www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2015, VOL. 61, NO. 4, PP. 357–363
Manuscript received October 15, 2015; revised November, 2015. DOI: 10.1515/eletel-2015-0047

Passive Radar Parallel Processing Using General-Purpose Computing on Graphics Processing Units

Karolina Szczepankiewicz, Mateusz Malanowski and Micha Szczepankiewicz

***Abstract*—In the paper an implementation of signal processing chain for a passive radar is presented. The passive radar which was developed at the Warsaw University of Technology, uses FM radio and DVB-T television transmitters as ”illuminators of opportunity”. As the computational load associated with passive radar processing is very high, NVIDIA CUDA technology has been employed for effective implementation using parallel processing. The paper contains the description of the algorithms implementation and the performance results analysis.**

***Keywords*—PCL, Passive Coherent Location, Paralell Implementation, NVIDIA CUDA**

I. Introduction

PASSIVE coherent location (PCL) radars are a special case of a bistatic radars that exploit third-party transmitters as their sources of signal which are referred to ”illuminators of opportunity”. Commercial transmitters make PCL radar low cost and undetectable due to the lack of own signal emission. Radar compares the reference signal with the echoes reflected from targets by means of the crossambiguity function. The numerical analysis of the crossambiguity gives the opportunity to detect target reflections and estimate their parameters. In general, passive radars can make use of analog, digital television and radio signals, as well as cellular phone base stations, GPS satellites and others [1].

PCL system developed at the Warsaw University of Technology makes use of digital television (DVB-T) and radio (FM) signals in order to locate airborne objects. It is focused on constructing passive radar that works in real time. Several attempts have been made so far including parallel implementation of Passive Radar Demonstrator (PaRaDe) presented in [2] which utilizes FM transmitters as the source of the signal. Processing in passive radar systems is more complex and demanding in terms of computing power than in classic active radars. Therefore, we refer to recent technologies providing parallel computation capabilities to achieve required performance. This paper is an extension of work presented in [3].

This is the extended version of the ”Effective implementation of passive radar algorithms using General-Purpose computing on Graphics Processing Units” presented in Signal Processing Symposium, Debe, Poland, June 10 - 12, 2015 [3].

K. Szczepankiewicz is with the Institute of Computer Science, Warsaw University of Technology, Poland (e-mail: k.krej@stud.elka.pw.edu.pl).

M. Malanowski is with the Institute of Electronic Systems, Warsaw University of Technology, Poland (e-mail: m.malanowski@elka.pw.edu.pl).

M. Szczepankiewicz is with the Institute of Computer Science, Warsaw University of Technology, Poland (e-mail: m.szczepankiewicz@stud.elka.pw.edu.pl).

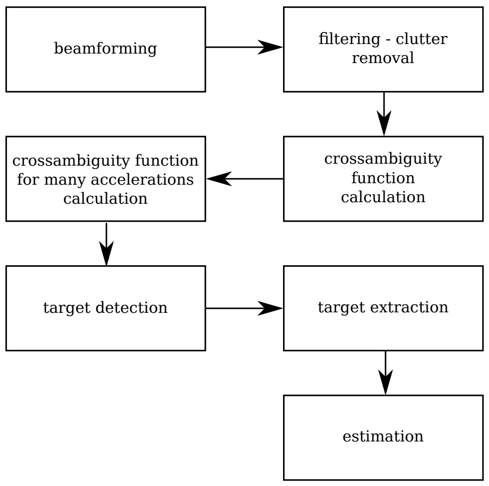
beamforming
filtering - clutter removal
crossambiguity function calculation
crossambiguity function for many accelerations calculation
target detection
target extraction
estimation

Fig. 1. Passive radar data processing pipeline

Driven by the growing market demand for high performance computing and real time processing, Graphic Processor Units (GPUs) have evolved into highly parallel, multithreaded processors. One of the producers that offers graphic cards equipped with multi-core GPUs and programming toolkit for developing numerical processing applications is NVIDIA with technology called Compute Unified Device Architecture (CUDA). PCL system makes use of the NVIDIA CUDA technology to achieve better performance. In this paper, the computation time of signal processing algorithms tested in passive radar is presented and compared to Matlab implementation.

II. Radar Processing

Passive radar taken into consideration processes data that is digitized at the first stage of processing. Most of algorithms are performed digitally by the developed software. Consecutive stages of the process chain are shown in Fig. 1. They are arranged in the pipeline which means the output of the previous operation is the input of the next one. Data (input signal) is divided into blocks, the size of which is arbitrary. This paper focuses on data processing starting from receiving the digitized signal to achieving objects with estimated coordinates of bistatic range, velocity, acceleration and azimuth.

