DOI: 10.15587/2312-8372.2017.117889

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ ЗА РАХУНОК РОЗПОДІЛУ ЗАВДАНЬ МІЖ ЯДРАМИ ОБЧИСЛЮВАЛЬНОГО ПРОЦЕСОРА

Журавська І. М., Боровльова С. Ю., Костиря М. А., Корецька О. О.

1. Вступ

Обчислювальні системи більшості безпілотних літальних апаратів (БПЛА) базуються на однокристальних 4-ядерних 32-бітних процесорах ARM Cortex, Qualcomm Snapdragon та ін. [1, 2]. Кількість обчислювальних потоків в них дорівнює кількості фізичних ядер, між котрими можуть бути розподілені завдання, що виконуються (орієнтація у просторі, керування рухом, отримання та обробка показників з датчиків, тощо).

Для комп'ютерної системи (КС) з обмеженими ресурсами (обчислювальними або енергоживлення) виникає наступна проблема – робота автоматичного планувальника (scheduler) потоків операційної системи (ОС) призводить до невиправданого навантаження процесора. Але для мобільних комп'ютерів, на яких базується конструкція БПЛА, частіше є важливим економічність в енергоспоживанні (на практиці – час роботи від акумуляторної батареї), ніж залучення в обчислювальному процесі усіх наявних ядер в однокристальному обчислювальному процесорі (CPU).

Тому актуальною задачею є підвищення ефективності використання БПЛА за рахунок розробки кращих, ніж автоматичні, алгоритмів роботи багатоядерного процесора, на базі якого побудована обчислювальна система БПЛА.

2. Об`єкт дослідження та його технологічний аудит

Об'єктом дослідження є обчислювальна система БПЛА. Більшість сучасних БПЛА базуються на 2- або 4-ядерних однокристальних процесорах, між котрими автопланувальник ОС намагається рівномірно розподілити обчислювальні завдання. В той же час, перше ядро може миттєво стати надзвичайно перевантаженим у разі виникнення термінового завдання від системи управління БПЛА. Тому *предметом дослідження* є комплексні показники стану ядер процесора при різних алгоритмах розподілу завдань між ядрами багатоядерного однокристального процесору.

Зважаючи на те, що безпосереднє натурне моделювання на БПЛА може привести до критичного застосування останнього, було вирішено провести імітаційне моделювання диспетчеризації завдань на стаціонарній КС з 4-ядерним одноплатним процесором.

Тест проводився на КС з CPU Intel Core i7-4790 3.60GHz при вимкненій функції підтримки процесором технології Hyper-Threading [3]. Для експерименту була створена тестова програма, яка на кожній ітерації сортує масив з 10000 елементів.

В межах даної роботи було прийнято рішення під час написання програмного забезпечення на C# самостійно реалізувати навантаження ядер заданими потоками. Також планувалось здійснити процес еталонного зіставлення з авто-

матичною диспетчеризацією в ОС Windows. Інакше цей процес можна визначити як бенчмаркінг ефективності використання ядер багатоядерного процесора.

Досліджувались такі критерії, як таймінг завдань, енергоспоживання процесора, температура та відсоткове навантаження кожного ядра та всього процесора протягом виконання кожного з тестових завдань чотирьох типів. Таймінг оцінювався за результатами роботи власноруч написаного програмного забезпечення. Температура (або тепловиділення – параметр TDP, Thermal Design Power), енергоспоживання та відсоткове використання процесора можуть бути оцінені за допомогою попередньо запущеної моніторингової утиліти.

Одним з найбільш проблемних місць є перевантаження та перегрів процесору обчислювальної системи БПЛА. Це може привести до зависання КС, втрати керування та крешу дрону. Крім того, у критичній ситуації перевантажений процесор не забезпечить терміново необхідні обчислення зміни курсу БПЛА або перехід на автономний режим польоту.

3. Мета та задачі дослідження

Мета дослідження – підвищити ефективність використання БПЛА з подовженням часу польоту останніх за рахунок зниження перенавантаження та енергоспоживання обчислювальної системи.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Проаналізувати можливість диспетчеризації завдань процесору БПЛА з багатопотоковою організацією розподілу завдань між ядрами CPU та розробити відповідні алгоритми.
2. Виконати імітаційне моделювання роботи розроблених алгоритмів для 4-ядерного CPU.
3. Дослідити технічні показники стану CPU КС БПЛА (температуру у розрізі по ядрах, енергоспоживання та навантаженість процесору, кількість обертів кулера для забезпечення необхідного тепловідводу тощо) при реалізації різних алгоритмів диспетчеризації.
4. Дослідити можливість підвищення ефективності використання БПЛА за рахунок джерел додаткового енергоспоживання.
5. Визначити з досліджених оптимальний алгоритм розподілу завдань між ядрами багатоядерного однокристального процесору.

4. Дослідження існуючих рішень проблеми

Проблема оптимізації енергоспоживання, навантаження та запобігання перегріву комп’ютерних компонентів обчислювальних систем БПЛА широко розглянута у ресурсах світової наукової періодики. Основним направленням її вирішення є багатопотокова диспетчеризація завдань між ядрами багатоядерного процесора.

Однак, існуючі алгоритми планування розподілу завдань на рівні ОС [4, 5] неспроможні вирішувати проблему запобігання максимальному навантаженню процесора та кожного з його ядер майже до 100 % [6]. Крім того, можлива ситуація, коли завдання будуть стоять у черзі до процесора [7], що неприпустимо для завдань оптичної навігації та керування БПЛА.

Для зменшення енергоспоживання в системах, що живляться від батареї, іноді використовується механізм відключення модулів зв'язку [8] під час швидкісного декодування даних або під час компресії отриманого відео [9, 10]. Але, у такому випадку може мати місце так званий ефект Дала, коли частина ядер процесора простоює, незважаючи на те, що своєчасність виконання деяких завдань не забезпечується [11]. Таке планування завдань є неприпустимим для БПЛА, тому що він, наприклад, не зможе прийняти керуючі сигнали від головного БПЛА або з наземного командного центра.

Втрати енергії на постійне підключення до батареї живлення модулів зв'язку можуть бути частково компенсовані за допомогою механізмів, описаних у [12]. Для вибору типу фізичного впливу на первинний перетворювач були проаналізовані десять найбільш поширених впливів та ефектів, які вони викликають: ефект Зеєбека, піроелектричний ефект тощо [13]. В результаті аналізу виявлено, що п'єзоелектричний ефект є найбільш енергетично привабливим для вирішення поставленої задачі побудови ефективних інформаційно-вимірювальних систем (ІВС), до яких можна віднести й БПЛА. Тобто, це означає, що для живлення компонентів такої ІВС від енергії вимірювального сигналу найкраще підходять п'єзоелектричні датчики. П'єзоелектричні перетворювачі дозволяють вирішувати різноманітні завдання: вимірювання механічних параметрів (тиску, прискорення, маси, кутових швидкостей, моментів, деформацій тощо).

Важливе значення також має вибір технології бездротового типу передачі вимірювальної інформації від безпровідного пристрою передачі інформації (БППІ) до система збору інформації (СЗІ). В результаті аналізу були обрані такі перспективні для БПЛА специфікації мережевих протоколів, як ZigBee та LoRa (в залежності від типу поставленої задачі) [14].

