Advanced GNSS-R Signals Processing With GPUs

Oriol Cervelló i Nogues, Daniel Pascual, *Student Member, IEEE*, Raul Onrubia, *Student Member, IEEE*, and Adriano Camps

***Abstract*—Global navigation satellite system reflectometry (GNSS-R) is a group of techniques that uses satellite navigation signals as signals of opportunity for remote sensing applications. In GNSS-R, large amounts of data are acquired and have to be processed. Computation time is typically the bottleneck for ground and airborne experiments. This article presents an efficient solution for off-line GNSS-R processing data taking advantage of graphics processing units (GPUs). After comparing to the typically used CPU languages, such as MATLAB and C++, the advantage of using parallel processing on the GPU is clear. GPU-based computation can reduce the processing time by as much as 95% of the acquisition time of the data. An implementation taking advantage of a home-use GPU is proposed for the data processing units. Thanks to its efficiency, even real-time processing experiments are feasible.**

***Index Terms*—Compute unified device architecture (CUDA), global navigation satellite system reflectometry (GNSS-R), graphics processing unit (GPU) processing, parallel computing, real-time processing.**

I. Introduction

GRAPHICS processing units (GPUs) were initially designed to accelerate graphics computing. However, more recently, they have also been used to accelerate computation times of large amounts of data. In global navigation satellite system reflectometry (GNSS-R), large amounts of data have to be processed either in real-time or offline, and the computation time is a bottleneck. Common programming languages in academia, such as MATLAB [1], are too slow to process large amounts of data. GPU processing has already been applied to some GNSS receiver experiments [2], [3], and it has been used in GNSS-R for specific experiments and specific constellations [4], [5]. The goal of this article is to take advantage of GPUs to develop an efficient generic GNSS-R processor and present efficiency results. A full software tool has been developed to process real GNSS-R data, and has been validated with data collected by the microwave interferometric reflectometer (MIR) airborne campaign [6], [7].

The MIR instrument is a multibeam dual-band GNSS-R with real-time beam-steering capabilities to compensate the aircraft attitude so as to point to the desired GNSS (GPS or Galileo) satellites and reflection points. The up-looking array points directly to the GNSS satellites, while the down-looking one points to the corresponding specular reflection points on the earth's surface. The instrument can track up to four different satellites, and their corresponding specular reflection points at L1/E1 (1575.42 MHz) with a bandwith of 20 MHz and L5/E5a (1176.45 MHz) with a bandwith of 34 MHz [6]. During the MIR maiden flight experiments, the signals were down-converted to baseband and sampled at 32.736 MSps at 1 b for both the I and Q components, and then stored in a computer for off-line processing. In the future, it is foreseen to keep the multibit sampling and real-time data processing.

During that campaign, MIR recorded about 2 TB of data that had to be processed in a time and cost-effective manner. This involves processing from the different constellations and bands and extracting all possible observables—waveforms (WF), delay-Doppler Map (DDM), and signal-to-noise ratio (SNR) of the signal, for conventional (cGNSS-R) or interferometric (iGNSS-R) techniques at different coherent/incoherent integration times [8]. WF are the time cross-correlations between the received reflected signal and either the direct one (iGNSS-R) or a locally generated replica of the transmitted signal (cGNSS-R) [8]. The DDMs are multiple WF computed for different Doppler frequencies. The SNR is defined as the ratio between the peak of the DDM minus the noise, divided by the noise. The noise is the average power computed for the delays well before the correlation peak.

This article proposes a GPU-based software approach to implement a GNSS-R data processing unit. Section II describes the proposed implementation. Section III presents results and analysis of time elapsed to process experimental data. Section IV presents the validation of the processed data, and Section V summarizes the main conclusions.

II. Implementation

The first step in any implementation is the algorithmic and hardware tradeoff. Dealing with large amounts of data to be processed efficiently is the main bottleneck in this implementation.