After data acquisition, reference and several echo signals are obtained through a beamforming algorithm. It is assumed that statistical properties of the desired signal, interference and antenna parameters are known. Beamforming coefficients are obtained by the calibration procedure [4]. Then several beams of the signal are obtained by the linear combination of input data and calculated coefficients.

Signal after beamforming still contains strong clutter component, which can mask reflected echoes of the targets. For that reason, the filtering procedure is done. It uses two filters:

- Adaptive lattice filter [5], [6] with short order.
- FFT (Fast Fourier Transform)-based clutter canceller for the best performance.

Both filters divide data into arbitrarily sized blocks. The lattice filter consists of vector operations: dot products, additions and multiplications, whereas FFT-based clutter removal uses mainly Fast Fourier Transform.

Afterwards, the crossambiguity function is calculated and the results are range-doppler matrix for all signal beams (3-dimensional structure). Likewise in filtering, signal is divided into small blocks and partial correlation is calculated. Signals are modulated and range profiles are obtained using FFT. For DVB-T signals an additional stretch procedure is executed [7]. Algorithm can work in two modes. In the first one, the size of the block is computed using only the parameters of the radar, such as bistatic range and bistatic velocity. This can result in block sizes which are not power of 2, which in turn can lead to slow calculation of the FFTs. In the second mode the size of blocks is assumed to be a power of 2. This usually means that the amount of data is increased, however, the time of FFT algorithm computation is reduced. Crossambiguity function is calculated for several acceleration values. For that purpose, Finite Impulse Response (FIR) filter has been used [1], [8], [9]. After the filtering stage, 4-dimensional matrix of range, doppler, acceleration and beam is obtained.

The 4-dimensional matrix from the previous stage is passed to the detection procedure. It uses the two dimensional Cell Averaging Constant False Alarm Rate (CACFAR) [10] operating in range-velocity dimensions to detect targets. Data from each acceleration and each beam are processed separately. Bins of the beam-acceleration-range-Doppler surface which exceed selected detection threshold are marked as containers of target echoes. These detections are passed to the extractor.

Extraction consists of copying target echoes from large four dimensional matrix to object containers. The individual detections are clustered together using image processing algorithms. As a result, single detections which are close to each other are combined into potential targets.

Potential objects are passed to the estimator module to determine precise targets parameters. As a result plots corresponding to detected targets are created. Further processing, which is out of scope of this paper, includes bistatic tracking, target localization and Cartesian tracking.

III. Matlab Implementation of Passive Radar

Presented radar processing was firstly implemented in the Matlab environment to check correctness and improve the quality of the algorithms in terms of target detection. An example of application performance is presented in Tab. I. In the experiment parameters typical for FM based radar were assumed: sampling frequency of $200\,\mathrm{kHz}$, integration time of $1\,\mathrm{s}$ and carrier frequency of $100\,\mathrm{MHz}$. Maximum radar range in the crossambiguity function was set to $400\,\mathrm{km}$ and maximum velocity was set to $1000\,\mathrm{m/s}$.

The obtained results clearly show that Matlab implementation is not suitable for real-time operations. It is, however, a good starting point for other implementations, as it may serve as a reference solution for assessing numerical correctness of alternative implementations. Among many possibilities of different languages and hardware platforms the CUDA GPU technology has been selected for further development.

TABLE I
Time Results of Processing Stages for FM Signal (Matlab Implementation)

Algorithm (FM)	Processing time (Matlab)
Beamforming	170 ms
Filtering	5000 ms
Crossambiguity	40000 ms
CACFAR	7000 ms
Extraction	200000 ms

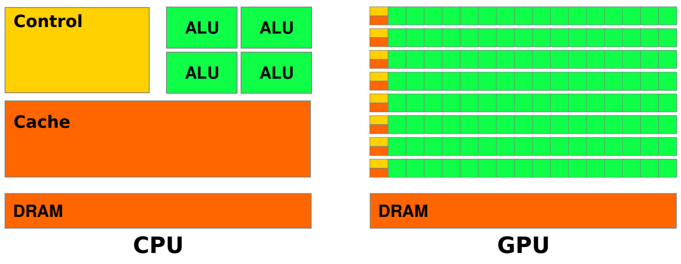

Fig. 2. GPU and CPU architecture comparison, Fig. 3 in [11]

IV. NVIDIA CUDA Technology

CUDA is a parallel computing platform released by NVIDIA, the programming model of which allows programmers to access existing parallel computational elements in graphics cards. Not only can Graphical Processing Units (GPUs) be used for graphic calculations but also for general purpose computing. GPUs differ from Central Processing Units (CPUs) in more transistors dedicated to data processing than controlling activities, which is presented in Fig. 2. Therefore, different programming approach has to be used [12]. Moreover, the application development may be done using CUDA libraries e.g. CUDA C language. Furthermore, highly optimized tools such as cuBLAS (for numerical computation) and cuFFT (parallel implementation of FFT algorithm) libraries facilitate creating effective processing applications. Constantly enhanced computing environment makes this technology attractive for radar processing engineers.

