11th International Conference on Electrical, Electronics and Computer Engineering (IcETRAN), Nis, 3-6 June 2024

Multithreaded programming in LabVIEW measuring applications

Josif Tomić
Faculty of Technical Sciences
Novi Sad, Serbia
tomicj@uns.ac.rs

Miodrag Kušljević
Termoelektro Enel AD
Belgrade, Serbia
miodrag.kusljevic@te-enel.rs

Nemanja Gazivoda
Faculty of Technical Sciences
Novi Sad, Serbia
nemanjagazivoda@uns.ac.rs

Platon Sovlilj
Faculty of Technical Sciences
Novi Sad, Serbia
platon@uns.ac.rs

***Abstract*—Modern measuring instruments are today software-oriented and rely entirely on information and computer technologies. Virtual instruments or, to be more precise, microprocessor-based measurement devices usually consist of a general purpose PC or Laptop computer and various types of measuring cards. Measurements are generally reduced to writing programs in different program packages, such as: LabVIEW, Matlab, and the like. For this reason, engineers and scientists are extremely interested in the high level of computer and information technology, as this will ensure that their measuring devices have superior performance. In this paper, methods that enable multithreaded programming in the LabVIEW program have been analysed and compared. Multithread programming allows parallel execution of tasks and significantly accelerates the data acquisition process in measuring applications.**

***Index Terms*—data acquisition, LabVIEW, measurement, multicore processing, multithreading.**

I. Introduction

For decades, the LabVIEW program package has been a basic tool for implementing measuring applications [1] and digital signal processing [2]. Program tasks can range from very simple to very complex, such as, for example, SCADA systems. To solve such complex tasks it is necessary to provide very complex computer and IT equipment. As computer hardware reaches its limit in the execution speed of instruction, engineers begin to realize more complex systems that would remind that some tasks are executed simultaneously in parallel. Thus, computers were created with multitasking and multithreading capabilities.

In computer architecture, multithreading is the capability of a CPU that has one or more cores (multi-core) to support multiple threads at the same time. It is understood that the operating system is capable of providing this kind of work. In a multithreaded application, the processes and threads share the resources of a single or multiple cores, which include the computing units, the CPU caches, and the Translation Lookaside Buffer (TLB).

Core is a hardware term that describes the number of independent central processing units in a single computing component. A thread is a software term for the basic ordered sequence of instructions that can be passed through or processed by a single CPU core.

A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce time and energy, to access data from the main memory. A cache is a smaller, faster memory, closer to a processor core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc.).

Memory organization in multicore computing systems affects communication overhead and LabVIEW program execution speed. Common memory architectures are shared memory, distributed memory, and hybrid shared-distributed memory. Shared memory systems use one large global memory space, accessible by all processors, providing fast communication. However, as more processors are connected to the same memory, a communication bottleneck between the processors and memory occurs. Multithreading extends the idea of multitasking into applications, so you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications but also among each thread within an application. Applications that take advantage of multithreading have numerous benefits, including the following: more efficient CPU use, better system reliability, and improved performance on computers [3].

II. Multicore Programming

A multiprocessor represents multiple physical processors, whereas a multicore processor represents multiple cores in a single physical processor. That means a multiprocessor can be built from a number of multicore processors. A multicore processor resides on a single socket.

Replacing a single-core computing system with a multicore system that uses processing units with clock speeds equal to that of the single-core system shortens LabVIEW program execution time. Ideally, program execution speed increases by a factor equal to the number of cores on the multicore computing system (for example, a four times speed increase on a quad-core system); however, communication overhead between threads and cores prevents ideal execution time improvement. If the LabVIEW program in question is completely sequential and running on a single processor, it needs less shared processor time with other software, leading to program execution time improvement. If the program in question is completely parallel and consists of tasks of equal size having no data dependency, you can achieve near ideal execution time improvement [4].

