UDC 629.734:004.451.45
DOI: 10.15587/2312-8372.2017.117889

**Zhuravska I.,
Borovlyova S.,
Kostyria M.,
Koretska O.**

EFFICIENCY IMPROVEMENT OF USING UNMANNED AERIAL VEHICLES BY DISTRIBUTION OF TASKS BETWEEN THE CORES OF THE COMPUTING PROCESSOR

Розглянуті алгоритми розпаралелювання завдань між ядрами 4-ядерних процесорів обчислювальних систем безпілотних літальних апаратів (БПЛА). Показано, що вивільнення 1-го ядра та розподіл завдання між 2–4-ми ядрами на 10,3 % зменшує енергоспоживання процесору. Застосування розробленого алгоритму у комплексі з запропонованим додатковим живленням від енергії вимірювального сигналу датчиків підвищує ефективність використання БПЛА.

Ключові слова: *обчислювальні системи безпілотних літальних апаратів (БПЛА), 4-ядерний процесор, імітаційне моделювання.*

1. Introduction

The computing systems of most unmanned aerial vehicles (UAVs) are based on single-chip 4-core 32-bit processors ARM Cortex, Qualcomm Snapdragon, etc. [1, 2]. The number of computational flows in them is equal to the number of physical cores, between which tasks can be distributed (orientation in space, traffic control, receiving and processing indicators from sensors, etc.).

For a computer system with limited resources (computing or power supply), the following problem arises: the operation of an automatic scheduler of operating system (OS) flows leads to an unjustified processor load. But for mobile computers on which UAV construction is based, it is often more important to economize in power consumption (in practice – battery life) than to attract all available cores in a single-chip computing processor (CPU) in the computational process.

Therefore, the actual task is to improve the efficiency of using UAVs by developing better, than automatic, algorithms for the operation of a multi-core processor, on the basis of which a UAV computer system is built.

2. The object of research and its technological audit

The object of research is UAV computer system. Most modern UAVs are based on 2- or 4-core single-chip processors, between which the OS automated scheduler tries to evenly distribute computational tasks. At the same time, the first core can instantly become extremely congested in the case of an urgent task from the UAV control system. Therefore, *the subject of research* is the complex indicators of the state of the processor cores under various algorithms for the distribution of tasks between the cores of a multi-core single-chip processor.

Despite the fact that direct in-situ simulation on UAV can lead to critical application of the latter, it was decided to simulate the dispatching of tasks on a stationary CS with a 4-core single-board processor.

The test was conducted on a CS with an Intel Core i7-4790 3.60GHz CPU with disabled Hyper-Threading Technology support [3]. For the experiment, a test program was created, which sorts an array of 10.000 elements at each iterations.

Within the framework of this work, it is decided to independently implement the load of the cores by given threads when writing software in C#. It is also planned to implement the process of reference comparison with automatic dispatching in Windows. Otherwise, this process can be defined as benchmarking the efficiency of using the cores of a multi-core processor.

Such criteria are investigated: task timing, CPU power consumption, temperature and percentage load of each core and the entire processor during the execution of each of the four test tasks. Timing is evaluated based on the results of the work of the own written software. Temperature (or heat dissipation – TDP parameter (Thermal Design Power), power consumption and percentage usage of the processor can be estimated using a pre-started monitoring utility.

One of the most problematic places is overload and overheating to the processor of the UAV computer system. This can lead to CS freezing, loss of control and crash of the drones. In addition, in a critical situation, the processor is overloaded, it will not provide the urgently needed calculations of the UAV rate change or the transition to the autonomous flight mode.

3. The aim and objectives of research

The aim of research is the efficiency improvement of using UAVs with the lengthening of the flight time of the latter by reducing the congestion and power consumption of the computer system.

To achieve this aim, it is necessary to solve the following tasks:

1. To analyze the possibility of dispatching tasks to a UAV processor with multi-threaded organization of task distribution between CPU cores and to develop appropriate algorithms.

2. To perform a simulation of the developed algorithms for the 4-core CPU.

3. To study the technical parameters of the UAV CPU state (the temperature in the section along the cores, the power consumption and the CPU utilization, the number of cooler rotations to provide the necessary heat dissipation, etc.) in the implementation of various dispatch algorithms.

4. To investigate the possibility of increasing the efficiency of using UAVs at the expense of sources of additional energy consumption.

5. To determine the optimal algorithm for distribution of tasks between the cores of a multicore single-chip processor.

4. Research of existing solutions of the problem

The problem of optimizing energy consumption, load and preventing overheating of computer components of UAV computer systems is widely considered in the resources of the world scientific periodicals. The main direction of its solution is multi-threaded dispatching tasks between the cores of a multi-core processor.

However, existing algorithms for scheduling task distribution at the OS level [4, 5] can solve the problem of preventing the maximum load of the processor and each of its cores to almost 100 % [6]. In addition, it is possible that the tasks will be queued for the processor [7], which is unacceptable for the tasks of optical navigation and UAV control.

To reduce power consumption in battery-powered systems, the mechanism of switching off communication modules [8] is sometimes used for high-speed data decoding or for compression of the received video [9, 10]. But, in that case, there may be a so-called Dahl effect, when part of the processor's cores is idle, despite the fact that the timeliness of some tasks is not ensured [11]. Such scheduling of tasks is unacceptable for a UAV, because it, for example, can't receive control signals from the main UAV or ground command center.

The loss of energy for a constant connection to the battery supply of communication modules can be partially compensated by the mechanisms described in [12]. To select the type of physical impact on the primary converter, the ten most common effects and effects they cause are analyzed: the Seebeck effect, the pyroelectric effect, etc. [13]. As a result of the analysis it was revealed that the piezoelectric effect is the most energetically attractive for the solution of the task of constructing effective information and measurement systems (IMS), to which UAVs can also be assigned. That is, this means that piezoelectric sensors are best suited for feeding the components of such IT system from the energy of the measurement signal. Piezoelectric transducers allow solving various tasks: measuring mechanical parameters (pressure, acceleration, mass, angular velocities, moments, deformations, etc.).