The devices used at the Warsaw University of Technology for passive radar processing have compute capability [11] more than 2.0, such as GTX Titan (Fig. 3) with 2688 CUDA cores, base clock of $837\,\mathrm{MHz}$ and memory bandwidth of $288.4\,\mathrm{GB/s}$ [13].

In the following section, parallel realization of passive radar algorithms is presented. Described stages of processing are performed using single precision floating point numbers.

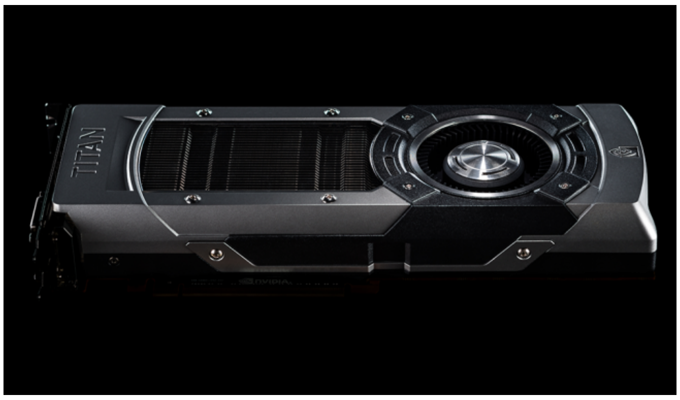

Fig. 3. Geforce GTX Titan, Fig. in [13]

V. Parallel Realization of Passive Radar Algorithms

Overall performance of developed solution depends on the configuration of CUDA blocks dimensions [14], [15]. However, it is strictly connected to the architecture of a graphic card and requires separate tuning.

During the process of implementation several problems were encountered. Large amount of input and processing data limited possible optimization techniques. In some cases, data layout could not be properly padded. Amount of graphics card memory appeared to be the bottleneck of the implementation for more demanding DVB-T signals.

A. Beamforming

The beamforming procedure is realized simply by multiplying the matrix of the input signals by the matrix of fixed beamforming coefficients. As a result, the signal corresponding to a digitally formed beam is a linear combination of the input signals. One CUDA function (*kernel*) is used for additional scaling the coefficients and data transposition, but since the number of coefficients is very low, this operation is irrelevant as far as computational time is concerned. The linear combination was firstly realized with the usage of cuBLAS library function *cublasCgemm* [16]. However, the results of the profiling tool (NVIDIA Visual profiler) showed that the coefficient of GPU card utilization was at the level of 46.4%. The reason for this is that the routine from cuBLAS library used for calculations is optimized for multiplying large matrices. Input matrix for beamforming procedure is $N \times M$, where M is number of antennas and N is the number of signal samples in a processed block. The coefficient matrix size is $M \times L$, where L is the number of beams and usually $M, L \ll N$. Therefore, beamforming can be decomposed to many tasks of small matrices multiplications as shown in Fig. 4. This kind of multiplication can be performed by *cublasCgemmBatched* routine from cuBLAS library. This solution was not satisfactory either, because internal matrix division performed by the cuBLAS routine did not provide any growth in efficiency. On the other hand, the result achieved for undivided input data matrix was about $31,33\%$ faster than initial solution. Taking that all into consideration, a

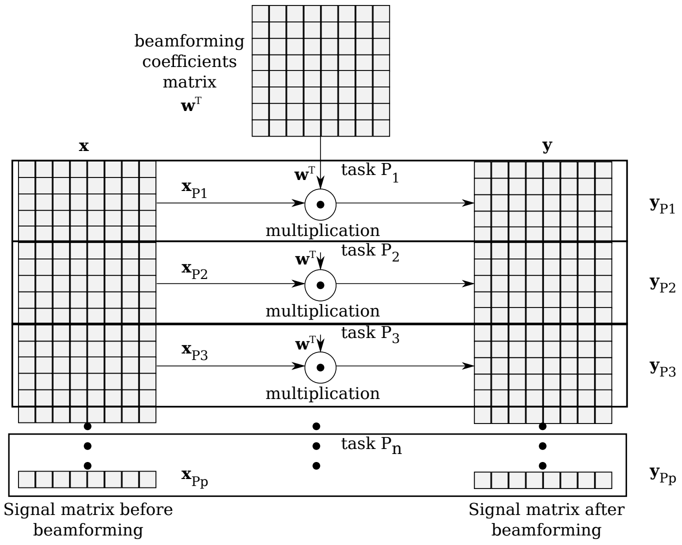

Fig. 4. Beamforming matrix multiplication decomposition to many tasks

special kernel was implemented for multiplying matrices in the beamforming procedure. Designed algorithm performed $65,50\%$ faster than the initial solution and achieved 91.9% GPU utilization due to significant reduction of used registers by each block of threads.