In order to achieve parallel execution in software, hardware must provide a platform that supports the simultaneous execution of multiple threads. Computer architectures can be classified by two different dimensions: a) number of instruction streams that a particular computer architecture may be able to process at a single point in time, b) number of data streams that can be processed at a single point in time.

The introduction of multicore processors provides a new challenge for software developers, who must now master the programming techniques necessary to capitalize on multicore processing potential. These programming techniques are task parallelism and data parallelism [5].

Task parallelism is simply the concurrent execution of independent tasks in software. Consider a single-core processor that is running two programs at the same time. Although these applications run on separate threads, they still ultimately share the same processor. On the dual-core machine, these two applications essentially can run independently of one another. Although they may share some resources that prevent them from running completely independently, the dual-core machine can handle the two parallel tasks more efficiently.

Data parallelism is a programming technique for splitting a large data set into smaller chunks that can be operated on in parallel. After the data has been processed, it is combined back into a single data set. With this technique, programmers can modify a process that typically would not be capable of utilizing multicore processing power, so that it can efficiently use all processing power available.

III. Amdahl's Law

Amdahl's law [6] is used to find the maximum expected improvement of the overall system, when only one part of the system is improved. It is often used in parallel processing to predict theoretical maximum acceleration when using multiple processors. The law was named after computer architect Gene Amdahl. The law was introduced by the American Federation of Information Processing Societies (AFIPS) at the Spring Joint Computer Conference in 1967.

Amdahl's law has formulation:

$$Speedup = \frac{1}{S + \left(\frac{1-S}{n}\right)} \qquad (1)$$

In this equation, S is the time spent executing the serial portion of the parallelized part of the program and n is the number of processor cores. Note that the numerator in the equation assumes that the program takes 1 unit of time to execute the best sequential algorithm.

To better understand Amdahl's law, let's go through a calculation example. The total time to execute a program is set to 1. The non-parallelizable part of the programs is 40% which out of a total time of 1 is equal to 0.4. The parallelizable part is thus equal to 1-0.4=0.6. The execution time of the program with a parallelization factor of 2 (2 threads or CPUs executing the parallelizable part, so *n* is 2) would be:

$$Speedup = \frac{1}{0.4 + \left(\frac{1-0.4}{2}\right)} = 1.42 \qquad (2)$$

Amdahl's Law is a statement of the maximum theoretical speed-up you can ever hope to achieve. The actual speed-ups are always less than the speed-up predicted by Amdahl's Law.

IV. Multithreading with LabVIEW

LabVIEW automatically divides each application into multiple execution threads. The complex tasks of thread management are transparently built into the LabVIEW execution system [7].

LabVIEW uses preemptive multithreading on OSs that offer this feature. LabVIEW also uses cooperative multithreading. OSs and processors with preemptive multithreading employ a limited number of threads, so in certain cases, these systems return to using cooperative multithreading. The execution system automatically creates multitask VIs using threads. However, a limited number of threads are available. For highly parallel applications, the execution system uses cooperative multitasking when available threads are busy. Also, the OS handles preemptive multitasking between the application and other tasks.

In traditional programming languages, the program must be divided into several independent threads for parallel execution. Each thread can work at the same time. However, there is a difference between writing code that is safe to run in a multithreaded application, and writing code that runs in parallel to increase the performance you received from the multicore system. This difference is often illustrated in the drivers or functions used when writing the program. Multithread-safe functions can be called from multiple threads and do not overwrite their data, which prevents conflicts from blocking execution. If one thread calls a function, all other threads trying to call this function must wait for the first thread to end. The reentrant functions go one step further by allowing multiple threads to call and execute the same function simultaneously, in parallel [8].

In a multithreaded LabVIEW program, an example application might be divided into four threads: a) data acquisition, b) mathematical analysis of signal, c) numerical and graphical presentation, and d) data storage of results on disc. You can prioritize each of these so that they operate independently. Thus, in multithreaded applications, multiple tasks can progress in parallel with other applications that are running on the system.