It is also important to choose a wireless technology for transmitting measurement information from a wireless information transfer device (WITD) to an information collection system (ICS). As a result of the analysis, the following promising network specifications for network protocols were chosen, such as ZigBee and LoRa (depending on the type of task) [14].

An evaluation of the effectiveness of measures taken to improve the efficiency of the use of CS components can be carried out using a number of software products for monitoring and changing the load regimes of processors [15]. However, it should be noted that for CS UAV categorically can't use the so-called «fry tests» (AIDA64, LinX, IntelBurnTest, Prime95, OCCT Perestroika etc.). The principle of their action is stress testing, when the processor performs specialized intensive calculations that simulate the scenario of operation under the most severe conditions and lead to maximum heating of the core [16]. The use of such tests can lead to irreparable consequences with the burnout of individual CS components. With the introduction of limit values in the test modes of such utilities, it is possible to derive the investigated CS.

Therefore, for benchmarking, it is advisable to use software, which among other indicators of the hardware state of the compressor displays the readings of the sensors built into such a sensor (Core Temp, CPU-Z, HWMonitor, SpeedFan, etc.).

The HWMonitor v.1.33.0 utilities have been selected for benchmarking.

Thus, the results of the analysis lead to the conclusion that the issue of improving the UAV efficiency use requires additional research.

5. Methods of research

So, it was decided to direct efforts to develop methods and algorithms for dispatching tasks on various computer components of the UAV. First, let's take a closer look at the problem of the optimal CPU load in the processor core.

Binding of the process with the specified cores does not cause any difficulties, it is necessary to use the *ProcessorAffinity* property of the Process class, which is registered in the *System.Diagnostics* namespace [17]. Depending on the bitmask that is assigned to this property, it is possible to obtain all combinations of cores on the selected device. If there are 4 cores, the OS threads scheduler automatically use them all, trying to get the maximum benefit from the multi-core processor (hereinafter – the CPU). In this case, *ProcessorAffinity* is 15 and is determined by the formula:

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

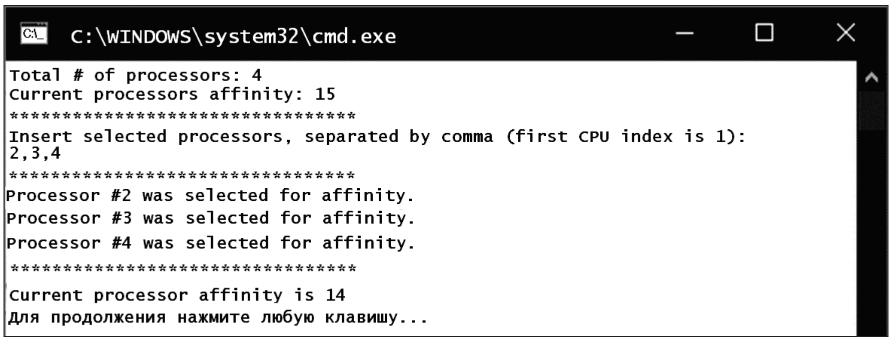
where n – the number of cores of the multi-core processor, in the test $n = 4$.

It should be noted that when developing applications, it is recommended to avoid using the core on which the operating system performs data transfer, memory clearing and other system processes. Typically, all such processes are limited to the first core of the processor. Thus, excluding the first core from a computational task can lead to improved application performance.

Therefore, let's select the cores from the 2nd to the 4th for the benchmarking and assign a new bitmask, obtaining the value 14 (Fig. 1).

Accordingly, for each of the four cores of the quad-core processor, the value of *ProcessorAffinity* will be:

1) for the 1st core $ProcessorAffinity = 1$;
2) for the 2nd core $ProcessorAffinity = 2$;
3) for the 3rd core $ProcessorAffinity = 4$;
4) for the 4th core $ProcessorAffinity = 8$.



```
C:\WINDOWS\system32\cmd.exe
Total # of processors: 4
Current processors affinity: 15
**********************************
Insert selected processors, separated by comma (first CPU index is 1):
2,3,4
**********************************
Processor #2 was selected for affinity.
Processor #3 was selected for affinity.
Processor #4 was selected for affinity.
**********************************
Current processor affinity is 14
Для продолжения нажмите любую клавишу...
```

Fig. 1. The value of the *ProcessorAffinity* property when using 4 cores and the 2–4th cores

It should be noted that it will be more useful to bind not to the processor cores, but to use a binding to the threads, but this path contains many difficulties. The point is that Intel is trying to facilitate the work of the programmer, taking the binding of threads to the cores to itself [3]. Obviously, this makes sense, because most large companies are developing products for a mass user, for whom understanding and using their own parallelization is not necessary.

For high mathematical productivity, it is possible to use the pre-built components library for programming applications in C/C++ and Fortran – Intel Math Kernel Library (MKL) [18]. For fast cycle operation, it is advisable to enable autoparallelization in the Intel compiler – Intel C++ Compiler (ICC) [3]. To maximize the use of the cache include support for Hyper-Threading technology. Although the availability of such technology is a good marketing solution, however, for those who still need to determine the parallelization manually, puts on the way certain obstacles. The programmer can set his own thread binding to the cores, but the Intel library will bring them to naught.

In the case under investigation, additional difficulties arise because the .NET stream that is run by the Common Language Runtime (CLR) does not correspond to the OS thread, and only the threads of the operating system can bind to the cores [19, 20]. To solve this problem, it is possible to use the above methods of the class *Thread*:

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

In this case, the code between these calls is performed on a single OS thread, significantly weakening the management of the CLR threads.