B. Filtering – Clutter Removal

Formed beams of the signal are passed to the filtering procedure. FFT-based clutter removal algorithm uses several CUDA kernels for signal shifting, modulation and correlation calculation. During correlation procedure blocks are processed simultaneously. For inner product calculation, the schema known as ”map and reduce” is used (Fig. 5). Each number from the signal is mapped into one CUDA thread. A block of threads realizes summation with the usage of on-chip shared memory which is faster than global RAM memory. Results from multiple blocks are gathered (reduced) through atomic addition CUDA instructions on global memory to achieve as much parallelism as possible. FFT is proceeded by highly optimized cuFFT library [17] for many batches with overlay at once.

Adaptive lattice filtering is performed when a low order filter is desired. The algorithm consists of two parts: a lattice predictor and a delay line [2]. The signal is divided into blocks. Partial correlation and norm calculation is implemented for all blocks simultaneously in terms of the map-reduce procedure described above.

C. Crossambiguity Function Calculation

The correlation function for small blocks is calculated by performing FFT/multiplication/IFFT consecutively. Results of multiplications are obtained by a simple kernel function. Each number in the signal matrix is mapped into one CUDA thread. CUDA cuFFT library is used to compute FFT of the signal divided into blocks. The procedure consists of several CUDA kernel functions used to modulate and normalize the signal after inverse FFT. Afterwards, the FIR filter implemented in

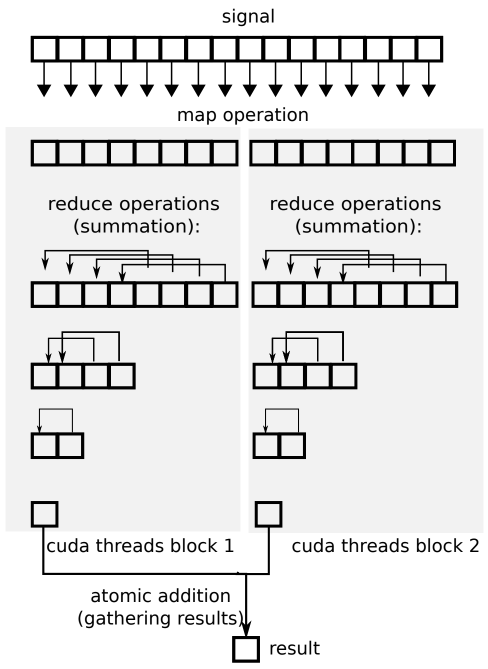
signal
map operation
reduce operations (summation):
reduce operations (summation):
cuda threads block 1
cuda threads block 2
atomic addition (gathering results)
result

Fig. 5. ”Map-reduce” operations

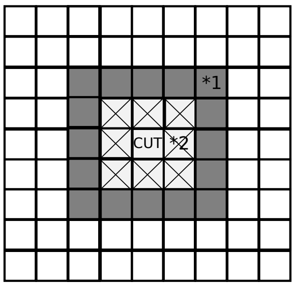

Fig. 6. The schema of the two dimensional CACFAR window. CUT - cell under test, *1 - window cells, *2 - guard cells

the CUDA kernel function is used to determine the crossambiguity function results for different accelerations. During optimization steps several changes have been made. The crossambiguity function was firstly computed for all beams simultaneously, but optimization process has shown that better GPU utilization is achieved when one thread computes more output values. One of the optimization techniques was also changing the dimensionality of CUDA blocks and separating the range and velocity dimensions.

D. Target Detection

In the detection procedure, 2-dimensional CACFAR algorithm is used (Fig. 6). A signal matrix is mapped to CUDA threads. Each thread from the block of threads loads a piece of data from 2D input chunk of the signal matrix into faster shared memory of the graphics processor. Then summation and threshold computation is done. Bins of the crossambiguity surface that exceed threshold are written as the result of the algorithm.