In many applications, a program calls measurement instruments. These instrument calls often take a long time to complete. In a single-threaded application, a synchronous call effectively blocks, or prevents, any other task within the application from executing until the operation completes. Multithreading prevents this blocking. While the synchronous call runs on one thread, other parts of the program that do not depend on this call run on different threads. Execution of the application progresses instead of stalling until the synchronous call completes. In this way, a multithreaded application maximizes the efficiency of the CPU because it does not idle if any thread of the application is ready to run [9].

V. Parallel programming in LabVIEW

In principle, LabVIEW programs are initially capable of executing program parts in a completely parallel manner because they are based on a data flow and not a code flow. For example, if there are two independent For Loops, that are not connected to the wire, it is quite clear that they can be performed in parallel. When started, the LabVIEW compiler program immediately begins to work on identifying the parallel sections of the program. Once identified, these independent pieces are assigned to the thread that LabVIEW automatically creates. This multithread process is very important because it is automatically executed by the LabVIEW program, so even ordinary users without much programming knowledge can use this extraordinary technique [10].

Some techniques that are most commonly used for parallel execution of the program are: a) While Loops, b) Shift Registers, c) Parallel For Loops, d) Timed Loops, e) Queues, f) Pipelining, and g) CPU Pool Reservation.

VI. Experimental results of measurement

In order to verify the multithread capability of working in the LabVIEW program and how much it contributes to the speed of execution of measuring applications, four programs have been implemented using the multithread principle.

The following parameters of the electricity network are measured: signal amplitude, signal frequency, and harmonic spectrum of the voltage signal. In addition, all the sampled values were recorded in a text file on a computer disk.

Measuring equipment consists of a voltage measuring transformer, a USB measuring card USB6210 (16 AI, 16-bit ADC, 250 kS/s), and a PC desktop computer that has an Intel Core Processor i3-4170. The Processor Base Frequency is 3.7 MHz, and the processor has 2 cores.

The sampling rate of the voltage signal was 25600 Hz, and 4096 samples were sampled in the continuous mode. The time interval was 160 ms or 8 periods of 50 Hz signal.

To check the execution speed of these LabVIEW programs, the Query Performance Counter located in the kernel32.dll program library is used and called from the LabVIEW program via the Call Library Function Node. This counter is in the processor, and it works completely independently of it. It is considered to be the most accurate reference time in a PC. Windows provides APIs that you can use to acquire high-resolution time stamps or measure time intervals.

A. LabVIEW algorithm

In the first experiment, a program was developed that uses only a LabVIEW algorithm for automatic task recognition that can be performed in parallel. The program was implemented through three Flat Sequences. In the first sequence, the Query Performance Counter is started, and its value is taken. In the second Flat Sequence, the acquisition of the voltage signal is carried out and the frequency of the signal as well as the harmonic spectrum is calculated. At the front panel, these values are presented graphically and numerically. The numerical values of the signal are stored in a file on the computer disk. In the third Flat Sequence, a new current value of the Query Performance Counter is taken and the estimated time required for these tasks is calculated and this value is displayed on the front panel in numerical form. Program execution speed was measured over 100 measurements, and the arithmetic mean was 160.13 ms and the standard deviation was 0.91 ms.

The measurement results showed that this method is very good and that LabVIEW has a very high-quality algorithm for recognizing tasks that can be executed in parallel. Otherwise, in the literature, there is a very large number of papers dealing with this topic that describe various methods for the effective execution of multithread programs [11]-[13].

Fig. 1. shows the front panel and Fig. 2. shows the block diagram of the realized program.

