживання пам’яті завдяки статичному виділенню (linear memory), на відміну від динамічної кучі Python.
Механізм оновлення	Бінарний модуль (.wasm). Компактний.	Текстовий скрипт або байт-код.	Текстовий скрипт (.py) або байт-код (.mpy).	.wasm файли зазвичай менші за вихідний код складних програм, що економить трафік.
Безпека та Ізоляція	Висока. Математично гарантована ізоляція пам’яті.	Слабка. Прямий доступ до C API може порушити стабільність.	Слабка. Розрахована на довірений код.	WASM дозволяє безпечно запускати код сторонніх розробників (наприклад, студентські експерименти) без ризику для ядра супутника.
Детермінованість	Висока. Відсутність GC (залежить від мови джерела, для C/Rust/Zig – відсутня).	Середня. Інкрементальний GC.	Низька. GC викликає непередбачувані паузи.	MicroPython непридатний для задач керування в реальному часі.

Висновки

Проведений комплексний аналіз дозволяє сформулювати наступні узагальнення та рекомендації для проектування архітектури наносупутників в епоху «NewSpace»:

1. Апаратна стратегія. Для забезпечення балансу між продуктивністю та надійністю оптимальним є різнорідний підхід: для критичних підсистем (OBC, ADCS) слід використовувати радіаційно-стійкі контролери з апаратним TMR, такі як Vorago VA41630, що усувають потребу у ресурсоємному програмному скрабінгу. Для задач обробки даних доцільно використовувати потужні COTS-процесори (Cortex-M7, RISC-V) із програмним захистом.

2. Вибір RTOS. Вибір операційної системи реального часу визначається специфікою виконуваних місій. FreeRTOS залишається оптимальним рішенням для енергодефіцитних платформ формату 1U, забезпечуючи мінімальні накладні витрати на перемикання контексту (223 цикли). Для більш складних платформ, інтегрованих у мережеві середовища, Zephyr OS забезпечує необхідний рівень безпеки та підтримку розширеної мережевої функціональності, при цьому відзначається лише помірне зниження продуктивності.

3. Гібридна архітектура. Використання федеративних мереж SkyGS дозволяє подолати обмеження каналу зв’язку, знижуючи затримки на 95%. Рішення про офлоудинг обчислень має прийматися динамічно на борту на основі розробленого енергетичного критерію.

4. Гнучкість та Безпека. Впровадження технологій віртуалізації (WebAssembly) дозволяє перетворити супутник на програмно-визначену платформу, здатну до безпечної модернізації функцій протягом усього терміну експлуатації.

Імплементація запропонованих підходів дозволить створити нове покоління «розумних» наносупутників, які будуть не просто збирачами даних, а інтегрованими вузлами глобальної обчислювальної інфраструктури.

References

1. Fortune Business Insights. CubeSat Market Size, Share & Growth Report, 2032. URL: https://www.fortunebusinessinsights.com/cubesat-market-113707.

2. Zhang, Y. Edge-Cloud Collaborative Satellite Image Analysis for Efficient Man-Made Structure Recognition / Y. Zhang [et al.]. – arXiv preprint arXiv:2410.05665. – 2024.

3. STM32H753ZI High-performance and DSP with FPU Arm Cortex-M7 MCU [Electronic resource] / STMicroelectronics. – 2025. – URL: https://www.st.com/en/microcontrollers-microprocessors/stm32h753zi.html

4. Liubimov, O. UAV Mission Computer Operation Mode Optimization Focusing on Computational Energy Efficiency and System Responsiveness / O. Liubimov, I. Turkin, V. Cheranovskiy, L. Volobuieva // Computation. – 2024. – Vol. 12, no. 12. – P. 235. – DOI: 10.3390/computation12120235.

5. Manzhos, Y. SITL-Based Formal Verification of Cyber-Physical Systems Software: Reliability Model, Method and Implementation / Y. Manzhos, Y. Sokolova, V. Kharchenko, S. Semenov // Preprints. – 2025. – DOI: 10.20944/preprints202503.0856.v1.