For better results the loop unrolling feature provided by the CUDA compiler is used. However, it demands fixed window

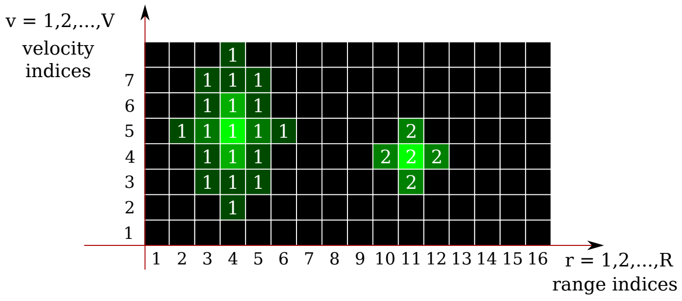

Fig. 7. Schema of the extraction procedure

size at compile time. In order to enable parameter change in the runtime phase, the detection function was implemented as a C++ language template. Such a solution is more flexible, as functions can be explicitly generated for desired window sizes at compile time, but provides slightly worse performance results.

Due to high computational requirements of the algorithm, additional detection procedure with a constant threshold was implemented in CUDA C language and can be used for more computationally demanding DVB-T signals. It consists of two kernels, the first of which computes the noise level. The second one indicates if the currently tested value is above the noise level and given detection threshold.

E. Target Extraction

The goal of the extraction is to select target echoes from the detection matrix which exceed the given threshold and connect all echoes that are to be considered as a single target. The schema of the extraction procedure for two targets (number 1 and number 2) is shown in figure 7. The green tone indicates the strength of the target echo. In order to achieve better time results, a simplified algorithm of the extraction that locates local maxima in detection matrix was used. The applied CUDA kernel function is designed to return only those results of the detection matrix that can be considered as target echoes and are local maxima in their neighbourhood. The result of the extraction is copied to CPU memory afterwards.

F. Estimation of Targets Parameters

The next step of the extraction is the estimation of target parameters. This part of the algorithm is proceeded by CPU and is implemented in C++ language. The main tasks of the estimation procedure is fitting the parabola in the crossambiguity matrix as shown in Fig. 8. However, further optimization steps led to the conclusion that connecting this processing stage to the extraction kernel provides better performance results. Estimated parameters are: bistatic range, velocity, acceleration and azimuth.

VI. Performance and Accuracy Results

Presented results were obtained using computer components listed below:

- Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50 GHz.
- 64 GB of DDR3 memory.

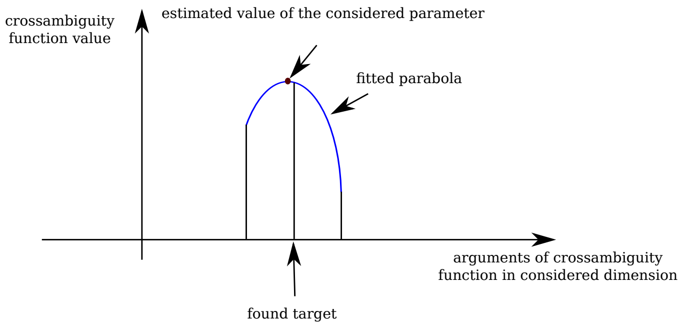
crossambiguity function value
estimated value of the considered parameter
fitted parabola
arguments of crossambiguity function in considered dimension
found target

Fig. 8. Fitting Parabola to Crossambiguity Function in One Dimension

- GeForce GTX 780 Ti, CC 3.5, 2880 cores, GPU clock: 928 MHz, memory clock: 3500 MHz, amount of global memory: 3072 MB.
- GeForce GTX TITAN, CC 3.0, 2688 cores, GPU clock: 876 MHz, memory clock: 3004 MHz, amount of global memory: 6143 MB.
- Debian Jessie (Stable), kernel 3.16.
- CUDA Toolkit 7.0.
- NVIDIA Visual Profiler 7.0.

For FM signals processing, GPU with higher clock frequency was used, namely GeForce GTX 780 Ti. Due to higher memory requirements, GeForce GTX TITAN was chosen for DVB-T signal processing.

A. *Performance Results of Consecutive Processing Stages*

The computation time of the consecutive processing stages was measured. Passive radar and signal parameters studied were:

- Integration time: FM – 1000 ms, DVB-T – 100 ms.
- Filter length: FM – 100, DVB-T – 1000.
- Detection window: 16 reference and 2 guard cells in velocity dimension, 8 reference and 3 guard cells in range dimension.
- Maximum acceleration: $100\,\mathrm{m/s^2}$.
- Maximum range: 300 km for FM signals and 50 km for DVB-T signals.
- Maximum velocity: $1000\,\mathrm{m/s}$.

Results for FM signal processing are presented in Tab. II. Tab. III contains performance results for CUDA realization of DVB-T signal processing. The long time of detection stage results from the number of crossambiguity matrices for various accelerations. In order to achieve real-time processing for DVB-T signals, either the algorithms have to be deployed to several graphics cards or the number of beams should be reduced.