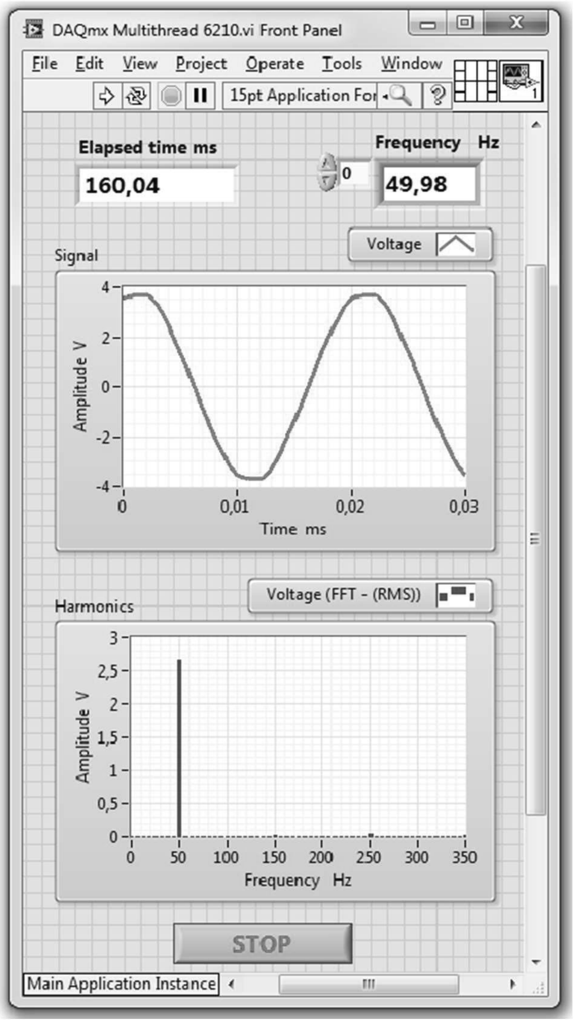

Fig. 1. Front panel of the realized program.

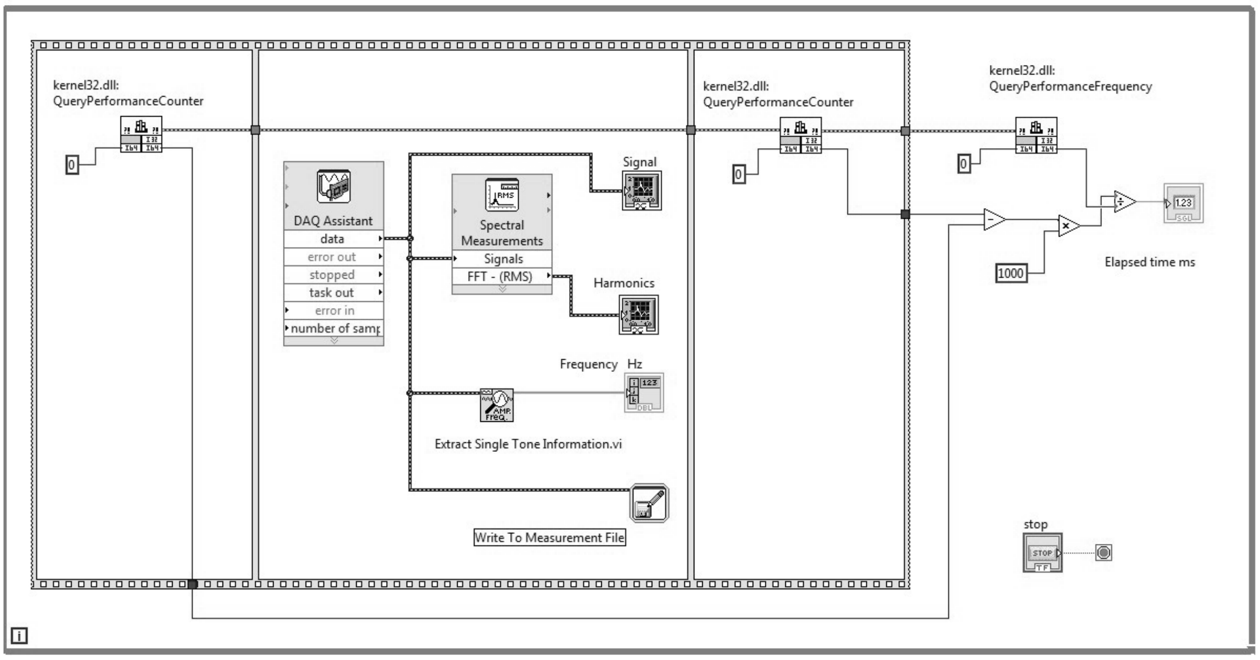
kernel32.dll: QueryPerformanceCounter
0
DAQ Assistant
data
error out
stopped
task out
error in
number of samp
Spectral Measurements
Signals
FFT - (RMS)
Signal
Harmonics
Frequency Hz
Extract Single Tone Information.vi
Write To Measurement File
kernel32.dll: QueryPerformanceCounter
0
kernel32.dll: QueryPerformanceFrequency
0
1000
Elapsed time ms
stop