Manuscript received August 1, 2019; revised December 18, 2019 and February 6, 2020; accepted February 15, 2020. Date of publication March 9, 2020; date of current version April 8, 2020. This work was supported in part by the Spanish Ministry of Economy and Competitiveness and FEDER EU under the project "AGORA: Técnicas Avanzadas en Teledetección Aplicada Usando Señales GNSS y Otras Señales de Oportunidad" (MINECO/FEDER) ESP2015-70014-C2-1-R by the Agencia Estatal de Investigación; in part by the Spanish Ministry of Science, Innovation, and Universities, "Sensing with Pioneering Opportunistic Techniques," under Grant RTI2018-099008-B-C21; in part by the Collaboration under Grant 998142 by the Spanish Ministry of Education; in part by Unidad de Excelencia María de Maeztu MDM-2016-0600; and in part by Prof. Camps ICREA Academia 2015 award of the Generalitat de Catalunya. *(Corresponding author: Oriol Cervelló i Nogues.)*

Oriol Cervelló i Nogues is with the Department of Information and Communication Technologies, Universitat Pompeu Fabra, 08002 Barcelona, Spain, and also with the CommSens Lab - Maria de Maeztu unit, Department of Signal Theory and Communications, Universitat Politècnica de Catalunya and IEEC/CTE - UPC, 08002 Barcelona, Spain (e-mail: oriol.cn@protonmail.com).

Daniel Pascual, Raul Onrubia, and Adriano Camps are with CommSens Lab - Maria de Maeztu unit, Department of Signal Theory and Communications, Universitat Politècnica de Catalunya and IEEC/CTE - UPC, 08002 Barcelona, Spain (e-mail: daniel.pascual@tsc.upc.edu; onrubia@tsc.upc.edu; camps@tsc.upc.edu).

Digital Object Identifier 10.1109/JSTARS.2020.2975109

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

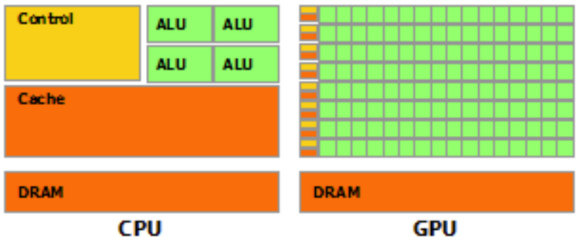


Fig. 1. Comparison of CPU and GPU hardware. GPU includes more ALUs for data processing (from [10]).

Luckily, the kind of operations needed for this processing are highly parallelizable. A software program taking advantage of a GPU can benefit from speed improvements, thanks to parallel computing. One of the objectives of this implementation is to use home-use GPUs to efficiently process the GNSS-R data. In almost all PCs, there is a GPU able to support parallel computing. In this way, getting new and specific hardware can be avoided.

An alternative would be running the software on the central processing unit (CPU), which also does not require extra hardware, but it offers limited computation resources as compared to the GPU, as it will be shown later. Likewise, to use a field-programmable gate array (FPGA)-based approach normally has much lower latency and lower clock frequency, but it has less power to compute and less parallelization efficiency [9]. Moreover, an expensive FPGA is required, while almost every computer has a GPU.

A generic computer program is a set of instructions to be executed, generally in a sequential order and interpreted by a CPU, one at a time (one thread). Nonetheless, some calculations or processes can be computed simultaneously with multiple threads (parallel computing). This technique allows to divide large problems or large amounts of data to be processed in parallel threads that can be executed independently at the same time, increasing the processing speed. An average home PC is able to parallelize tasks either in the CPU or in the GPU. The speed improvement depends on the number and power of the cores. Current CPUs have more than one core (e.g., four or eight), while GPUs usually have hundreds of them, since they are designed for video signal processing. In addition, GPUs include special vector instructions that support implementation of massive parallelism [3], and are able to process both graphic and general data.

Generally speaking, a CPU has fewer, but more powerful arithmetic logic units (ALUs) than a GPU, and a larger cache. On the other hand, a GPU has more, but simpler ALUs with a smaller cache (see Fig. 1). An operation or a function, such as a multiplication, an accumulation, or a fast Fourier transform (FFT), would be done faster in the CPU, as it is more powerful and saves the time to move all the data to the GPU. However, instead of a single operation, thousands of them have to be computed at a time, the extra ALUs in the GPU make the difference by parallelizing the workload.

The most general open source language for GPU parallelization is Open Computing Language (OpenCL) [11], which is supported by most GPU vendors. Compute unified device architecture (CUDA) language is another freeware parallel computing language developed for NVIDIA graphic cards [10]. One key point of the CUDA language is that it includes optimized signal processing libraries, called NPP, and an efficient implementation of the FFT, called cuFFT. Comparing OpenCL and CUDA, the latter offers a simpler programming language, and a better performance, although it can only be run in NVIDIA graphic cards. OpenCL is better if the application has to be a cross-platform program, to be distributed in many different devices with different graphic cards [11]. Looking for the best performance and friendly environment, the presented GPU implementation is done in CUDA.