Now it is possible to go to the binding itself. Obtain the OS threads of a .NET application using *Process.GetCurrentProcess().Threads* – a collection of threading objects. To obtain the OS thread that is currently running, it is necessary to use the following code:

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

Using the returned ID, it is possible to find the flow of the executable task and use the property of *ProcessThread.ProcessorAffinity*, which is very similar to the *Process.ProcessorAffinity* described above.

Next is the *DistributedThread* class, which allows to start threads on selected cores. The bottom line is encapsulation of the usual Thread-object, limiting its current operating system and setting the desired binding to the core:

```
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 GetCurrentProcessor-
       Number();
       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 InvalidOperationEx-
                    ception();
                }
            }
            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();
            }
        }
    }
}
```

Now the developed class *DistributedThread* can be used in the following projects.

To determine the optimal algorithm for distributing tasks between the cores, four projects are developed, the first of which (Project No. 1) simulates the work of the Windows Explorer, which tries to evenly distribute the tasks between the hardware 4 cores of one processor. This Project No. 1 is benchmark for the relative evaluation of other projects by timing criteria (s), power consumption (W), temperature (°C) and percentage load to the processor, taking into account the background tasks of the operating system.

Each of the projects is carried out with the following conditions:

1. Support for Hyper-Threading Technology is disabled.
2. The HWMonitor utility is active.
3. Projects are run in turn with the same number of background tasks.

To optimize the UAV's power consumption, it is necessary to focus on the functional composition of UAV components. Since UAV has a sufficiently large number of sensors, the UAV can be classified as an information and measuring system (IMS). Therefore, it is advisable to consider the possibility of additional feeding of the UAV components from the energy of the measuring signal.

The main stages in solving the problem of additional power supply of the UAV are (Fig. 2):

– selection of the type of physical impact on the primary converter, as well as the sensor (S), which operation is based on this type of signal conversion;
– selection of the parameters of the primary converter capable of satisfying both informational and energy needs of the information and measuring device (IMD);
– selection of the energy storage element – information processing device (IPD);
– selection of the standard for transmission of measurement results from the WITD to the ICS.

To investigate the main processes while ensuring the continuous operation of the IMS when feeding components from the energy of the measuring signal, a model for the interaction of the nodes of the circuit is constructed with allowance for time relationships (Fig. 3).

The energy that the sensor receives from the energy of the measuring signal (the first link) accumulates in the energy storage element (second link). The function of energy dependence on time (loss, accumulation, consumption) has the form $F_1(t)$. Depending on the chosen scheme for connecting the accumulation element, there are losses in the transmission of energy with a transmission factor δ_1. In the scheme of accumulation with time, energy is also lost in accordance with the law $F_2(t)$. The load is consumed according to the function $F_3(t)$ (third link). In this case, the transmission coefficient δ_2 describes the losses when the load is connected to the power storage scheme.

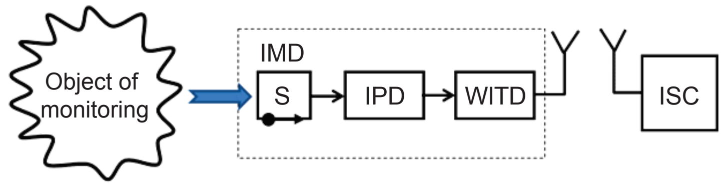


Fig. 2. Components of the information measuring system

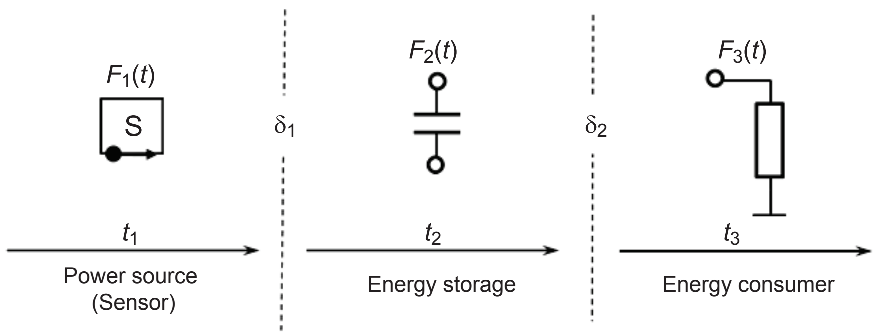


Fig. 3. Model of interaction of the nodes of the information measuring system

If the relationship between the described variables is fulfilled:

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

in this case, the continuous IMS operation will be ensured. That is, the addition of energy to the system will be greater than it is consumed at all nodes of the circuit.

6. Research results

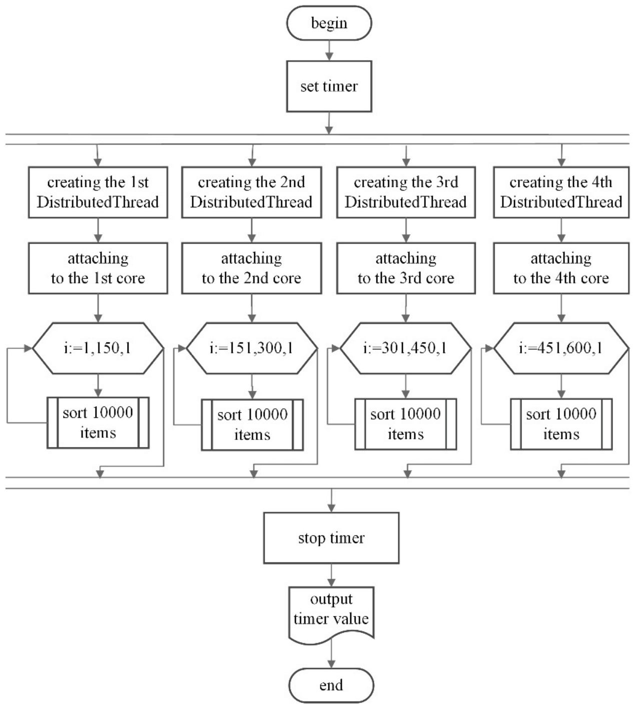
The content of the projects and the results of their implementation are shown in Fig. 4–7 and are summarized in Table 1.