Оцінку ефективності вжитих заходів щодо підвищення ефективності використання компонентів КС можна провести за допомогою низки програмних продуктів для моніторингу та зміни режимів навантаження процесорів [15]. Однак, слід зауважити, що для КС БПЛА категорично не можна використовувати так звані «прожарочні тести» (AIDA64, LinX, IntelBurnTest, Prime95, OCCT Perestroika тощо). Принципом їх дії є стрес-тестування, коли процесор виконує спеціалізовані інтенсивні обчислення, що імітують сценарій роботи в найбільш важких умовах та призводять до максимального нагрівання ядер [16]. Застосування таких тестів може спричинити невиправні наслідки з вигоранням окремих компонентів КС. При введенні граничних значень в тестові режими таких утиліт можливо вивести досліджувану КС з ладу.

Тому для бенчмаркінгу доцільно застосувати програмне забезпечення (ПЗ), яке серед інших показників стану апаратного забезпечення КС виводить на монітор показання вбудованих в таку КС датчиків (Core Temp, CPU-Z, HWMonitor, SpeedFan й т. п.).

З перелічених для бенчмаркінгу була обрана утиліти HWMonitor v.1.33.0.

Таким чином, результати аналізу дозволяють зробити висновок про те, що питання підвищення ефективності використання БПЛА потребують додаткових досліджень.

5. Методи дослідження

Отже, було прийняте рішення спрямувати зусилля на розробку методів та алгоритмів диспетчеризації завдань на різних комп'ютерних компонентах

БПЛА. По-перше, детальніше розглянемо проблему оптимального навантаження CPU у розрізі ядер процесора.

Прив'язка процесу до заданих ядер не викликає ніяких труднощів, потрібно лише скористатися властивістю *ProcessorAffinity* класу *Process*, прописаним в просторі імен *System.Diagnostics* [17]. Залежно від бітової маски, яка присвоюється цій властивості, можна отримати всі комбінації ядер на вибраному пристрої. При наявності 4 ядер, планувальник потоків ОС автоматично буде використовувати їх всі, намагаючись отримати максимальну вигоду від багатоядерності процесора (надалі – CPU). В такому випадку *ProcessorAffinity* дорівнює 15 та визначається за формулою:

$$ProcessorAffinity = 2^n - 1, \quad (1)$$

де n – кількість ядер багатоядерного процесора, у тесті n=4.

Слід зазначити, що при розробці програмних додатків рекомендується уникати використання для обчислень ядра, на якому операційна система виконує передачу даних, очищення пам'яті та інші системні процеси. Зазвичай всі такі процеси обмежені першим ядром процесора. Таким чином, виключення першого ядра з обчислювальної задачі може привести до підвищення продуктивності додатків.

Тому обираємо для бенчмаркінгу ядра з 2-го по 4-те та присвоюємо нову бітову маску, отримавши значення 14 (рис. 1).

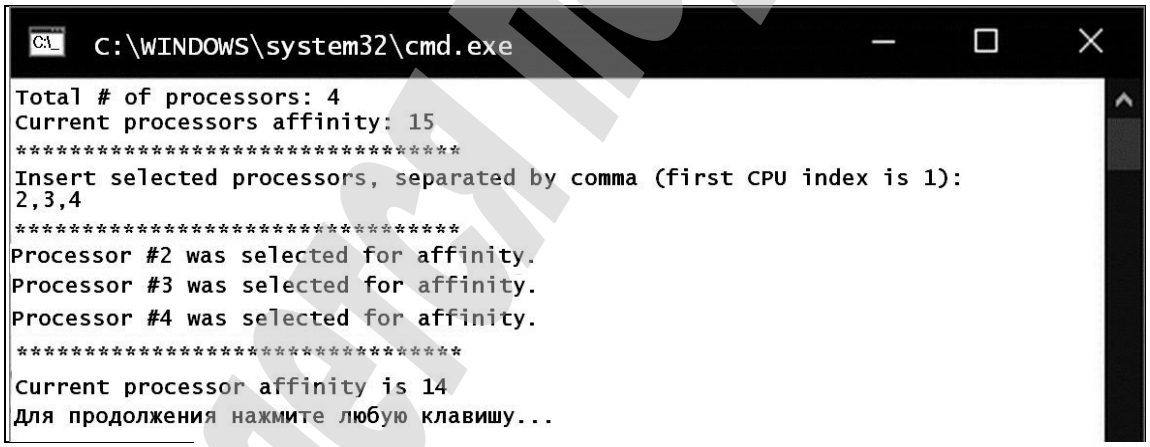


Рис. 1. Значення властивості *ProcessorAffinity* при використанні 4 ядер та 2–4-го ядер

Відповідно, для кожного з чотирьох ядер чотириядерного процесора значення *ProcessorAffinity* будуть такими:

1) для 1-го ядра *ProcessorAffinity*=1;
2) для 2-го ядра *ProcessorAffinity*=2;
3) для 3-го ядра *ProcessorAffinity*=4;
4) для 4-го ядра *ProcessorAffinity*=8.

Треба зауважити, що більш корисним буде прив'язуватись не до ядер процесорів, а використовувати прив'язку до потоків, але цей шлях містить багато труднощів. Справа в тому, що Intel намагається полегшити роботу програміста, взявши прив'язку потоків до ядер на себе [3]. Очевидно, це має сенс, бо біль-

шість великих компаній розробляють продукцію для масового користувача, для якого розуміння і використання власної паралелізації не є необхідним.

Для високої математичної продуктивності можна використовувати бібліотеку готових компонентів для програмування додатків на мовах C/C++ та Fortran – Intel Math Kernel Library (MKL) [18]. Для швидкої роботи циклу доцільно включити автопаралелізацію в компіляторі від фірми Інтел – Intel C++ Compiler (ICC) [3]. Для максимального використання кешу включають підтримку технології Hyper-Threading. Хоч наявність такої технології є хорошим маркетинговим рішенням, однак, для тих, кому все-таки потрібно визначати паралелізацію вручну, це ставить на шляху певні перешкоди. Програміст може встановити власну прив'язку потоків до ядер, однак бібліотека Intel зведе їх нанівець.

У досліджуємому випадку додаткові труднощі виникають через те, що потік .NET, яким керує загальномовне виконуюче середовище Common Language Runtime (CLR), не відповідає потоку ОС, а прив'язувати до ядер можна виключно потоки операційної системи [19, 20]. Щоб вирішити цю проблему, можна скористатися наведеними методами класу *Thread*:

```
Thread.BeginThreadAffinity();
...
Thread.EndThreadAffinity().
```

В такому разі код між наведеними викликами виконується на єдиному потоці ОС, що суттєво послаблює менеджмент потоків CLR.

Тепер можна перейти до самої прив'язки. Отримати ОС-потоки додатку .NET можна, використовуючи *Process.GetCurrentProcess().Threads* – колекцію потокових об'єктів. Щоб отримати ОС-потік, який виконується в даний момент, потрібно використати наступний код:

```
[DllImport("kernel32.dll")]
public static extern int GetCurrentThreadId().
```

Використовуючи повернутий ID, можна знайти потік завдання, що виконується, і використати властивість *ProcessThread.ProcessorAffinity*, робота якої дуже схожа на *Process.ProcessorAffinity*, яка описана вище.