The implementation design is based on a circular cross correlation based on FFTs. The algorithms to compute the Fourier transforms are so efficient that in most of the cases, when implementing a correlation, the best procedure to implement it is with FFTs rather than a linear correlation for each delay [12]. The procedure is simplified to operations that can be easily parallelized like the FFTs of the two signals, point-to-point multiplication in the frequency domain, the inverse FFT (IFFT) of the result, and finding the maximum. These operations can take advantage of the libraries cuFFT and NPP which have efficient implementations of the operations to be parallelized in the GPU. The correlation implemented is shown as

$$Y(\tau, f_d) = |\text{IFFT}(\text{FFT}(y_r * e^{j2\pi f_d}) * \text{conj}(\text{FFT}(y_d)))| \quad (1)$$

where y_r corresponds to the reflected signal in the time domain. In the case of using interferometric GNSS-R (iGNSS-R) technique, y_d is the direct signal from the satellite. If the technique used is conventional GNSS-R case (cGNSS-R), y_d is the locally generated PRN code replica. The $e^{j\pi f_d}$ is a complex sinusoidal, where f_d is the Doppler frequency. $Y(\tau, f_d)$ is the correlation, where τ is the time, and f_d the Doppler frequency. If there is only one f_d, the result is a waveform, while if there are more, is a DDM.

Fig. 2 shows a schematic of the implemented processing for cGNSS-R technique, following the (1). The reflected signal is divided in chunks, in this example 3 of 1 ms worth of data, and each chunk is multiplied by a complex sine with the corresponding Doppler frequency after they are multiplied by the complex conjugate of the Fourier transform of the clean replica. If it was using iGNSS-R technique instead of the clean replica, it would be the corresponding chunks of the direct signal from the satellite. Then, the IFFT is computed. Each group of M chunks (defined by the user, 3 in the example) are added to perform incoherent additions and find the peak, this is specially needed if the signal is very noisy. Finally, after incoherent averaging in each of the IFFTs, the zone where the maximum peak has been found is saved. In this way, only the region of interest is saved, reducing a lot of the data. The length of this zone around the peak is user-defined.

In this implementation, the parallelization is done simultaneously for all the samples of all the processed chunks. Fig. 3 shows a block diagram of the processing. After the initial preparation, the program enters in a loop, and N chunks of data will be processed. Once it has finished with the processing, it starts again with the next N chunks. In Fig. 3, the purple operations are the ones parallelized in the GPU. The parallelization is done at a sample level, except for the already defined functions FFT,

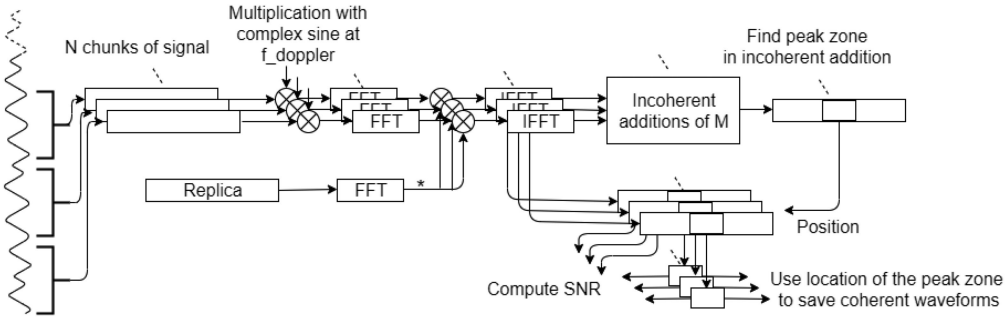


Fig. 2. Block diagram. Schematic of the processing to process a reflected signal with a local replica of PRN and cGNSS-R. N chunks correspond to the number of chunks processed simultaneously, but the signal can be longer by repeating the process (see Fig. 3). M is the number of chunks used for each incoherent addition, which has to be a divisor of N. In order to generate DDM, the same architecture is replied as many times as Doppler frequencies for the same data, to generate all the Doppler bins.