In Project No. 1, a simulation of the operation of the OS automatic scheduler is performed: four parallel flows are created for 150 iterations on the cores from the 1st to the 4th (Fig. 4, *a*), in spite of the fact that the automatic scheduler tries to evenly load each of the four cores (Fig. 4, *b*).

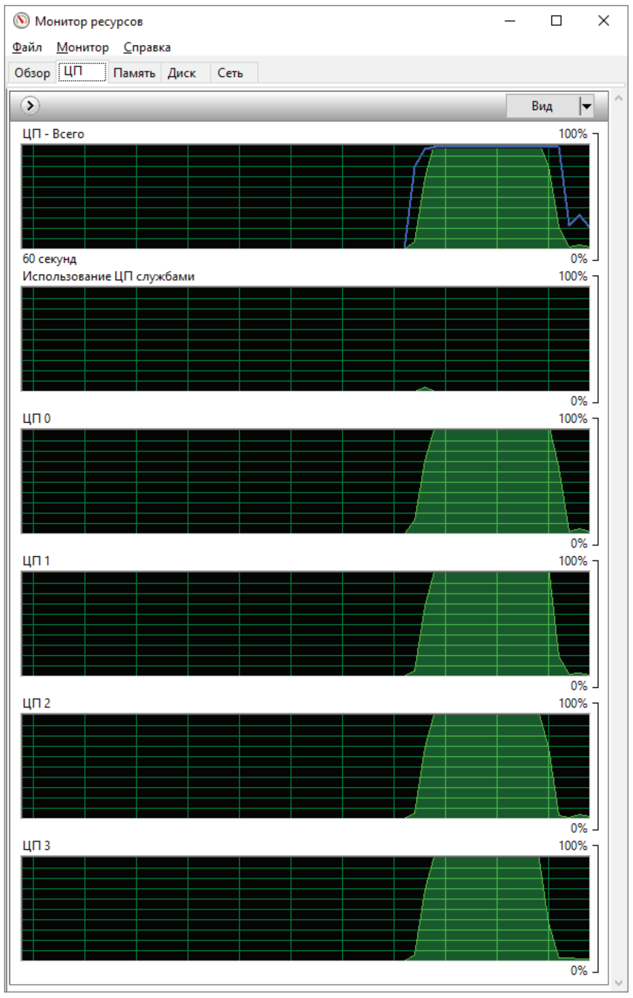
In Project No. 2, let's create two parallel flows: 300 iterations on the 1st core and 300 iterations distributed between the 2–4th cores (Fig. 5, *a*). The second thread is distributed by the Windows automatic scheduler into three cycles of X_i iterations in each such that: $X_2+X_3+X_4=300$. The exact values of X_i are not defined, since it is unknown how the OS automatic scheduler will divide the flow between the three cores. But, with the monitoring data in Fig. 4, *b* it can be seen that the OS automatic scheduler tries to load the cores 2–4th evenly. Thus, the algorithm implemented in Project No. 1 is confirmed to simulate the actions of the OS automatic scheduler in the distribution of tasks between the processor cores.

In Project No. 3, an algorithm has been implemented, along which one serial thread of 600 iterations on the 1st core is created (Fig. 6). Analysis of the results of this algorithm is important, as in the 4-core single-chip processors on the basis of which UAV computing systems are built (most often these are different models of ARM Cortex processors), there is no support for Hyper-Threading, as in Intel processors. Therefore, if do not separate the tasks into threads, as suggested in this paper, for example, for the ARM Cortex A9 processor it is possible in Asymmetric Multiprocessing (AMP) mode to get all the load on the 1st core (Fig. 6, *a*) [2]. Then the timing of the task for Project No. 3 will increase almost threefold compared to the benchmark of Project No. 1 (Table 1).

Project No. 4 implements an algorithm according to which the 1st core is not involved, and to the 2–4th cores 3 parallel threads are made for 200 iterations each (Fig. 7, *a*). But, according to the monitoring of resources on the 1st core, activity is registered, it can be explained by processing the background OS processes (Fig. 7, *b*). That is why, the proposed algorithm for manual binding of a test task to 2–4th cores can be considered optimal. In this case, the 1st core is unloaded in a fast way will be able to work out the tasks associated with the critical application of the UAV. For example, adjusting the route when a mechanical obstacle appears on the path of the UAV, switching to optical navigation when it is impossible to obtain GPS coordinates, and the like. As a test task of the Project No. 4 in this case, one can consider the compression of the received video stream, the preparation of data for transmission using the communication module with the UAV to the ground command center or to the main drone.