Далі оголошено клас *DistributedThread*, використання якого дозволяє запускати потоки на вибраних ядрах. Суть полягає в інкапсуляції звичайного Thread-об'єкта, обмежуючи його діючим потоком ОС і установкою бажаної прив'язки до ядра:

```
using System;
using System.Diagnostics;
using System.Linq;
using System.Runtime.InteropServices;
using System.Threading;
namespace DistributedWorkManager{
    public class DistributedThread{
     [DllImport("kernel32.dll")]
```

```
        public static extern int GetCurrentThreadId();
        [DllImport("kernel32.dll")]
        public static extern int GetCurrentProcessorNumber();
        private ThreadStart threadStart;
        private Thread thread;
        public int ProcessorAffinity { get; set; }
        public Thread ManagedThread{
            get{
                return thread;
            }
        }
        private DistributedThread(){
            thread = new Thread(DistributedThreadStart);
        }
        public DistributedThread(ThreadStart threadStart) : this(){
            this.threadStart = threadStart;
        }
        public void Start(){
            if (this.threadStart == null) throw new InvalidOperationException();
            thread.Start(null);
        }
        private void DistributedThreadStart(object parameter){
            try{
                Thread.BeginThreadAffinity();
                if (ProcessorAffinity != 0){
                    CurrentThread.ProcessorAffinity = new IntPtr(ProcessorAffinity);
                }
                if (this.threadStart != null){
                    this.threadStart();
                }
                else{
                    throw new InvalidOperationException();
                }
            }
            finally{
                CurrentThread.ProcessorAffinity = new IntPtr(0xFFFF);
                Thread.EndThreadAffinity();
            }
        }
        private ProcessThread CurrentThread{
            get{
                int id = GetCurrentThreadId();
                return (from ProcessThread th in Process.GetCurrentProcess().Threads
                    where th.Id == id select th).Single();
            }
        }
    }
}
```

Тепер розроблений клас *DistributedThread* можна використовувати в наступних проектах.

Для визначення оптимального алгоритму розподілення завдань між ядрами було розроблено чотири проекти, перший з яких (Проект № 1) імітує роботу автопланувальника ОС Windows, який намагається рівномірно розподілити завдання між апаратно наявними 4 ядрами одного процесора. Цей Проект № 1 є бенчмарком (еталонним тестом) для співвідносної оцінки інших проектів за критеріями таймінгу (с), енергоспоживання (Вт), температури (ºС) та відсоткового навантаження процесору з урахуванням фонових завдань операційної системи.

Кожен з проектів виконувався з дотриманням таких умов:

1. Підтримка технології Hyper-Threading відключена.
2. Утиліта HWMonitor активна.
3. Проекти запускаються почергово при однаковій кількості фонових задач.

Для оптимізації енергоспоживання БПЛА необхідно зосередити увагу на функціональному складі компонентів БПЛА. Оскільки на борту БПЛА встановлено достатньо велику кількість датчиків, БПЛА можна класифікувати як інформаційно-вимірювальну систему (ІВС). Тому, доцільно розглянути можливість додаткового живлення компонентів БПЛА від енергії вимірювального сигналу.

Основними етапами під час вирішення проблеми додаткового живлення БПЛА є (рис. 2):

– вибір типу фізичного впливу на первинний перетворювач, а також датчика (Д), робота якого заснована на такому типі перетворювання сигналу;

– вибір параметрів первинного перетворювача, здатного задовольнити як інформаційні, так і енергетичні потреби інформаційно-вимірювального пристрою (ІВП);

– вибір елемента накопичення енергії – пристрою обробки інформації (ПОІ);

– вибір стандарту передачі результатів вимірювання від БППІ до СЗІ.

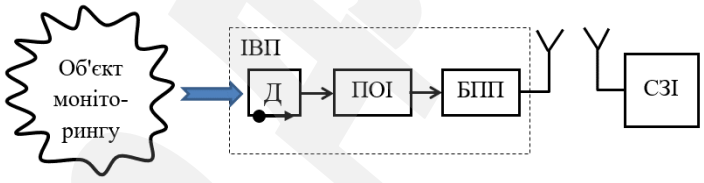


Рис. 2. Складові інформаційно-вимірювальної системи

Для дослідження основних процесів при забезпеченні безперервної роботи ІВС при живленні компонентів від енергії вимірювального сигналу була побудована модель взаємодії вузлів схеми з урахуванням часових співвідношень (рис. 3).

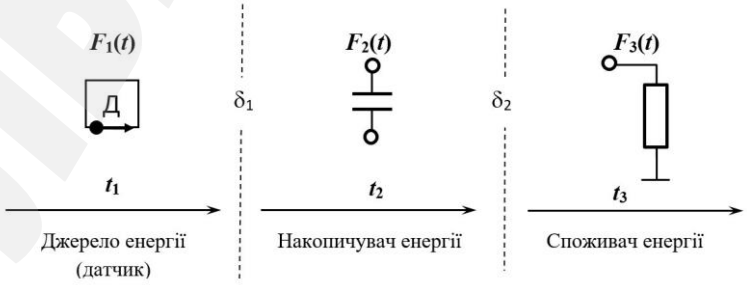


Рис. 3. Модель взаємодії вузлів інформаційно-вимірювальної системи

Енергія, яку отримує датчик від енергії вимірювального сигналу (перша ланка), накопичується в елементі накопичення енергії (друга ланка). Функція залежності енергії від часу (втрат, накопичення, споживання) має вигляд $F_1(t)$. В залежності від обраної схеми під'єднання елемента накопичення, відбуваються втрати при передачі енергії з коефіцієнтом передачі δ_1. В самій схемі накопичення з часом енергія також втрачається відповідно до закону $F_2(t)$. Споживання навантаження відбувається згідно функції $F_3(t)$ (третя ланка). При цьому, коефіцієнт передачі δ_2 описує втрати при під'єднанні навантаження до схеми накопичення енергії.

Якщо виконується залежність між описаними змінними:

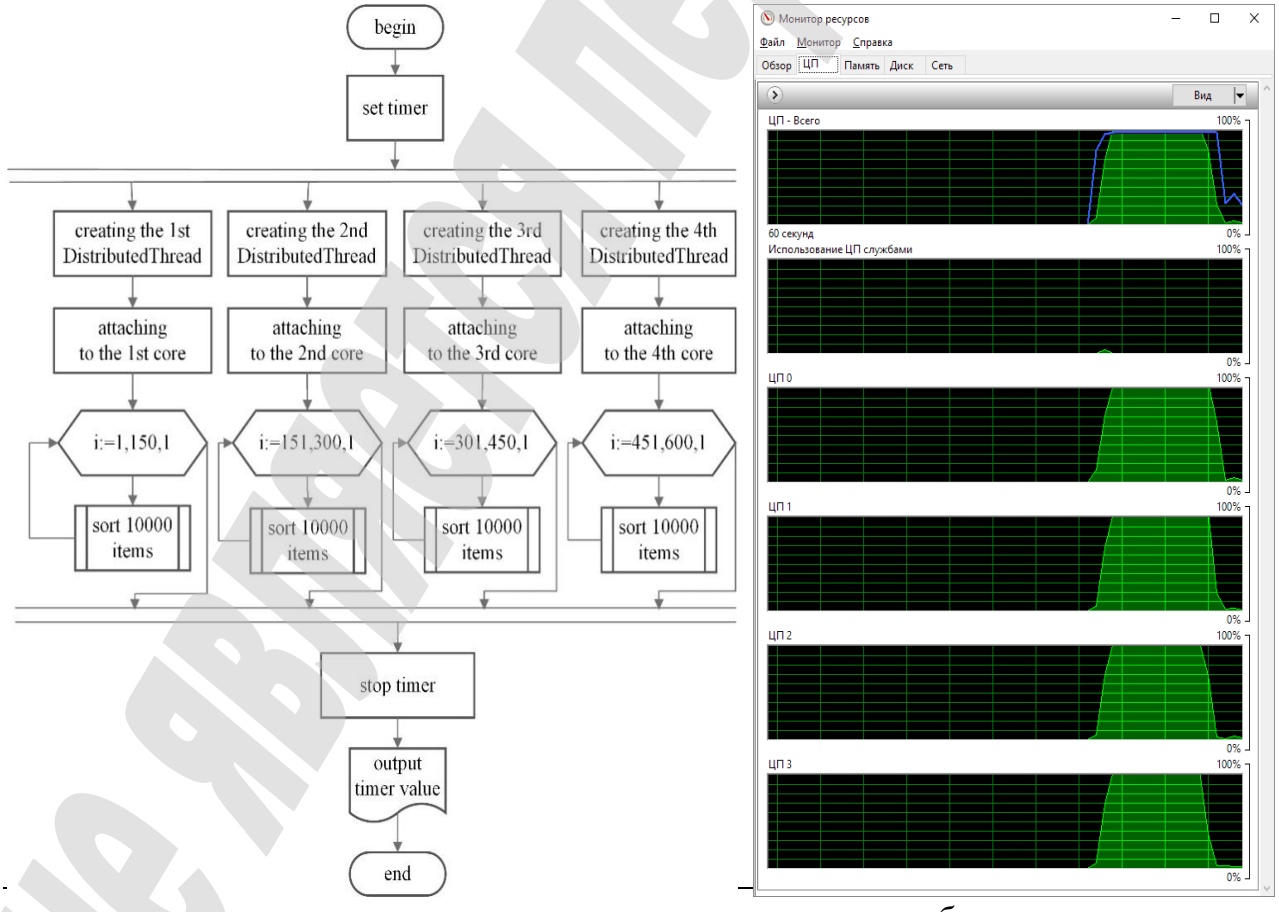
$$F_1(t)>\delta_1 F_2(t)+\delta_2 F_3(t), \qquad (2)$$

у такому разі буде забезпечена безперервна робота ІВС. Тобто, додавання енергії до системи буде більше, ніж вона витрачається на всіх вузлах схеми.

6. Результати дослідження

Зміст проектів та результати їх виконання наведені на рис. 4–7 та зведені у табл. 1.

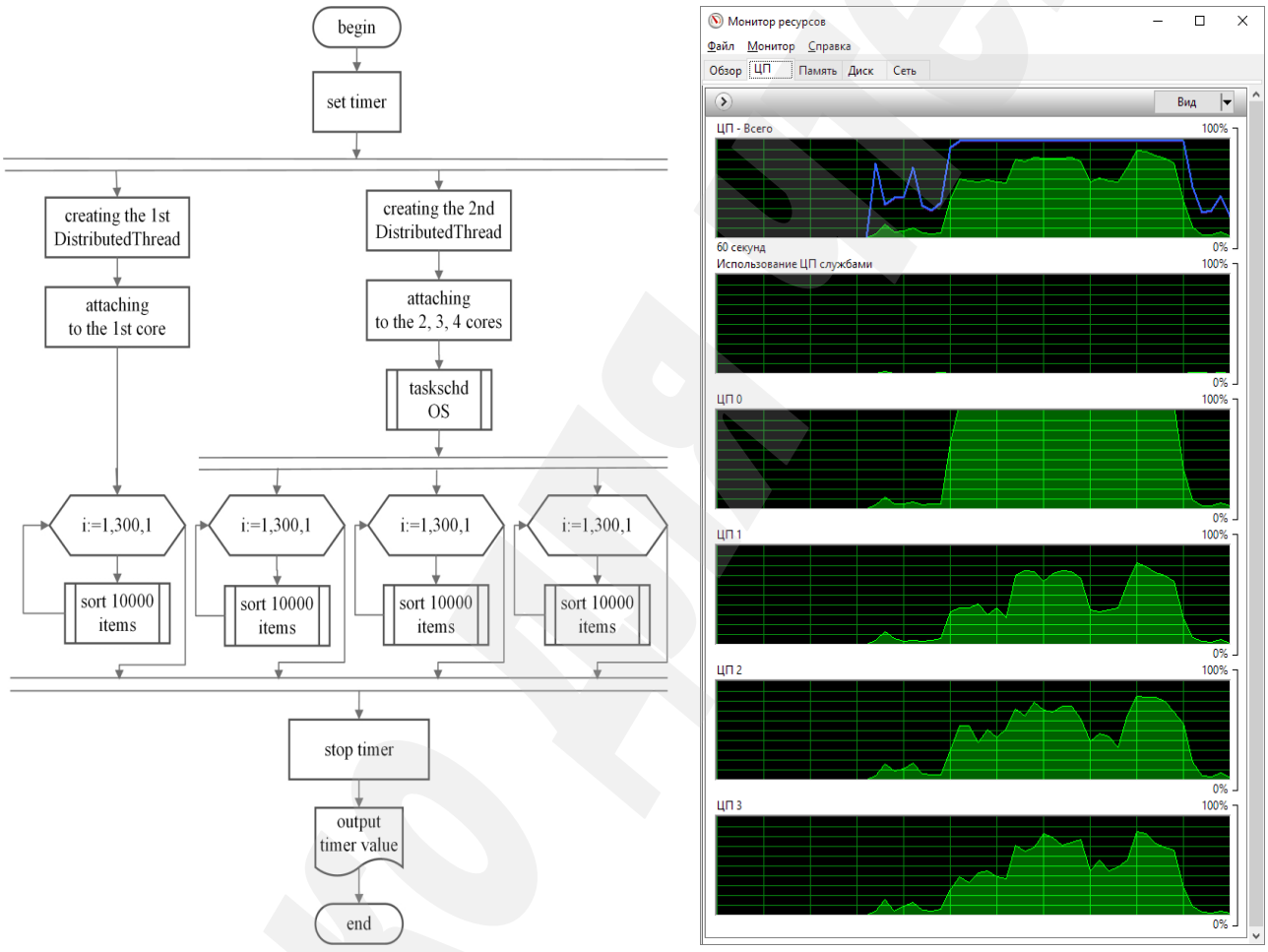
У Проекті № 1 виконане імітаційне моделювання роботи автопланувальника ОС: створено 4 паралельних потоки по 150 ітерацій на 1-4-му ядрах (рис. 4, *а*), зважаючи на те, що автопланувальник намагатиметься рівномірно навантажити кожне з чотирьох ядер (рис. 4, *б*).