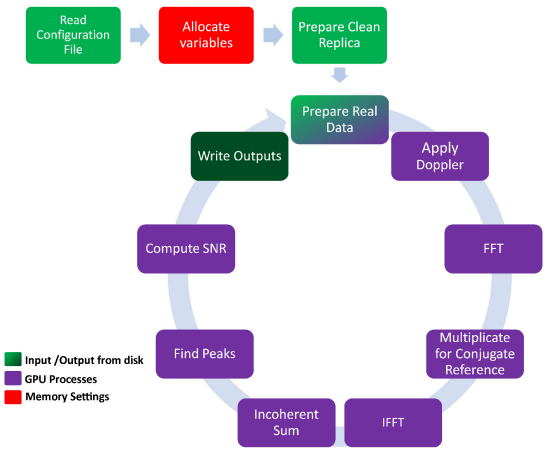


Fig. 3. Workflow diagram, after the 3 first blocks of initialization, the program start the loop of processing, each loop the program processes N chunks of data of the signal, for more detail of the processing see Fig. 2.

IFFT (in cuFFT), and the incoherent summation (in NPP) that are using their own methods.

The operations previously explained correspond to one chunk of data (see Fig. 2). By replicating this architecture for the same data as many times as Doppler frequencies, this implementation is capable of computing any number of Doppler bins of the same signal to create DDMs with variable resolution.

With this implementation, it is possible to process data from different constellations, bands, modulations, and code lengths. Latter in the article, for example, results of this implementation for GPS L1 C/A which is a BPSK(1) with a code length of 1 ms, and from Galileo E1OS which is a CBOC(6,1,1/11) with a code length of 4 ms will be presented. It was a requirement to be able to process different types of data, as the MIR captures different constellations (Galileo and GPS), different bands (L1/E1 and L5/E5a), using different modulations and code lengths. Other constellations and bands not mentioned are also configurable. It is independent of the modulation used, and it can process with any code lengths, even if they are not multiple of the basic PRN code length (i.e., processing 2.5 ms of GPS L1 C/A). In summary, the implementation achieves the processing of WF and DDM for both cGNSS-R or iGNSS-R techniques.

III. Time Results of Processing Experiments

The objective of this implementation is to process GNSS-R data time efficiently. Important operations for GNSS-R (shown in (1), are the FFT and its inverse (IFFT), the complex conjugate, the point-to-point complex multiplication, the point-to-point complex summation, and the maximum search of a signal. To test the performance of this implementation, these functions have been implemented in the following:

1) MATLAB R2018b as an interpreted language, and because it is widely used in research and academia;
2) C++ as a reference compiled procedural language using the FFTW library version 3.3.5 [13];
3) CUDA 10 with the cuFFT and NPP included libraries.

Tests have been performed in two different hardware platforms, both designed for home use. The first one is a low end—2013 Laptop Intel i7 (4th gen) at 2.20 GHz, 16 GB RAM, NVIDIA GeForce GT 750 M (384 cores at 967 MHz with 4 GB of dedicated memory), 64-b Windows 8. The second one is a high-end 2019 Tower Intel i7 (8th gen) at 3.20 GHz. 32 GB RAM, NVIDIA RTX 2080 (2944 cores at 1472 MHz with 8 GB of dedicated memory), 64-b Windows 10.

Fig. 4(a) shows the elapsed time performing each operation for 1000 chunks of 32 736 complex float samples of data, which is the equivalent to 1 s of the MIR GPS L1 C/A data. CUDA outstands in performance in every function, and it is considerably faster than C++ or MATLAB.

The performance of CUDA is equally remarkable considering the whole final implementation, not just some of the functions. Fig. 5(a) shows the elapsed time involved in the whole processing for different amounts of chunks of data of size 32 736 samples using cGNSS-R and iGNSS-R techniques. Comparing both methods, processing for iGNSS-R technique is always a bit slower because it is computing the FFTs of the whole reflected signal and the same for the direct signal from the up-looking antenna to correlate. For cGNSS-R, it is almost half the FFTs needed, as it only computes the FFT of the reflected signal and one time one FFT for the local replica PRN to correlate with.

Looking at the results, processing 5 s of data (5000 chunks of data) using cGNSS-R technique takes 0.262 s, and using iGNSS-R technique takes 0.336 s. In comparison to the acquisition time of the data (5 s), the process takes around 5.2%–6.7%