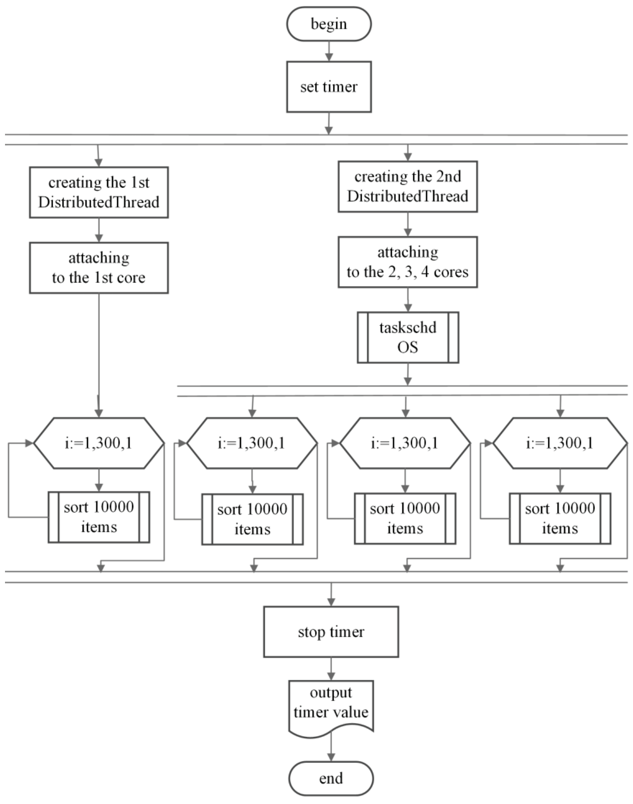

begin
set timer
creating the 1st DistributedThread
creating the 2nd DistributedThread
creating the 3rd DistributedThread
creating the 4th DistributedThread
attaching to the 1st core
attaching to the 2nd core
attaching to the 3rd core
attaching to the 4th core
i:=1,150,1
i:=151,300,1
i:=301,450,1
i:=451,600,1
sort 10000 items
sort 10000 items
sort 10000 items
sort 10000 items
stop timer
output timer value
end

a

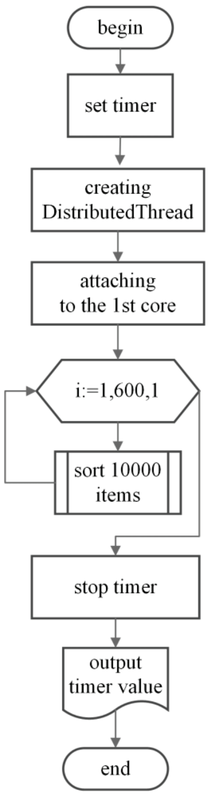


b

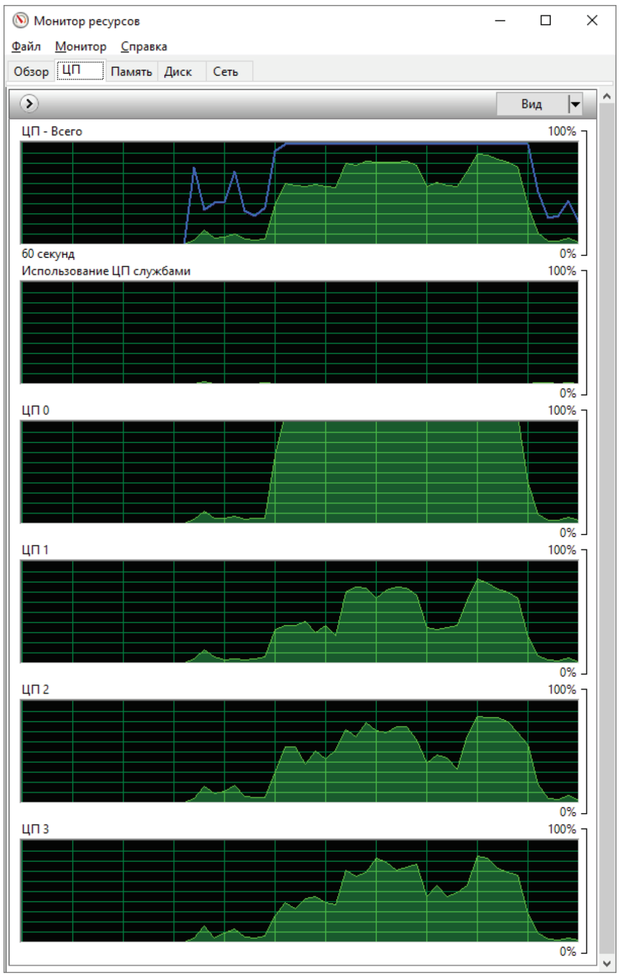
Fig. 4. CPU use for Project No. 1: *a* – block diagram; *b* – oscillogram


begin
set timer
creating the 1st DistributedThread
attaching to the 1st core
creating the 2nd DistributedThread
attaching to the 2, 3, 4 cores
taskschd OS
i:=1,300,1
sort 10000 items
stop timer
output timer value
end