а *б*

Рис. 4. Використання CPU за Проектом № 1: *а* – блок-схема; *б* – осцилограма

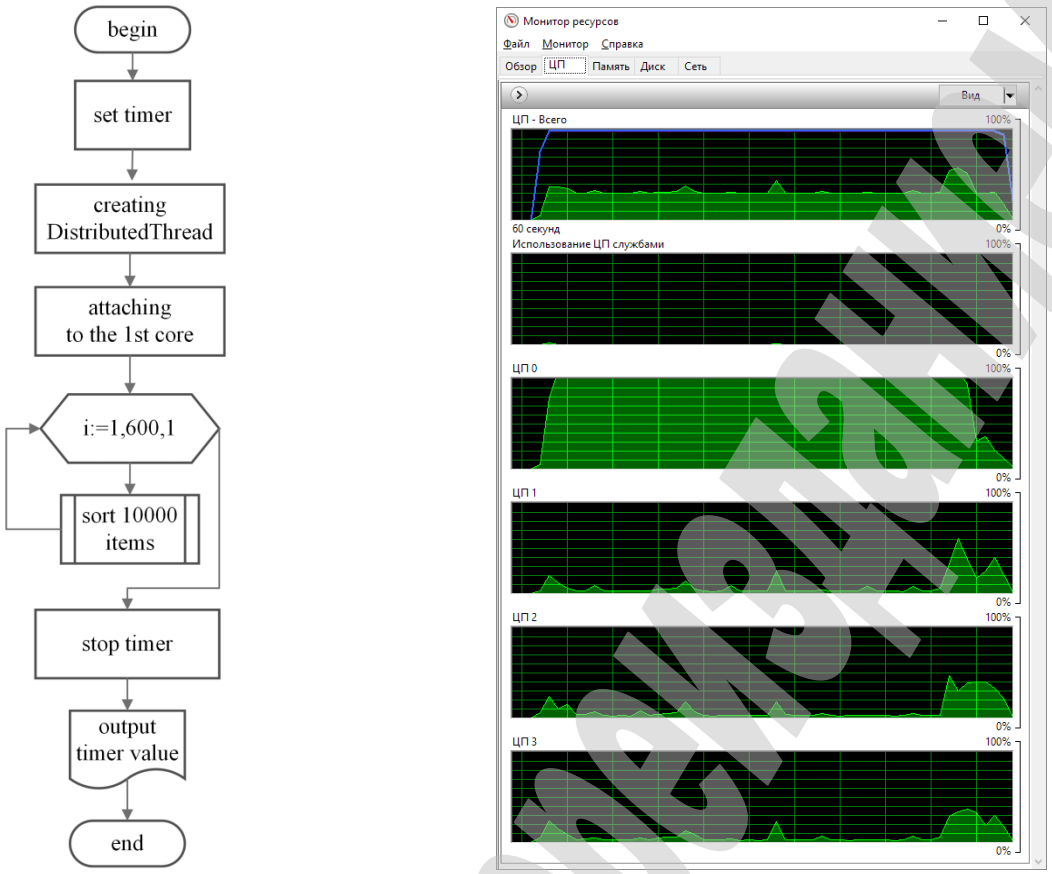
За Проектом № 2 створюємо два паралельних потоки: 300 ітерацій на 1-му ядрі та 300 ітерацій, розподілених між 2–4-им ядрами (рис. 5, *а*). Другий потік розподіляється автопланувальником ОС Windows на три цикли по X_i ітерацій у кожному так, що: $X_2+X_3+X_4=300$. Точні значення X_i не визначені, так як невідомо, як саме автопланувальник ОС поділить потік між трьома ядрами. Але, з моніторингових даних на рис. 4, *б* видно, що автопланувальник ОС намагається рівномірно навантажити ядра 2–4. Таким чином, підтверджуються алгоритм, реалізований у Проекті № 1 для імітації дій автопланувальника ОС з розподілу завдань між ядрами процесора.



а *б*

Рис. 5. Використання CPU за Проектом № 2: *а* – блок-схема; *б* – осцилограма

У Проекті № 3 реалізований алгоритм, за яким створений один послідовний потік з 600 ітерацій на 1-му ядрі (рис. 6). Аналіз результатів цього алгоритму є важливим, тому що в 4-ядерних однокристальних процесорах, на базі яких побудовані обчислювальні системи БПЛА (найчастіше це різні моделі процесорів ARM Cortex), немає підтримки технології віртуальної багатопотоковості Hyper-Threading, як у процесорів компанії Intel. Тому, якщо не розділяти завдання на потоки, як запропоновано в цій роботі, то, наприклад, для процесора ARM Cortex A9 можливо у режимі Asymmetric Multiprocessing (AMP) отримати все навантаження на 1-ше ядро (рис. 6, *а*) [2].



а *б*

Рис. 6. Використання CPU за Проектом № 3: *а* – блок-схема; *б* – осцилограма

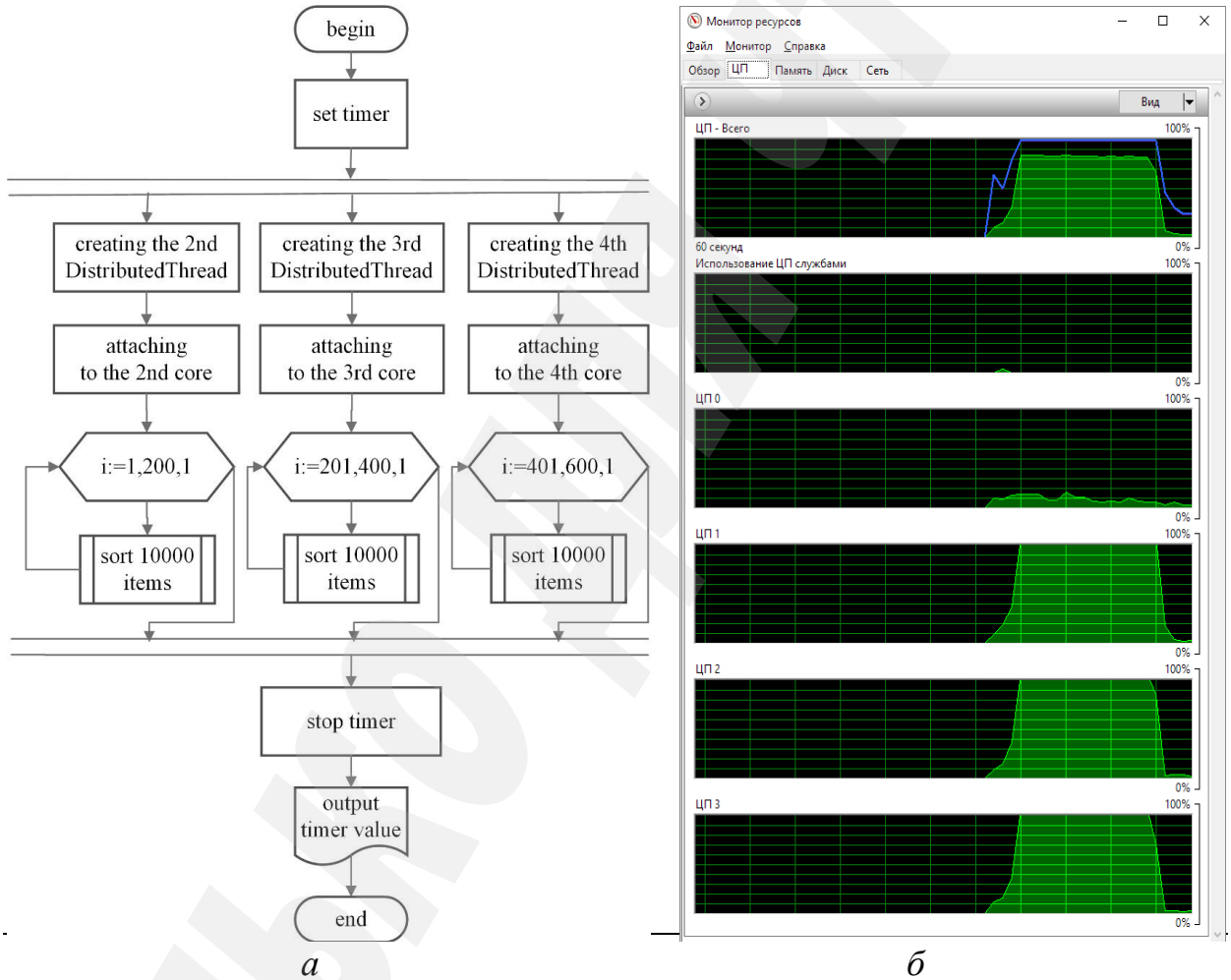
Тоді таймінг виконання завдання за Проектом № 3 зросте майже втричі у порівнянні з бенчмарком Проекту № 1 (табл. 1).