Due to the fact that extraction and estimation time depends mainly on the number of targets, separate tests were performed. The relationship between execution time of the extraction and the number of detections is presented in Fig. 9. The curve shape result from the parallel realization of computations. After extraction, the estimation procedure is performed on CPU platform. The linear relationship between execution time of the estimation and the number of targets is presented in Fig. 10.

TABLE II
PERFORMANCE RESULTS OF PROCESSING STAGES FOR FM SIGNALS (1000 ms INTEGRATION TIME, 7 BEAMS)

Algorithm (FM)	Processing time (CUDA C)
Beamforming	0.22 ms
Filtering	1.63 ms
Crossambiguity calculation	16.89 ms
Detection (CACFAR)	34.99 ms
Extraction (approx. 40000 detections)	3.80 ms
Estimation (approx. 100 targets)	1 ms
Overall time	58.53 ms

TABLE III
PERFORMANCE RESULTS OF PROCESSING STAGES FOR DVB-T SIGNAL (100 MS INTEGRATION TIME, 7 BEAMS), MAXIMUM ACCELERATION: $100\,\mathrm{m/s^2}$

Algorithm (DVB-T)	Processing time (CUDA C)
Beamforming	1.10 ms
Filtering	12.59 ms
Crossambiguity calculation	177.15 ms
Detection (CACFAR)	164.06 ms
Extraction (40000 detections)	3.8 ms
Estimation (100 targets)	1 ms
Overall time	359.70 ms

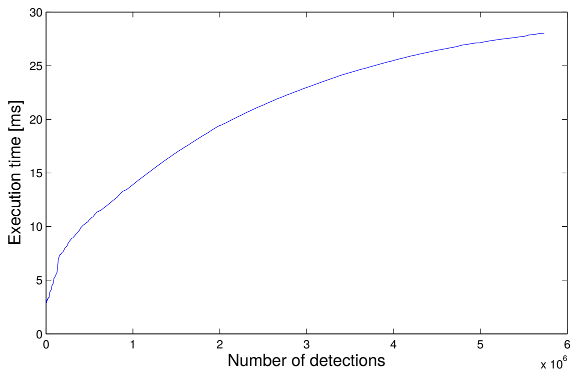

Fig. 9. Relationship between execution time of extraction and the number of detections

One of the concerns associated with implementing parallel processing is scalability. If the amount of data to process increases, it should not lead to disproportional increase in the execution time. This behavior was verified by processing varying number of the surveillance beams and measuring the execution time. The results are shown in Fig. 11. There is a linear relationship between the execution time of CUDA processing and the number of beams with the slope coefficient below 1.0. The diagram presents that calculation of data from other beams is executed in parallel.

Several measurements were made in order to investigate relation between execution time of processing in CUDA C language and the radar integration time. Results are presented in Fig. 12. It shows that the integration time enhancement results in a linear increase of the execution time.

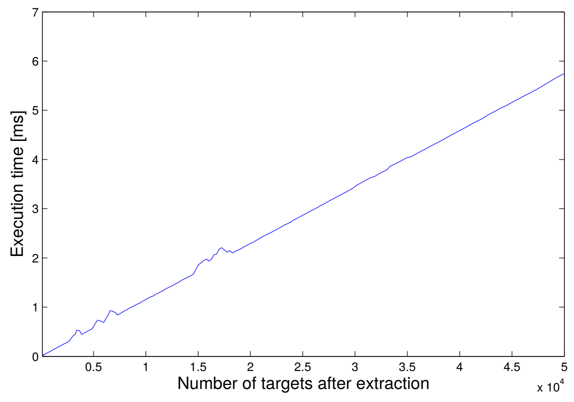

Fig. 10. Relationship between execution time of estimation and the number of targets after extraction procedure

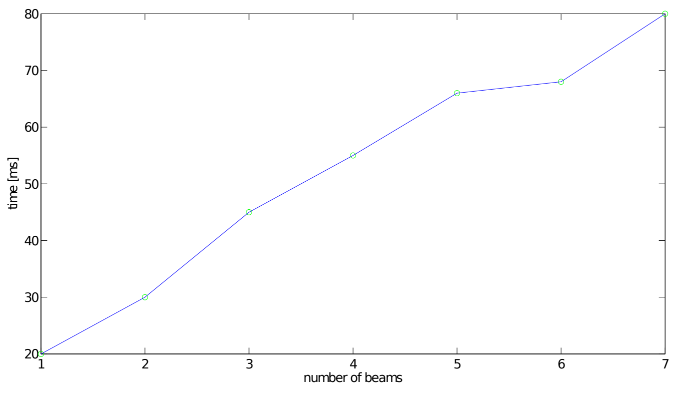

Fig. 11. Relationship between an execution time of CUDA processing and the number of beams (beamforming, filtering, crossambiguity calculation, detection), FM Signal, maximum velocity $1000\,\mathrm{m/s}$, maximum range $400\,\mathrm{km}$