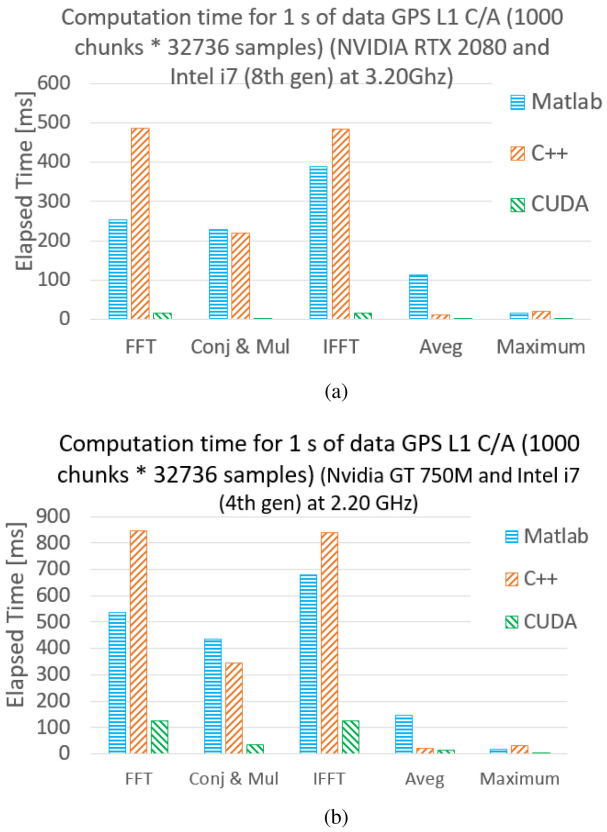


Fig. 4. Comparison of elapsed times of different functions implemented in different programming languages. (a) With NVIDIA RTX 2080 and Intel i7 (8G) at 3.20 GHz. (b) With NVIDIA GT 750 M and Intel i7 (4G) at 2.20 GHz.

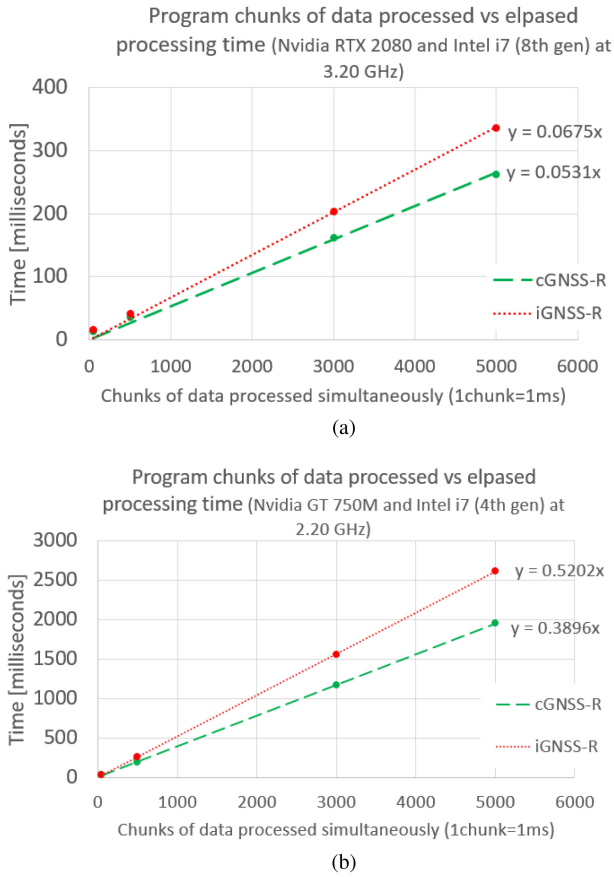


Fig. 5. Elapsed time for simultaneous processing of different amounts of chunks of data with the whole final implementation. In this case, GPS L1 C/A data which correspond a chunk size of 3 2736 samples. For cGNSS-R and iGNSS-R techniques. (a) With NVIDIA RTX 2080 and Intel i7 (8th gen) at 3.20 GHz. (b) With NVIDIA GT 750 M and Intel i7 (4th gen) at 2.20 GHz.

(see Table I) of the time, respectively. This is a reduction of time of 93.3%–94.8% of the acquisition time of the data.

The results mentioned until now are from a PC with a NVIDIA RTX 2080, though being a GPU designed for home computers it is a powerful, and high-end one. To contrast results, this implementation has also been tested with a less powerful and low-end hardware (NVIDIA GT 750 M).

Fig. 4(b) shows the comparison between CUDA, MATLAB, and C++, this time with the GT 750 M. Times are slower in general for the three languages and for all the functions. Nonetheless, the pattern of results is maintained and CUDA outperforms the other languages. Comparing the CUDA results from a home lower-end PC [see Fig. 4(b)] with MATLAB and C++ from a home high-end PC [see Fig. 4(a)], CUDA with a less powerful hardware is faster. Parallelization of these operations is essential to gain time efficiency.