Таблиця 1

Результати бенчмаркінгу за чотирма проектами

Показник	Проект № 1	Проект № 2	Проект № 3	Проект № 4	Використане ПЗ
Таймінг (AverageTime), с	12,7	24,6	47,2	16,5	Власна розробка
Енергоспоживання (Average Power), Вт	58,5	38,9	27,3	49,3	HWMonitor v.1.33.0
Температура (Temperature), ºС	75	62	62	80	HWMonitor v.1.33.0
Використання CPU (Utilization), %	100	56	31	77	HWMonitor v.1.33.0
Кількість обертів кулера на процесорі (Fan's number of rotations), RPM	1900	1560	1505	1599	HWMonitor v.1.33.0

У Проекті № 4 реалізовано алгоритм, згідно з яким 1-ше ядро не задіяне, а до 2–4-го ядер зроблена прив'язка 3 паралельних потоків по 200 ітерацій кожен (рис. 7, *а*). Але, за моніторингом ресурсів на 1-му ядрі все ж зареєстрована активність, що можна пояснити обробкою фонових процесів ОС (рис. 7, *б*). Саме тому, запропонований алгоритм ручної прив'язки тестового завдання на 2–4-те ядра можна вважати найоптимальнішим. В такому разі, розвантажене 1-ше ядро найшвидшим чином зможе відпрацювати завдання, пов'язані з критичним застосуванням БПЛА. Наприклад, це коригування маршруту при появі механічної перешкоди на шляху БПЛА, перехід на оптичну навігацію при неможливості отримання GPS-координат тощо. У якості тестового завдання Проекту № 4 в такому разі можна розглядати компресію отриманого відеопотоку, підготовку даних для передачі за допомогою модуля зв'язку з БПЛА до наземного командного центру або до головного дрону.



а *б*

Рис. 7. Використання CPU за Проектом № 4: *а* – блок-схема; *б* – осцилограма

При бенчмаркінгу можлива розбіжність показників, яка обумовлена тим, що програми моніторингу зчитують показання датчиків раз на 2, 3 або 5 с. Це може не збігатися з таймінгом проектів, що тестуються.

За даними моніторингу ресурсів (табл. 1), отриманими тестувальником HWMonitor, вдалося суттєво знизити температуру на 1-му ядрі шляхом ручного перерозподілу завдання між ядрами процесора за Проектом № 4 (рис. 8). Май-

же на 13 % у порівнянні зо всіма іншими проектами. Завдяки цьому при запуску нового фонового процесу ОС, який може бути пов'язаним з критичним застосуванням БПЛА (втраті GPS-координат, включення задачі оптичної навігації, миттєвий перерахунок курсу за причини пориву вітру й т. ін.), значно підвищується ефективність використання КС.

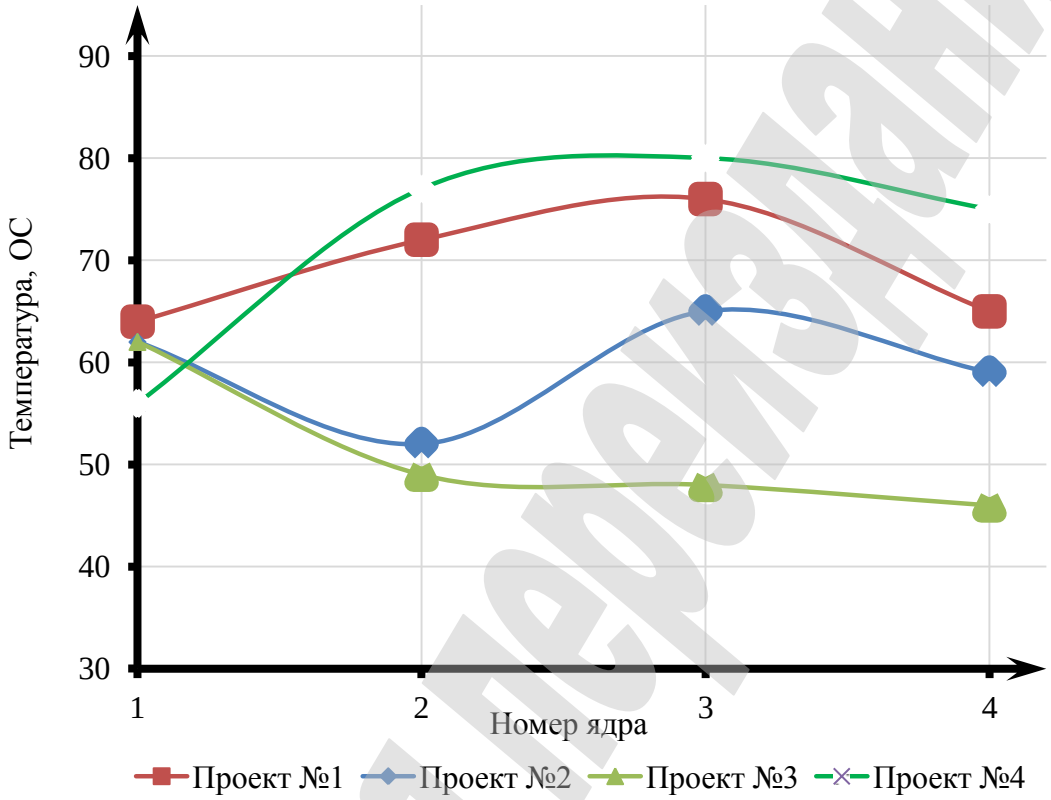


Рис. 8. Температура однокристального процесору з деталізацією по ядрах

Зважаючи на те, що за бенчмаркінгом алгоритм саме Проекту № 4 виявився оптимальнішим, пояснимо детальніше програмний код запропонованого алгоритму.

По-перше, підключаємо простір імен, в якому оголошено клас *DistributedThread*, розроблений авторами:

```
using System;
using System.Threading;
using DistributedWorkManager;

namespace ConsoleApplication1{
  class Program{
```

Наступним кроком встановлюємо таймери і розмір масиву:

```
    static Random rnd = new Random();
    static readonly int N = 10000;
    static void Main(string[] args){
        DateTime t= DateTime.Now;
```

Клас *CountdownEvent*, який використовується в наступному коді, є необхідним для синхронізації та розблокування потоків, що очікують у черзі. В такому разі основний код не буде виконуватись до тих пір, поки не отримає певну кількість сигналів (у нашому випадку – три).