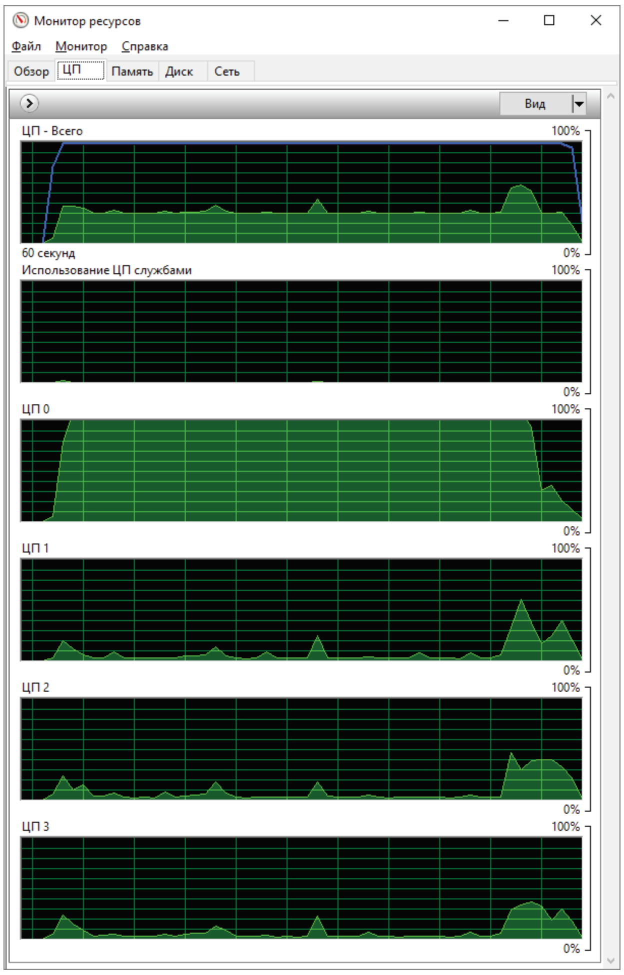
a



a



b

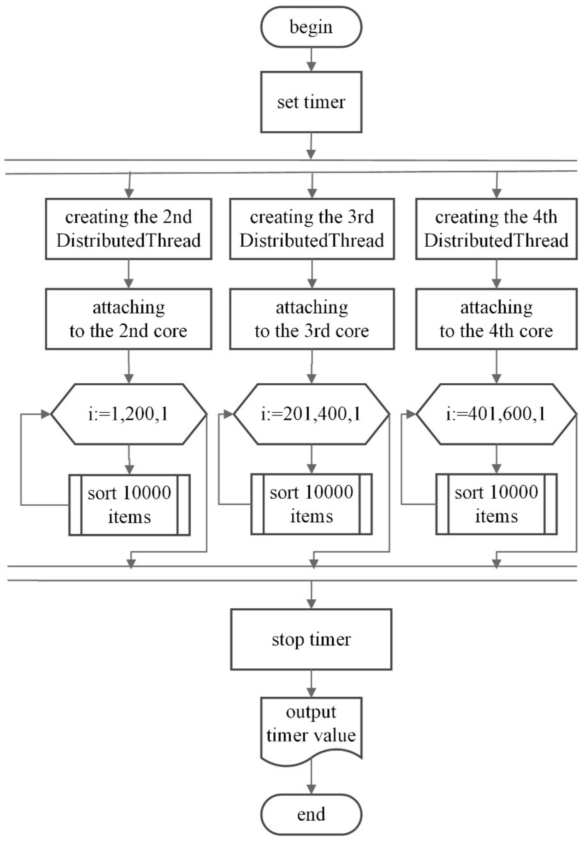


b

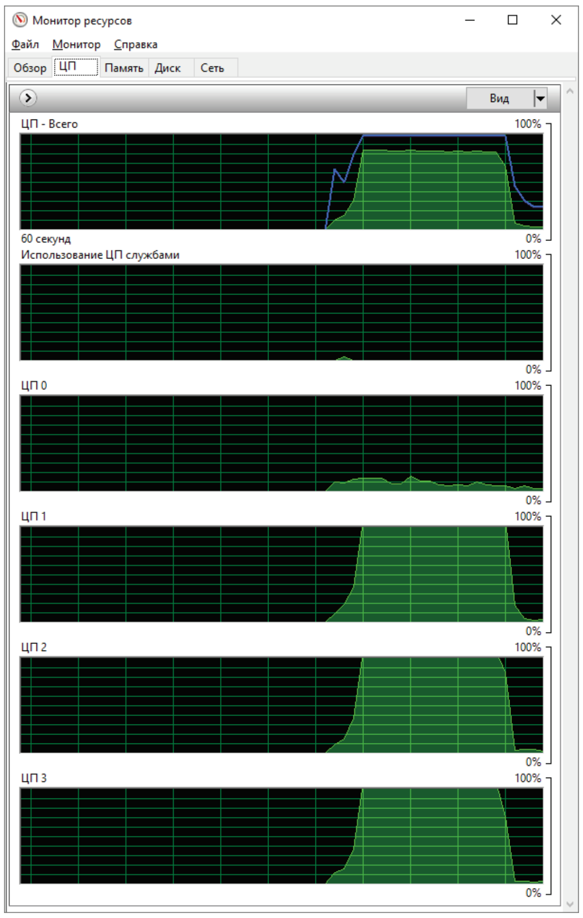
Fig. 5. CPU use for Project No. 2: *a* – block diagram; *b* – oscillogram

Fig. 6. CPU use for Project No. 3: *a* – block diagram; *b* – oscillogram

ISSN 2226-3780


begin
set timer
creating the 2nd DistributedThread
creating the 3rd DistributedThread
creating the 4th DistributedThread
attaching to the 2nd core
attaching to the 3rd core
attaching to the 4th core
i:=1,200,1
i:=201,400,1
i:=401,600,1
sort 10000 items
sort 10000 items
sort 10000 items
stop timer
output timer value
end

a



b

Fig. 7. CPU use for Project No. 4: *a* – block diagram; *b* – oscillogram

Table 1

Benchmarking results for four projects

Indicator	Project No. 1	Project No. 2	Project No. 3	Project No. 4	Used software
Timing (Average Time), s	12.7	24.6	47.2	16.5	Own development
Power Consumption (Average Power), W	58.5	38.9	27.3	49.3	HWMonitor v.1.33.0
Temperature, °C	75	62	62	80	HWMonitor v.1.33.0
CPU utilization, %	100	56	31	77	HWMonitor v.1.33.0
Fan's number of rotations, RPM	1900	1560	1505	1599	HWMonitor v.1.33.0

When benchmarking, a discrepancy of the indicators is possible, which is due to the fact that monitoring programs read the readings of the sensors every 2, 3 or 5 seconds. This may not coincide with the timings of test projects.