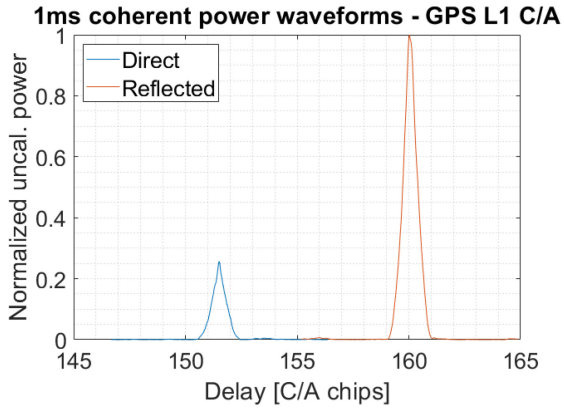
Fig. 5(b) shows the elapsed time of the whole processing for different amounts of chunks of data for the GT 750 M, as Fig. 5(a) for RTX 2080. The slope values indicate the increase in time for each extra chunk of data. The value for GT 750 M is around 7 times larger in comparison to the RTX 2080. GT 750 M takes more time to process the 5 s of data, 1.9 s for processing using cGNSS-R technique, and 2.6 s for processing using iGNSS-R technique (see Table I). Nevertheless, GT 750 M still processes faster than the acquisition time of the data.

TABLE I
RESULTS OF THE ELAPSED TIME PROCESSING 5000 CHUNKS OF DATA WITH THE TWO DIFFERENT GPUS, AND COMPARING CGNSS-R AND IGNSS-R

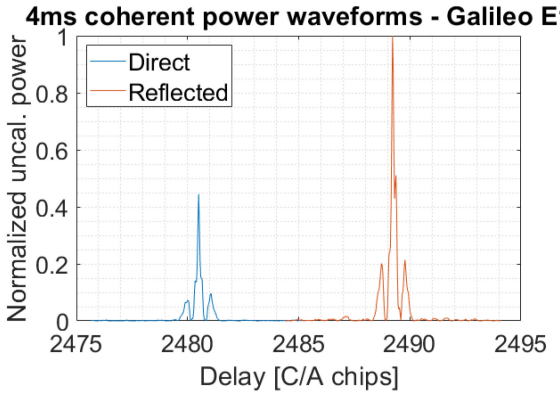
	RTX 2080 cGNSS-R	RTX 2080 iGNSS-R	GT 750M cGNSS-R	GT 750M iGNSS-R
Chunks[#]	5000	5000	5000	5000
Acquisition time[ms]	5000	5000	5000	5000
Mean Proc. time[ms]	262	336	1948	2609
Conf. Interv.[ms]	262±5	336±2	1948±19	2609±22
Std. deviation[ms]	18	7	69	76
Mean/Acq. time [%]	5.25	6.72	38,97	52.18
Conf. Int./Acq. t.[%]	5.25±0.10	6.72±0.04	38.97±0.40	52.18±0.44

Confidence interval computed using a student's t distribution and significance level alpha = 0.05.

The results for the RTX 2080 were a factor of 14.9x–19.1x faster respect the acquisition time. It is noticeable that GT 750 M is much less powerful than the RTX 2080, although its results validate the efficiency of the implementation in lower end hardware with an increment of 1.9x–2.6x faster than the acquisition time. For both GPUs, the data is always processed fast enough so as to enable real-time processing with this implementation.



(a)



(b)

Fig. 6. Direct and reflected WF computed with (a) 1 ms coherent of GPS L1 C/A and (b) 4 ms coherent Galileo E1. The reflected signal is stronger because of an additional amplifier installed in the down-looking antenna.

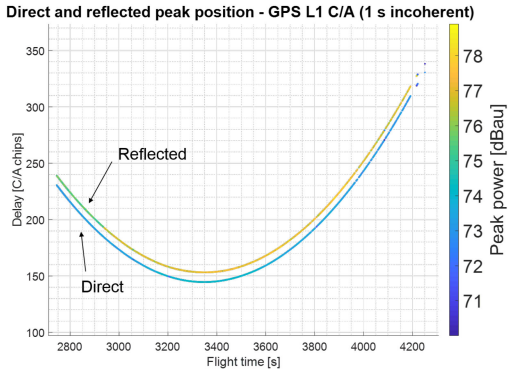


Fig. 7. Tracking of the direct and reflected signal of satellite over time for a satellite. The reflected signal has delayed some chips with respect to the direct one. The tracking of the signal is done by correlating it with a local replica of the original PRN. The direct signal has to travel from the satellite to the plane, while the reflected one has to travel from the satellite to the earth's surface and bounces back to the plane.