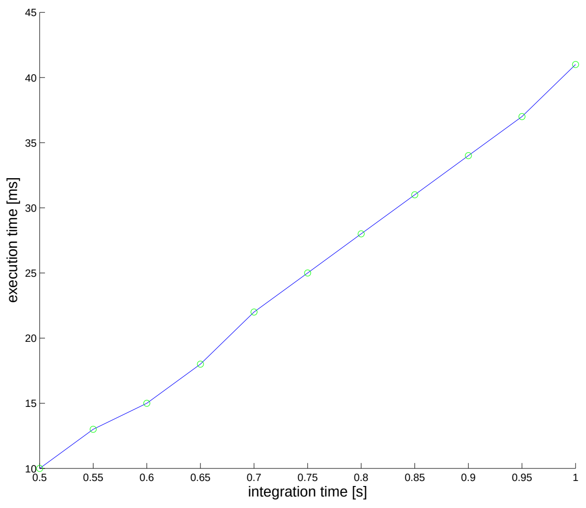

Fig. 12. Relationship between the radar integration time and the execution time of CUDA processing (beamforming, filtering, crossambiguity calculation, detection), FM Signal, maximum velocity $800\,\mathrm{m/s}$, maximum range $300\,\mathrm{km}$

B. *Accuracy of Results*

All calculations in first five stages of processing were implemented using single precision floating point numbers to achieve better performance (double precision calculations are several times slower on GPUs). The accuracy after these phases was compared to computations executed in Matlab realization of the passive radar which uses double precision numbers. The accuracy counted as a normalized difference of all data after extraction procedure is at the level of 10^{-4} which is satisfactory in the considered application.

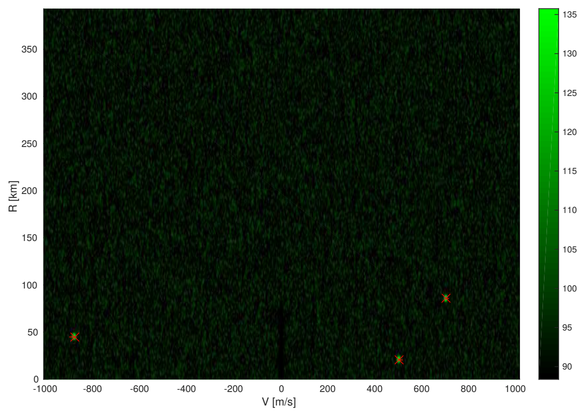

Fig. 13. Results of FM signal processing – the example of crossambiguity function and found targets (red crosses)

C. *Example Test Case of Target Detection and Estimation*

To show some sample results of radar processing, several FM and DVB-T signals with a few target echoes were simulated. Tab. IV lists parameters of simulated targets.

TABLE IV
TARGETS PARAMETERS STUDIED IN AN EXAMPLE SIMULATION

parameter	target 1	target 2	target 3
range $(r)[km]$	20	45	85
velocity $(v)[m/s]$	500	-880	700
acceleration $(a)[m/s^2]$	12	-15	10
azimuth $(doa)[^\circ]$	180	90	225

Results of processing (crossambiguity function for one beam and one acceleration) are shown in Fig. 13 (FM) and Fig. 14 (DVB-T). All found plots are marked with red crosses. Measured processing mean time was: $54.45\,\mathrm{ms}$ for FM signal ($1\,\mathrm{s}$ integration time) and $357.25\,\mathrm{ms}$ for DVB-T signal ($100\,\mathrm{ms}$ integration time).

VII. CONCLUSION

Real-time processing for passive radars is a computationally demanding task, especially for radars based on DVB-T signal, where a signal bandwidth, and therefore sampling frequency, is much higher than for FM radio. In such a situation the demand is high not only on the computing power, but also data throughput and numerical accuracy.

In this paper effective implementation of passive radar processing on GP-GPUs was presented. It was shown that the performance improvement in comparison to the original Matlab implementation is remarkable. The obtained numerical

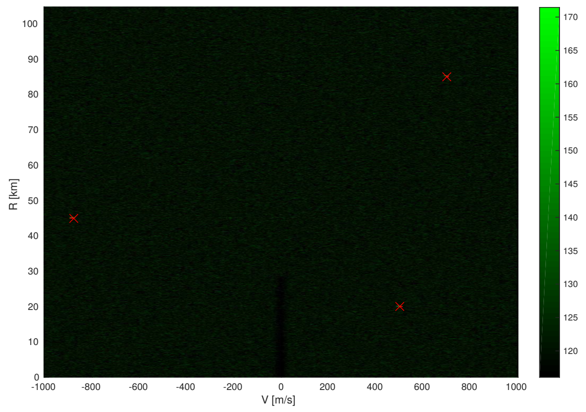

Fig. 14. Results of DVB-T signal processing – the example of crossambiguity function and found targets (red crosses)