Fig. 2. Blok diagram of realized program.

B. Parallel While Loops

While Loops are a fundamental structure that can be used with a variety of programming patterns (task parallelism, data parallelism, pipelining, or structured grid). These programs are often used in combination with Shift Registers. The programs are very easy to implement, and the results of the measurements have shown that they have excellent performance. With While Loop, you can greatly speed up Hardware-in-the-Loop Applications (HIL). Program execution speed was measured over 100 measurements, and the arithmetic mean was 160.11 ms and the standard deviation was 0.87 ms. Fig. 3. shows a program that uses the Parallel While Loop and Shift Registers for parallel execution of the program.

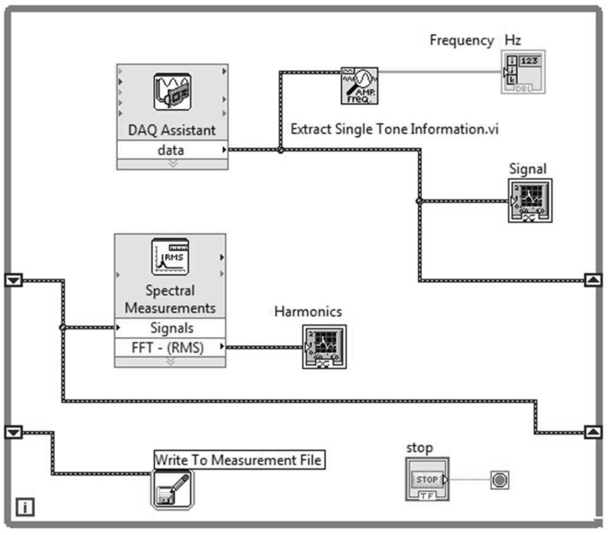

Fig. 3. Block diagram of Parallel While Loop.

C. Parallel For Loops

The Parallel For Loops are a valid approach for intensive operations that need to execute over and over in a loop, without having dependencies. When parallel iterations are allowed on a For loop, LabVIEW executes chunks concurrently. By default, LabVIEW schedules chunks by size from larger to smaller. Program execution speed was measured over 100 measurements, and the arithmetic mean was 160.19 ms and the standard deviation was 0.94 ms. Fig. 4. shows a program that uses For Loop for parallel execution of the program.

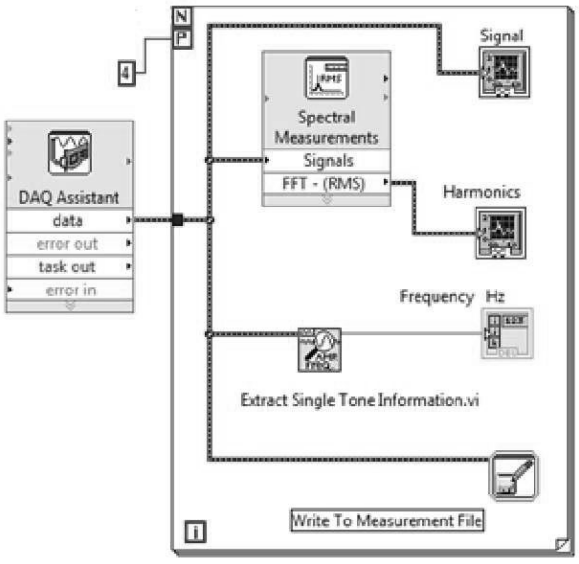

Fig. 4. Block diagram of Parallel For Loops.