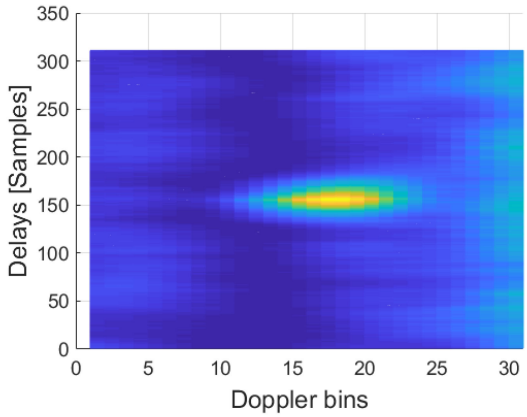


Fig. 8. Example DDM from of 1 ms coherent of GPS L1 C/A build from 31 Doppler slices (WF). Reflection over the sea.

IV. Validation

The MIR instrument already flew in an airborne campaign conducted in Australia from April–June 2018 [7]. The raw data acquired during the flights are useful to validate the proper functioning and the achievement of the requirements of the implementation presented.

The configuration parameters of the program include the adjustable integration times (coherent and incoherent). Fig. 6 shows two different coherent integration times. The first one, 1 ms tested for a PRN code of GPS L1 C/A, whose length is 1 ms and a BPSK modulation. The second one, 4 ms tested on a Galileo E1 PRN code, whose length is 4 ms and CBOC modulation. The reflected signal is stronger than the direct one because of an additional amplifier inserted in the down-looking antenna and not compensated in these results. Altimetric applications are computed with the delay between the direct and reflected signals. Fig. 7 shows the tracking of the delay of a satellite direct signal and the reflected on the earth's surface over time.

Incoherent integration time is a key factor in GNSS-R applications. Lower values of incoherent integration time results in more frequent sampling of the reflections, but also lead to higher noise. On the other hand, higher values of incoherent integration time results in less temporal/spatial resolution of the reflections but less noisy data. Thanks to GPU processing, it is possible to compute different incoherent integration times in an efficient manner.

Fig. 8 is an example of a coherent DDM computed with this implementation. For a chunk of data corresponding to 1 ms of GPS L1 C/A, it has been processed 31 times, in this case, with different Doppler frequencies in order to create a DDM. In this implementation, it is possible to change the number of Doppler bins and the resolution in Hz between them.

V. Conclusion

This article has presented an approach to efficiently process GNSS-R signals, and has analyzed its efficiency. The key point of this implementation is to take advantage of GPUs designed for home use. In a comparison made using different CPU and GPU languages (see Fig. 4), the time involved is around the same, or higher, than the acquisition time in the CPU languages. In the test, only five of the required functions for GNSS-R processing have been implemented, which means the whole implementation would take even more time. With GPU processing the advantage is clear, the whole implementation processes data in around 5% of the acquisition time of the same data, as it can be seen in Table I, 5000 ms of data are being processed in less than 300 ms.

This time efficiency is achieved by taking advantage of parallel processing. In addition, a home lower-end GPU has been tested to contrast the time results of processing experiments and its results have special relevance as they had also demonstrated the capability of processing faster than the acquisition time.

The proper functioning of this implementation has been validated with real data from the maiden airborne campaign of the MIR. Different observables that can be computed have been shown, for different constellations (GPS and Galileo), time lengths, and modulations.

This software approach demonstrates the improvement when using GPU-based computing for GNSS-R off-line processing. From now on, GPU-based implementations will be a valuable option to take into account in future designs of GNSS-R processing units not only for off-line data processing units, but also for real-time processing ones.

REFERENCES

[1] MathWorks, "MATLAB." Accessed: Jul. 9, 2019. [Online]. Available: https://www.mathworks.com/products/matlab.html

[2] L. Xu, N. I. Ziedan, X. Niu, and W. Guo, "Correlation acceleration in GNSS software receivers using a CUDA-enabled GPU," *GPS Solutions*, vol. 21, no. 1, pp. 225–236, Jan. 2017.

[3] P. Roule, O. Jakubov, P. Ková, P. Kamaík, and F. Vejražka, "GNSS signal processing in GPU," *Artif. Satell.*, vol. 48, no. 2, pp. 51–61, 2013.

[4] T. Hobiger, J. Amagai, M. Aida, and H. Narita, "A real-time GNSS-R system based on software-defined radio and graphics processing units," *Adv. Space Res.*, vol. 49, no. 7, pp. 1180–1190, Apr. 2012.