Потім створюємо об'єкт класу *DistributedThread*, прив'язуємо його до 2-го ядра, запускаємо. Для дотримання логіки дій називаємо змінну *thread2* (а не *thread1*). Така нотація надає наочності, до якого саме ядра виконується прив'язка:

```
using (CountdownEvent e = new CountdownEvent(3)){
   DistributedThread thread2 = new DistributedThread(delegate
    {
     for (int k = 0; k < 200; k++){
        int[] intArray = new int[N];
        for (int i = 0; i < N; i++)
             intArray[i] = rnd.Next(100 + k, 1000 + k);
        int temp, j;
        for (int i = 1; i < intArray.Length; i++){
           temp = intArray[i];
            j = i - 1;
           while (j >= 0 && intArray[j] > temp){
                     intArray[j + 1] = intArray[j];
                     j--;
             }
             intArray[j + 1] = temp;
       }
     }
     e.Signal();
   });
     thread2.ProcessorAffinity = 2;
   thread2.Start();
```

Створення та робота потоків *thread3* та *thread4* аналогічні потоку *thread2*, тому код опускаємо. Після завершення роботи всіх трьох потоків на консоль виводимо значення таймеру.

```
            e.Wait();
        }
        Console.WriteLine($"{DateTime.Now - t}");
      }
    }
}
```

Нижче представлена діаграма, що наочно демонструє підвищення споживаної енергії в залежності від кількості задіяних ядер (рис. 9).

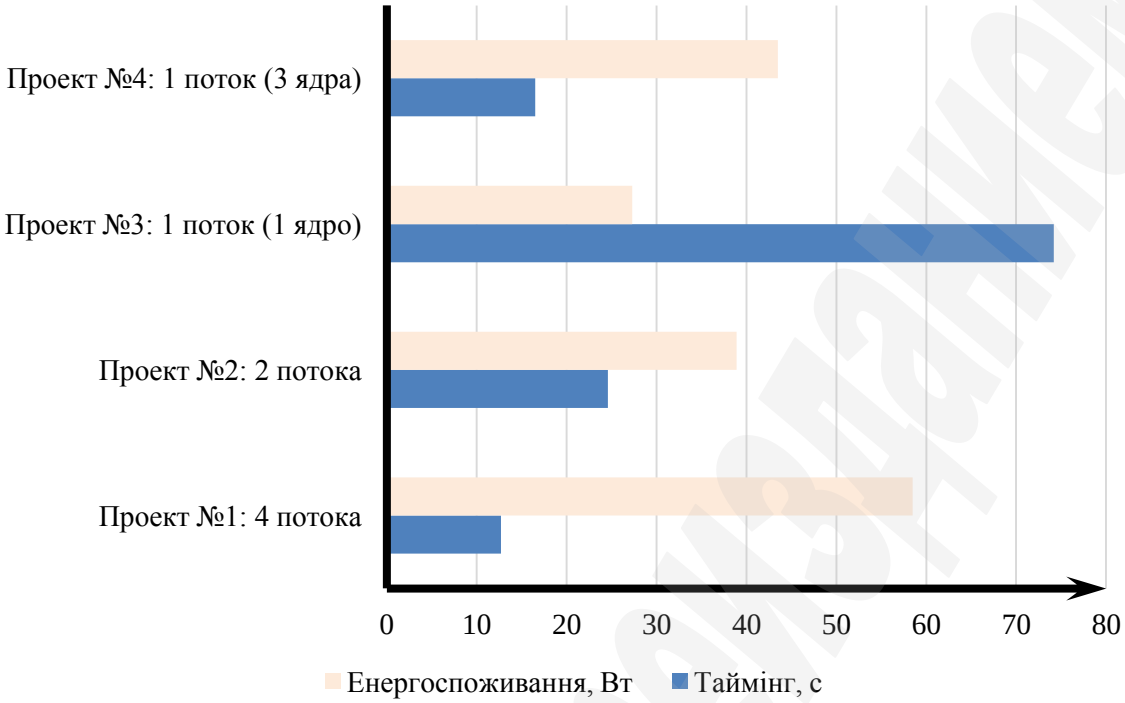


Рис. 9. Енергоспоживання в однокристальному обчислювальному процесорі в залежності від кількості навантажених потоків

Підхід з використанням 4 потоків є найбільш ефективним за таймінгом, однак, він також є найменш енергозберігаючим. Тому найбільш ефективне використання обчислювальних ресурсів БПЛА забезпечить розподіл згідно з алгоритмом Проекту № 4 завдань, не пов'язаних з навігацією та з керуванням дроном.

7. SWOT-аналіз результатів досліджень

Strengths. Під час використання БПЛА всі стикаються з необхідністю мати стабільно працюючу систему, особливо при можливості критичного застосування БПЛА. Тому вкрай важливо миттєве відпрацювання обчислювальною системою БПЛА необхідних змін курсу при появі перешкод, при втраті зв'язку з оператором, при різкому падінні заряду батареї й т. ін. У такому разі алгоритми розвантаження 1-го ядра, на якому зазвичай виконуються всі системні процеси, від обрахування моніторингових та функціональних завдань можуть принести безперечну користь. Крім того, запропоноване додаткове живлення КС БПЛА від енергії вимірювального сигналу дозволить не відключати, наприклад, модулі зв'язку, як це передбачено керуючими системами БПЛА задля економії заряду батареї. Запропонований підхід у комплексі суттєво підвищує ефективність використання БПЛА на базі багатоядерного обчислювача та покращує умови його взаємодії з наземним командним центром.

Weaknesses. Слабким місцем дослідження є відсутність аналізу енергозатрат при виконанні різних за якістю завдань з використанням запропонованого алгоритму диспетчеризації у порівнянні з автопланувальником декількох моделей CPU. Повна автоматизація поядерної диспетчеризації ускладняється тим,

що кожен алгоритм запускається автономно, тобто немає можливості вибору між режимами або найкращого таймінгу, або найменшого енергоспоживання.

Opportunities. У перспективі доцільно провести дослідження впливу розподілення обчислень між ядрами CPU для БПЛА на базі різних моделей однокристальних процесорів. Крім того, дуже корисним було б чисельне визначення долі спожитої енергії, яка може бути зкомпенсованою за рахунок додаткового живлення КС від енергії вимірювального сигналу для різних наборів датчиків на борту БПЛА.

Threats. У галузі створення нових моделей БПЛА кожна фірма-виробник створює власне пропрієтарне ПЗ для керування навігаційними та обчислювальними процесами в КС БПЛА. Тому розроблене алгоритмічне та програмне забезпечення має дуже малі шанси для виходу на світовий ринок. Його ніша – це застосування на окремих підприємствах, які пов'язані не з економічними галузями застосування БПЛА, а з критичними ситуаціями (моніторинг викиду в атмосферу отруйних речовин, пошук постраждалих людей або техногенних загроз тощо). Зараз важко передбачити негативні ризики розробленого підходу. Але можна сказати, що додаткові витрати не потрібні для використання запропонованого підходу в конкретних виробничих ситуаціях.

8. Висновки

1. Проаналізовано десять найбільш поширених фізичних впливів на первинні перетворювачі енергії вимірювального сигналу від датчиків БПЛА. Встановлено, що п'єзоелектричні датчики є найбільш енергетично привабливими для забезпечення додаткового живлення модулів БПЛА.

2. Побудована модель взаємодії вузлів БПЛА як інформаційно-вимірювальної системи. Визначено, що безперервна робота ІВС буде забезпечена за умови, коли додавання енергії до системи буде більше, ніж вона витрачається на всіх вузлах схеми.