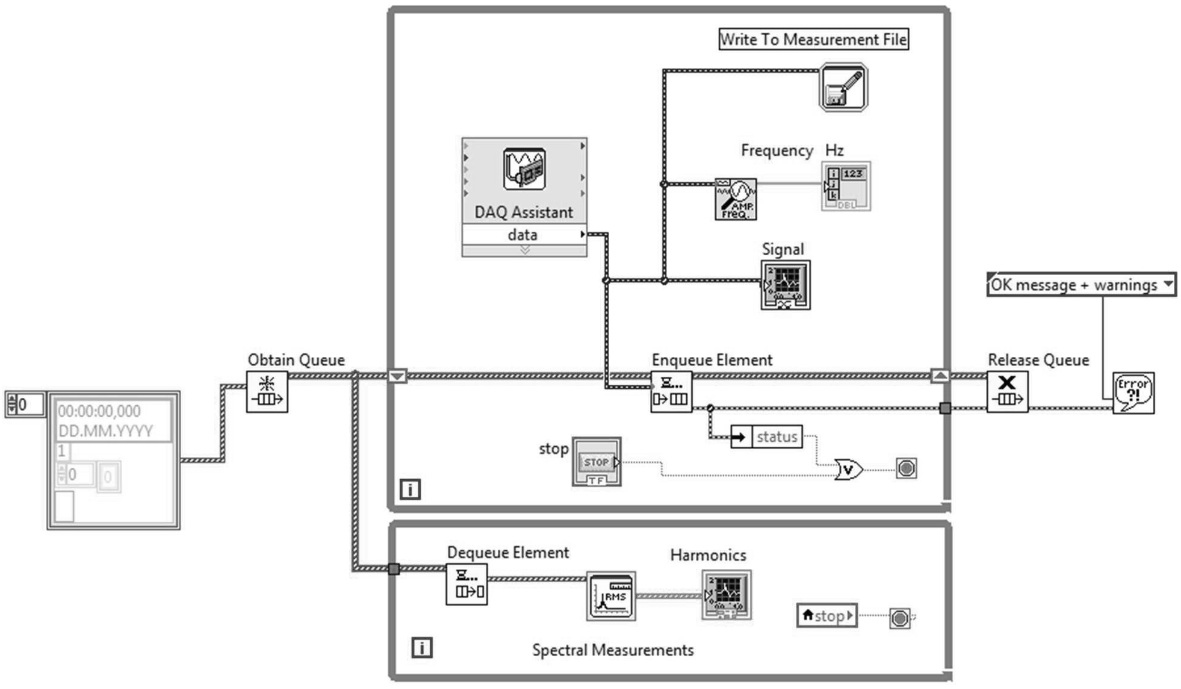

Fig. 5. Block diagram of Queues structure.

D. Queues structure

Queues are used for synchronizing data between multiple loops, and they are most used for the implementation of program-type producer-consumer architecture. This architecture is a general purpose and is not directly intended for parallel programming. Fig. 5. presents this algorithm. Unfortunately, practical measurements have shown that this method has many problems with the synchronization of multiple threads. Data Flow Parallelism - Pipelining can be used with Queues and with While Loop structures. Program execution speed was measured over 100 measurements, and the arithmetic mean was 160.15 ms and the standard deviation was 0.89 ms. Fig. 6. shows the Pipelining execution of multithread operations.