TABLE V
RESULTS OF PASSIVE RADAR PROCESSING - ESTIMATED PARAMETERS OF FOUND TARGETS

target	parameter	FM		DVB-T	
		Matlab	CUDA	Matlab	CUDA
1	r $[m]$	20506.15	20506.15	19990.05	19990.04
	v $[m/s]$	499.99	499.99	500.00	500.00
	a $[m/s^2]$	12.96	12.96	11.50	11.50
	doa $[^\circ]$	180.10	180.10	180.02	180.02
2	r $[m]$	44872.68	44872.68	44987.10	44987.10
	v $[m/s]$	-879.92	-879.92	-880.00	-880.00
	a $[m/s^2]$	-23.73	-23.73	-15.27	-15.27
	doa $[^\circ]$	89.98	89.98	89.96	89.96
3	r $[m]$	85711.43	85711.43	85009.73	85009.73
	v $[m/s]$	699.95	699.95	700.00	700.00
	a $[m/s^2]$	9.99	9.99	7.38	7.38
	doa $[^\circ]$	269.92	269.92	270.01	270.01

accuracy is also satisfactory. The final implementation was obtained in the multi-iteration process of refining, implementing and optimizing consecutive stages of processing. In some cases, algorithm execution time was reduced at the expense of worse quality, however, each of such trade-offs was carefully examined.

Despite the fact that the current implementation provides very good performance, the research on better implementation continues, as demand to process more data (more frequency channels or more physical channels) increases.

REFERENCES

[1] H. Griffiths and C. Baker, “Passive coherent location radar systems. part 1: performance prediction,” *Radar, Sonar and Navigation, IEE Proceedings -*, vol. 152, no. 3, pp. 153–159, June 2005.
[2] K. Szumski, M. Malanowski, J. Kulpa, W. Porczyk, and K. Kulpa, “Real-time software implementation of passive radar,” in *Radar Conference, 2009. EuRAD 2009. European*, Sept 2009, pp. 33–36.
[3] K. Szczepankiewicz, M. Malanowski, and M. Szczepankiewicz, “Effective implementation of passive radar algorithms using general-purpose computing on graphics processing units,” in *Signal Processing Symposium (SPSympo), 2015*, June 2015, pp. 1–5.
[4] M. Malanowski and K. Kulpa, “Digital beamforming for passive coherent location radar,” in *Radar Conference, 2008. RADAR ’08. IEEE*, May 2008, pp. 1–6.
[5] K. Kulpa, “Adaptive clutter rejection in bi-static cw radar,” in *International Radar Symposium, 2004*, May 2004, pp. 61–66.
[6] M. Malanowski, “Comparison of adaptive methods for clutter removal in pcl radar,” in *Radar Symposium, 2006. IRS 2006. International*, May 2006, pp. 1–4.
[7] M. Malanowski, K. Kulpa, and K. Olsen, “Extending the integration time in dvb-t-based passive radar,” in *Radar Conference (EuRAD), 2011 European*, Oct 2011, pp. 190–193.
[8] M. Malanowski, K. Kulpa, and J. Misiurewicz, “Acceleration estimation for passive coherent location radar,” in *Radar Conference, 2008. RADAR ’08. IEEE*, May 2008, pp. 1–5.
[9] M. Malanowski, “Target acceleration estimation for continuous wave noise radar,” in *Proc. International Radar Symposium 2005*, September 2005, pp. 177–183.
[10] L. Scharf and C. Demeure, *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis*, ser. Addison-Wesley series in electrical and computer engineering. Addison-Wesley Publishing Company, 1991. [Online]. Available: https://books.google.pl/books?id=y_dSAAAAMAAJ
[11] (2015) Cuda computing guide. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/
[12] D. Kirk and W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier Science, 2012. [Online]. Available: https://books.google.pl/books?id=E0Uaag8qicUC
[13] (2015) The nvidia gtx titan. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan
[14] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Pearson Education, 2013. [Online]. Available: https://books.google.pl/books?id=ynydqKP225EC
[15] R. Farber, *CUDA Application Design and Development*, ser. Applications of GPU computing series. Morgan Kaufmann, 2011. [Online]. Available: https://books.google.pl/books?id=MtLvlQvYDOEC
[16] (2015) Cuda toolkit documentation (cublas). [Online]. Available: http://docs.nvidia.com/cuda/cublas/
[17] (2015) Cuda computing guide (cufft). [Online]. Available: http://docs.nvidia.com/cuda/cufft/