6. Measuring Real-Time Operating System Performance – Part II: Comparing FreeRTOS vs. Zephyr / UL Solutions. – 2021. URL:

https://www.ul.com/sis/blog/measuring-real-time-operating-system-performance-part-ii-comparing-freertos-vs-zephyr

7. Meyrick, E. Ground Station as a Service: A Space Cybersecurity Analysis / E. Meyrick [et al.] // 72nd International Astronautical Congress (IAC), Dubai. – 2021.

8. Zhu, A. Y. The Space above the Sky: Uniting Global-Scale Ground Station as a Service for Efficient Orbital Data Processing / A. Y. Zhu. – arXiv preprint arXiv:2501.00354. – 2025. – DOI: 10.48550/arXiv.2501.00354.

9. Liubimov, O. Use of micro-services architecture and containerization for the fast development and testing of the CubeSat nanosatellites software / O. Liubimov // Journal of Rocket-Space Technology. – 2023. – Vol. 31, no. 4. – P. 128–137. – DOI: 10.15421/452317.

10. Peach, G. eWASM: Practical Software Fault Isolation for Reliable Embedded Devices / G. Peach, R. Pan, Z. Wu, G. Parmer. – GW Engineering : The George Washington University, 2025. – URL: https://www2.seas.gwu.edu/~gparmer/publications/emsoft20wasm.pdf

Надійшла до редакції 17.12.2025, розглянута на редколегії 30.01.2026

Analysis of Architectural Methods for Integrating Nanosatellite Onboard Systems into Cloud Computing Environments

This paper presents a comprehensive analysis of architectural methods for optimizing computational resources of CubeSat standard nanosatellites within the transition to the NewSpace paradigm. The fundamental problem of imbalance between exponentially growing onboard computing requirements and severe hardware constraints of 1U-3U platform formats is examined, characterized by critical power deficit, high radiation vulnerability of components, and limited memory capacity. A detailed comparative efficiency analysis of leading real-time operating systems FreeRTOS and Zephyr OS was conducted based on context switching latency, energy efficiency, and reliability criteria, demonstrating FreeRTOS advantages for power-constrained missions due to minimal kernel overhead at 223 machine cycles. A mathematical model for the onboard computer's power consumption has been formalized, accounting for dynamic and static power components relative to the space thermal environment and the impact of radiation on leakage currents in semiconductor structures. The prospects of implementing a hybrid Edge-to-Cloud architecture are investigated, involving intelligent offloading of resource-intensive computational tasks to ground-based cloud environments via GSaaS (Ground Station as a Service), with detailed analysis of energy and communication feasibility criteria for offloading. The necessity of using software-hardware methods for ensuring radiation tolerance is substantiated, including EDAC memory scrubbing, and multi-level watchdog timer systems consuming up to 20% of processor resources. The feasibility of lightweight WebAssembly-based containerization technologies is demonstrated by recent experimental studies, confirming their applicability to software-defined satellite architectures with secure in-orbit updates and fault isolation.

Keywords: CubeSat; real-time operating systems; computation offloading; edge computing; cloud computing; energy efficiency

 ISSN 2071-1077(print)
ISSN 2663-2411(online

Відомості про авторів:

Лимаренко Олексій Борисович – аспірант, Національний аерокосмічний університет «Харківський авіаційний інститут», м. Харків, Україна, email: oleksii.lymarenko@gmail.com, ORCID: 0009-0000-9996-2901.

Соколова Євгенія Віталіївна – канд. техн. наук, доц., доц. каф. інженерії програмного забезпечення, Національний аерокосмічний університет «Харківський авіаційний інститут», Харків, Україна, e-mail: y.sokolova@khai.edu, ORCID: 0000-0002-1497-4987.

About the authors:

Oleksii LYMARENKO – Ph.D student, National Aerospace University «Kharkiv Aviation Institute», Ukraine, email: oleksii.lymarenko@gmail.com, ORCID: 0009-0000-9996-2901.

Yevheniia SOKOLOVA – Ph.D in Information Technologies, Associate Professor at the Department of Software Engineering and Business, National Aerospace University «Kharkiv Aviation Institute», Kharkiv, Ukraine, e-mail: y.sokolova@khai.edu, ORCID: 0000-0002-1497-4987.