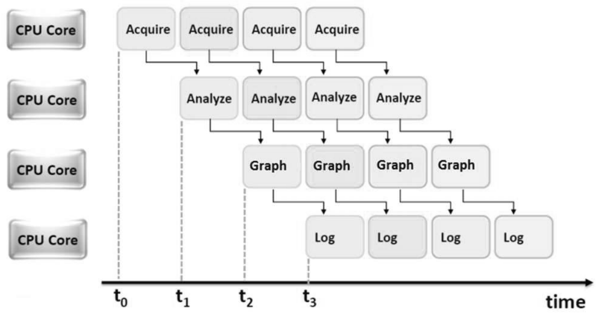

Fig. 6. Pipelining strategy.

When programming using multithread, attention should be paid to some problems, such as: thread synchronization, race conditions, deadlocks, shared resources, and data transfer between processor cores. In general, the CPU and data bus operate most efficiently when processing large blocks of data.

VII. CONCLUSION

In this paper, four methods of multithread programming in the LabVIEW program have been practically implemented and tested. These methods allow parallel execution of tasks in measurement-based applications. Based on the obtained measurement results, it can be seen that the LabVIEW program has a very powerful tool for parallel execution of tasks, which makes it probably the best program for measuring and acquiring signals.

ACKNOWLEDGMENT

This research has been supported by the Ministry of Science, Technological Development, and Innovation (Contract No. 451-03-65/2024-03/200156) and the Faculty of Technical Sciences, University of Novi Sad through the project "Scientific and Artistic Research Work of Researchers in Teaching and Associate Positions at the Faculty of Technical Sciences, University of Novi Sad" (No. 01-3394/1).

REFERENCES

[1] R. King, *Introduction to Data Acquisition with LabView*, McGraw-Hill Higher Education, New York, USA, 2012.

[2] N. Kehtarnavaz, N. Kim, *Digital Signal Processing System-Level Design Using LabVIEW*, Elsevier Inc., New York, USA, 2005.

[3] National Instruments, *Archived: Multicore Programming with LabVIEW,* [Online]. Available: https://www.ni.com/en/ support/documentation/ supplemental/07/multicore-programming-with-labview.html

[4] National Instruments, *Multicore Programming with LabVIEW Technical Resource Guide,* [Online]. Available: https://download.ni.com/ evaluation/labview/ekit/ multicore_programming_resource_guide.pdf

[5] G. Tan, N. Sun, G. Gao, "Improving Performance of Dynamic Programming via Parallelism and Locality on Multicore Architectures", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 20, Issue: 2, pp. 261-274, Feb. 2009.

[6] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", *AFIPS Conference Proceedings* (30), pp. 483–485. 1967.

[7] National Instruments, *Differences Between Multithreading and Multitasking for Programmers,* [Online]. Available: https://www.ni.com/en/support/ documentation/ supplemental/07 /differences-between- multithreading-and-multitasking-for-programm.html.

[8] National Instruments*, Using LabVIEW to Create Multithreaded DAQ Applications,* [Online]. Available: https://www.ni.com/en/support/ documentation/ supplemental/06/using-labview-to-create-multithreaded-daq-applications.html.

[9] R. Bitter, T. Mohiuddin, M. Nawrocki, *LabVIEW Advanced Programming Techniques*, Taylor & Francis Group, New York, USA, 2007.

[10] N. Dorst, "Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability", *National Instruments - Application Note 114*, July 2000. [Online]. Available: http://labview360.com/document/an/pdf/an114.pdf

[11] N. Li, G. Gong, X. Peng, Z. Chen, "Scene Matching Algorithm Evaluation Based on Multi-core Parallel Computing Technology", *WRI World Congress on Software Engineering*, Xiamen, China, 19-21 May, 2009.

[12] P. Bilski, W. Winiecki, "Analysis of the Time Efficiency Assessment in the Virtual Measurement Systems", *IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, Rende (Cosenza), Italy, 21-23 September, 2009.

[13] W. Winiecki, P. Bilski, "Multi-Core Programming Approach in the Real-Time Virtual Instrumentation", *IEEE International Instrumentation and Measurement Technology Conference*, Victoria, Vancouver Island, Canada, 12-15 May, 2008.