3. В результаті аналізу механізму роботи автоматичних планувальників завдань ОС з'ясовано, що в зазначених умовах здійснюється розподіл процесів між усіма чотирма ядрами CPU. Але, в такому разі 1-ше ядро не має резервів у порівнянні з іншими ядрами по навантаженості (навантажене на 100 %) та температурі (є на рівні з іншими задіяними ядрами). Кількість обертів кулера та енергоспоживання процесора в режимі автопланування є найвищим. Тому, при виникненні критичної ситуації, коли необхідним може бути суттєве збільшення обчислювань на 1-му ядрі для задач системи управління БПЛА, такі задачі можуть стояти у черзі за попередньо запущеними завданнями. З точки зору життєзабезпечення БПЛА це є неприпустимим.

4. Дослідження показали, що розпаралелювання завдань, які можуть бути виконані як багатопоточні (компресія відеопотоку, передача даних різними каналами зв'язку та ін.), підвищує ефективність використання БПЛА. Прив'язка процесів до трьох ядер замість чотирьох на 23 % збільшує загальний час виконання завдання у порівнянні з часом виконання аналогічного завдання на одному ядрі.

5. Отримані результати досліджень розроблених алгоритмів диспетчеризації завдань на 4-ядерному процесорі дозволили визначити як найоптимальніший алгоритм резервування ресурсів 1-го ядра багатоядерного однокристально-

го CPU для обчислень першочергової важливості. Застосування такого алгоритму обчислень сприяє підвищенню ефективності використання БПЛА. Так, наприклад, для DJI Phantom 4 розподіл завдань лише на 2–4-те ядро зменшує енергоспоживання на 10,3 % та забезпечує подовження часу польоту на 3,1 хв. Навіть при немаксимальній швидкості польоту цієї моделі БПЛА у 50 км/год (Режим P) [21] це збільшить дальність професійної аерофотозйомки на 1,3 км з урахуванням повернення на базу.

Подяка

Робота виконана за підтримки Міністерства освіти і науки України в рамках держбюджетних науково-дослідних робіт Чорноморського національного університету ім. Петра Могили за темами «Створення поліметричних датчиків інформаційно-вимірювальних систем з живленням елементів від енергії вимірювального сигналу» (державний реєстр № 0115U000316, 2015–2016 рр.) та «Розроблення бездротових енергонезалежних інформаційно-вимірювальних мереж критичного застосування військово-цивільного призначення» (державний реєстр № 0117U000447, 2017–2018 рр.).

Література

1. Tencent and ZEROTECH Unveil Commercial Drone Based on Qualcomm Snapdragon Flight Platform [Electronic resource] // Qualcomm Technologies, Inc. – 2016, January 5. – Available at: \www/URL: https://www.qualcomm.com/news/releases/2016/01/05/tencent-and-zerotech-unveil-commercial-drone-based-qualcomm-snapdragon

2. Cortex™-A9. Revision: r4p1. Technical Reference Manual [Electronic resource]. – ARM, 2012. – Available at: \www/URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388i/DDI0388I_cortex_a9_r4p1_trm.pdf

3. Development of multi-threaded applications using optimization method for platforms [Electronic resource] // Intel Software Developer Zone. – 2011, February 3. – Available at: \www/URL: https://software.intel.com/ru-ru/articles/61695

4. Task Scheduler How To... [Electronic resource] // Microsoft TechNet. – Available at: \www/URL: https://technet.microsoft.com/en-gb/library/cc766428(v=ws.11).aspx

5. Prostaia model' planirovshchika OS [Electronic resource] // Habrahabr. – 2012, October 12. – Available at: \www/URL: https://habrahabr.ru/post/154609/

6. Troubleshooting Task Scheduler [Electronic resource] // Microsoft TechNet. – Available at: \www/URL: https://technet.microsoft.com/en-gb/library/cc721846(v=ws.11).aspx

7. Tanenbaum, A. S. Modern Operating Systems [Text] / A. S. Tanenbaum, H. Bos. – Ed. 4. – Amsterdam, The Netherlands: Pearson Prentice-Hall, 2015. – 1072 p.

8. Arhitektura: gibkaia, effektivnaia [Text] // CHIP. – 2013. – No. 9. – P. 52–53.

9. Krainyk, Y. Hardware-oriented turbo-product codes decoder architecture [Text] / Y. Krainyk, V. Perov, M. Musiyenko, Y. Davydenko // Proceedings of the 2017 IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2017), Bucharest, Romania, September 21–23, 2017. – Vol. 1. – P. 151–154. doi:10.1109/idaacs.2017.8095067

10. Burlachenko, I. Devising a method for the active coordination of video cameras in optical navigation based on multi-agent approach [Text] / I. Burlachenko, I. Zhuravska,

M. Musiyenko // Eastern-European Journal of Enterprise Technologies. – 2017. – Vol. 1, No. 9 (85). – P. 17–25. doi:10.15587/1729-4061.2017.90863

11. Nikiforov, V. V. Basic Requirements to the SPIIRAS Transactions Paper Format Feasibility of Real-Time Applications on Multicore Processors [Text] / V. V. Nikiforov // SPIIRAS Proceedings. – 2009. – Vol. 8. – P. 255–284. doi:10.15622/sp.8.12

12. Zhuravska, I. M. Self-powered information measuring wireless networks using the distribution of tasks within multicore processors [Text] / I. M. Zhuravska, O. O. Koretska, M. P. Musiyenko, W. Surtel, A. Assembay, V. Kovalev, A. Tleshova // Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments, Wilga, Poland. – 2017, August 7. – P. 1–13. doi:10.1117/12.2280965

13. Sharapov, V. The electromechanical feed-back in piezoceramic sensors and transducers [Text] / V. Sharapov, I. Sarwar, I. Chudaeva, M. Musienko // Proceedings of the IEEE Ultrasonics Symposium, Sendai, Japan, October 5–8, 1998. – Vol. 1. – P. 543–544. doi:10.1109/ultsym.1998.762208

14. Trasvina-Moreno, C. Unmanned Aerial Vehicle Based Wireless Sensor Network for Marine-Coastal Environment Monitoring [Text] / C. Trasvina-Moreno, R. Blasco, A. Marco, R. Casas, A. Trasvina-Castro // Sensors. – 2017. – Vol. 17, No. 3. – P. 460. doi:10.3390/s17030460

15. CPU Stability Test [Electronic resource] // BenchmarkHQ. – 2017. – Available at: \www/URL: http://www.benchmarkhq.ru/russian.html?/b.html

16. Chakos, B. Here's how [Text] / B. Chakos // PCWorld. – 2013. – P. 89.

17. Property Process.ProcessorAffinity [Electronic resource] // Microsoft Developer Network. – 2016, October. – Available at: \www/URL: https://msdn.microsoft.com/ru-ru/library/system.diagnostics.process.processoraffinity(v=vs.110).aspx

18. Intel® Math Kernel Library – Documentation [Electronic resource] // Intel Software Developer Zone. – 2017, September 13. – Available at: \www/URL: https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation

19. Grama, A. Introduction to Parallel Computing [Text] / A. Grama, G. Karypis, V. Kumar, A. Gupta. – Ed. 2. – Boston, MA, US: Addison-Wesley, 2003. – 656 p.

20. Richter, J. CLR via C# [Text] / J. Richter. – Ed. 4. – Redmond, WA, US: Microsoft prePress, 2012. – 813 p.

21. Phantom 4 Pro: specifications [Electronic resource] // DJI. – 2017. – Available at: \www/URL: https://www.dji.com/ru/phantom-4-pro/info