According to the monitoring of resources (Table 1), obtained by the HWMonitor, it is possible to significantly reduce the temperature at the 1st core by manually redistributing the problem between the processor cores for Project No. 4 (Fig. 8). Almost 13 % compared to all other projects. Due to this, when starting a new OS background process, which can be associated with a critical application of the UAV (loss of GPS coordinates, inclusion of the optical navigation task, instant recalculation of the course due to a gust, etc.), the use efficiency of the CS is greatly improved.

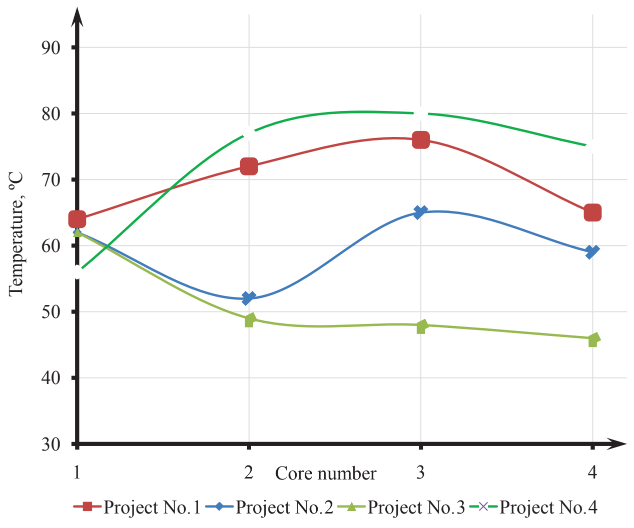


Fig. 8. The temperature of a single-chip processor with cores details

In spite of the fact that the algorithm of Project No. 4 proved to be optimal for benchmarking, we will explain the program code of the proposed algorithm in more detail.

First, let's connect the namespace in which the *DistributedThread* class, developed by the authors, is declared:

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

namespace ConsoleApplication1{
    class Program{
```

The next step is to set the timers and size of the array:

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

The *CountdownEvent* class, which is used in the following code, is necessary to synchronize and unblock threads waiting in the queue.

In this case, the main code will not be executed until it receives a certain number of signals (in our case, three).

Then create an object of the *DistributedThread* class, bind it to the 2nd core, and start it. To adhere to the logic of actions, we call the variable *thread2* (and not *thread1*). Such notation provides clarity to which core the binding is made to:

```
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();
```

Creating and running threads *thread3* and *thread4* are similar to *thread2*, so the code is omitted. After the completion of all three threads on the console, let's display the timer value.

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

Below is a diagram that clearly demonstrates the increase in power consumption depending on the number of involved cores (Fig. 9).

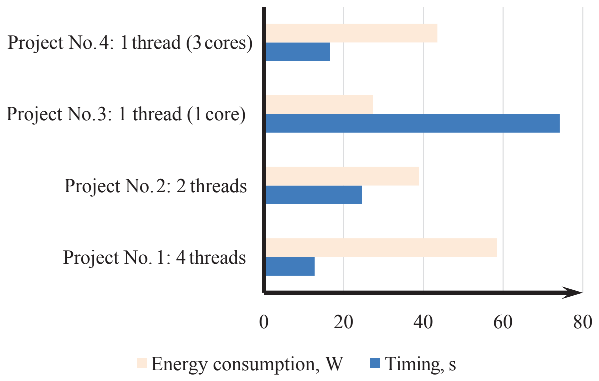


Fig. 9. Power consumption in a single-chip computing processor depending on the number of loaded threads

A 4-thread approach is most effective with timing, however, it is also the least energy-efficient. Therefore, the most effective use of UAV computing resources will ensure distribution according to the algorithm of project No. 4 of tasks not related to navigation and with the control of drones.

7. SWOT analysis of research results

Strengths. When using a UAV, everyone is faced with the need to have a stable system, especially when the UAV can be used critically. Therefore, it is extremely important that the UAV computer system immediately learns the necessary course changes when obstacles appear, when communications with the operator fail, with a sudden drop in the battery charge, etc. In this case, the algorithms for unloading the 1st core, on which all the system processes are normally performed, from calculation of monitoring and functional tasks can bring undoubted benefits. In addition, the additional power supply of the UAV CS from the energy of the measuring signal is suggested, so as not to disconnect, for example, the communication modules, as provided by the UAV control systems to save battery power. The proposed approach in the complex significantly improves the efficiency of using UAVs based on a multi-core calculator and improves the conditions for its interaction with the ground command center.

Weaknesses. The weak side of the research is the lack of analysis of energy costs when performing various tasks in quality with the use of the proposed dispatch algorithm in comparison with the automatic scheduler of several CPU models. Full automation of the on-board dispatching is complicated by the fact that each algorithm runs autonomously, that is, there is no choice between modes or better timing, or the lowest power consumption.

Opportunities. In the long term, it is expedient to carry out a study of the effect of the distribution of computations between CPU cores for UAVs based on various models of single-chip processors. In addition, it would be very useful to numerically determine the fate of the power consumption, which can be compensated by the additional supply of the CS from the energy of the measuring signal for various sets of sensors on board the UAV.

Threats. In the field of creating new UAV models, each manufacturer creates its own proprietary software for controlling the cursor and computing processes in the UAV CS. Therefore, the algorithmic and software development has very few chances to enter the world market. Its niche is the use in individual enterprises, which are not related to the economic areas of UAV applications, but to critical situations (monitoring emissions of toxic substances into the atmosphere, search for injured people or man-made threats, etc.). It is now difficult to predict the negative risks of the developed approach. But it is possible to say that additional costs are not needed to use the proposed approach in specific production situations.

8. Conclusions

1. Ten most common physical effects on the primary power converters of the measuring signal from UAV sensors are analyzed. It is established that piezoelectric sensors are the most energetically attractive for providing additional power to the UAV modules.

2. The model of interaction of UAV units as an information-measuring system is constructed. It is determined that the continuous IMS operation will be provided that the addition of energy to the system will be greater than it is spent on all nodes of the circuit.

3. As a result of the analysis of the mechanism of automatic schedulers of OS tasks it is established that under these conditions the processes are distributed among all four CPU cores. But, in that case, the 1st core has no reserves in comparison with other cores by loading (loaded by 100 %) and temperature (there is a level with the other involved cores). The number of revolutions of the cooler and the power consumption of the processor in the automatic scheduler mode is the highest. Therefore, in the case of a critical situation, when it is required that there may be a significant increase in computations on the 1st core for UAV control system tasks, such tasks can queue on previously started tasks. From the point of view of life support of the UAV, this is unacceptable.

4. Studies have shown that parallelizing tasks that can be performed as multithreaded (video stream compression, data transmission over various communication channels, etc.), improves the efficiency of UAV usage. Binding processes to three cores instead of four by 23 % increases the total time of the task in comparison with the time of executing a similar task on one core.

5. The obtained results of the studies of the developed scheduling algorithms for tasks on a 4-core processor makes it possible to define as the optimal algorithm for reserving the resources of the first core of a multi-core single-chip CPU for calculations of primary importance. The use of such algorithm of calculation contributes to the improvement in the efficiency of the use of UAVs. So, for example, for DJI Phantom 4, the distribution of tasks only for the 2–4th cores reduces power consumption by 10.3 % and provides an increase in the flight time by 3.1 minutes. Even with a non-maximal flight speed of this UAV model at 50 km/h (P mode) [21], this will increase the range of professional aerial photography by 1.3 km, taking into account the return to the base.

Acknowledgements

The work is supported by the Ministry of Education and Science of Ukraine within the state budget research project of the Petro Mohyla Black Sea National University on the topics «Development of Polymetric Sensors with Powerupply from Measuring Signal Energy for Information and Measurement Systems» (State Registration No. 0115U000316, 2015–2016) and «Development of Wireless Non-volatile Information and Measuring Networks of Critical Applications of Military and Civil Use» (State Registration No. 0117U000447, 2017–2018).

References

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

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ БЕСПИЛОТНЫХ ЛЕТАТЕЛЬНЫХ АППАРАТОВ ЗА СЧЕТ РАСПРЕДЕЛЕНИЯ ЗАДАЧ МЕЖДУ ЯДРАМИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССОРА

Рассмотрены алгоритмы распараллеливания задач между ядрами 4-ядерных процессоров вычислительных систем беспилотных летательных аппаратов (БПЛА). Показано, что высвобождение 1-го ядра и распределение задачи между 2–4-м ядрами на 10,3 % уменьшает энергопотребление процессора. Применение разработанного алгоритма в комплексе с предложенным дополнительным питанием от энергии измерительного сигнала датчиков повышает эффективность использования БПЛА.

Ключевые слова: вычислительные системы беспилотных летательных аппаратов (БПЛА), 4-ядерный процессор, имитационное моделирование.

Zhuravska Iryna, PhD, Associate Professor, Department of Computer Engineering, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, e-mail: irina.zhuravska@chmnu.edu.ua, ORCID: http://orcid.org/0000-0002-8102-9854

Borovlyova Svitlana, Senior Lecturer, Department of Software Engineering, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, e-mail: svetlana.borovlyova@chmnu.edu.ua, ORCID: http://orcid.org/0000-0003-1994-0556

Kostyria Mykhailo, Department of Intelligent Information Systems, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, e-mail: assassin19741@gmail.com, ORCID: http://orcid.org/0000-0001-9537-6374

Koretska Oleksandra, Postgraduate Student, Department of Computer Engineering, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, e-mail: alex.koretska@chmnu.edu.ua, ORCID: http://orcid.org/0000-0002-1240-1472

UDC 519.853.33
DOI: 10.15587/2312-8372.2017.118338

Raskin L., Sira O., Parfeniuk Y.

ANALYSIS AND DEVELOPMENT OF COMPROMISE SOLUTIONS IN MULTICRITERIA TRANSPORT TASKS

Розглянуто метод розв'язання багатокритеріальних транспортних завдань. Запропонована ітераційна процедура, в якій початковий план завдання є оптимальним за основним з критеріїв. На наступних ітераціях реалізується уступка за основним з критеріїв з метою поліпшення значення додаткових. Процедура триває до отримання компромісного рішення. Розглянуто приклади розв'язання задачі.

Ключові слова: *багатокритеріальна транспортна задача, ітераційне рішення, формування Парето-безлічі рішень.*

1. Introduction

In the practice of planning and organization of transportation of goods, two different mathematical models are traditionally used:

– the transport task by the cost criterion (at the same time the average total cost of transportation is minimized);

– the transport task by the time criterion (the maximum of the traffic durations is minimized).

These tasks are alternative in the sense that their optimal plans, as a rule, do not coincide (the shortest route is not necessarily the cheapest one). The technologies for solving these problems have been well worked out [1–3] and constructively take into account the different specifics and features of the productions of each of them. For this reason, they are fundamentally different and their integration into a single computational procedure is very problematic. At the same time, when solving the practical problems of transport logistics, there is a need to solve compromise problems, for example, such:

a) to find a transportation plan that minimizes the average total cost of transport, provided that the longest of them does not exceed the prescribed one;

b) to find a transportation plan that minimizes the maximum of the transportation duration, provided that their average total cost does not exceed the specified value.

It should be noted that when solving practical tasks of transportation planning, it is expedient to take into account one more criterion – the probability of successful implementation of the transportation plan for the aggregate of routes from suppliers to consumers, which are determined by the selected plan. The development of a method for solving this problem is of theoretical and practical interest.