[5] T. Hobiger, R. Haas, and J. S. Löfgren, "GLONASS-R: GNSS reflectometry with a frequency division multiple access-based satellite navigation system," *Radio Sci.*, vol. 49, no. 4, pp. 271–282, 2014.

[6] R. Onrubia, D. Pascual, J. Querol, H. Park, and A. Camps, "The global navigation satellite systems reflectometry (GNSS-R) microwave interferometric reflectometer: Hardware, calibration, and validation experiments," *Sensors*, vol. 19, no. 5, Feb. 2019, Art. no. 1019.

[7] R. Onrubia *et al.*, "Satellite cross-talk impact analysis in airborne interferometric global navigation satellite system-reflectometry with the microwave interferometric reflectometer," *Remote Sens.*, vol. 11, no. 9, May 2019, Art. no. 1120.

[8] V. U. Zavorotny, S. Gleason, E. Cardellach, and A. Camps, "Tutorial on remote sensing using GNSS bistatic radar of opportunity," *IEEE Geosci. Remote Sens. Mag.*, vol. 2, no. 4, pp. 8–45, Dec. 2014.

[9] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of FPGAs and GPUs," in *Proc. 26th IEEE Int. Symp. Field-Programmable Custom Comput. Mach.*, Sep. 2018, pp. 93–96.

[10] NVIDIA, "CUDA Toolkit Documentation." Accessed: May 29, 2019. [Online]. Available: https://docs.nvidia.com/cuda/

[11] L. C. M. D. Paula, "CUDA vs. OpenCL: Uma comparação teórica e tecnológica," *ForScience*, vol. 2, no. 1, pp. 31–46, 2014.

[12] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed., Prentice Hall, 1998.

[13] FFTW, "FFTW." Accessed: Jul. 9, 2019. [Online]. Available: fftw.org

Oriol Cervelló i Nogues was born in Barcelona, Spain. He received the bachelor's degree in telecommunications engineering from the Universitat Pompeu Fabra (UPF), Barcelona, Spain, in June 2019.

During his last year of degree, he has joined the Passive Remote Sensing Group, Signal Theory and Communications Department, Universitat Politècnica de Catalunya-BarcelonaTech (UPC), Barcelona, Spain, to develop his degree thesis in GNSS-reflectometry (GNSS-R). His current work follows the degree thesis he has been working on and involves GNSS-R data processing and GPU parallel programming.

Daniel Pascual (Student Member, IEEE) was born in Barcelona, Spain, in 1985. He received the B.Sc.+5 degree in telecommunications engineering specialized in communications in 2011, and the M.Sc.+2 degree in research on information and communication technologies in 2014, from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, where he is currently working toward the Ph.D. degree in GNSS-reflectometry (GNSS-R) with the Passive Remote Sensing Group focused on GNSS-R real-time data processing using FPGA, since 2011.

Raul Onrubia (Student Member, IEEE) was born in Barcelona, Spain. He received the B.Sc. degree (B.Sc.+5) in telecommunications engineering in 2012 and the M.Sc. degree (M.Sc.+2) in research on information and communication technologies in 2014 from the Universitat Politècnica de Catalunya, Barcelona, Spain, where he is currently working toward the Ph.D. degree in GNSS-reflectometry with the Passive Remote Sensing Group, Signal Theory and Communications Department.

His current work is the RF hardware development and calibration, GNSS-R data processing, and the study of satellite cross talk, radio frequency interferences, and mitigation techniques.

Adriano Camps was born in Barcelona, Spain, in 1969. He received the degree and Ph.D. degrees in telecommunications engineering from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1992 and 1996, respectively.

From 1991 to 1992, he was with the ENS des Télécommunications de Bretagne, France, with an Erasmus Fellowship. In 1993, he joined the Electromagnetics and Photonics Engineering Group, Department of Signal Theory and Communications, UPC, an Associate Professor in 1997, and has been a Full Professor since 2007. In 1999, he was on sabbatical leave from the Microwave Remote Sensing Laboratory, University of Massachusetts, Amherst, MA, USA. His research interests include microwave remote sensing, with special emphasis in microwave radiometry by aperture synthesis techniques (MIRAS instrument onboard ESA's SMOS mission), remote sensing using signals of opportunity (GNSS-R), and nanosatellites as a tool to test innovative remote sensors. He is the Scientific Coordinator of the CommSensLab—María de Maeztu Excellence Research unit with the Department of Signal Theory and Communications. Within CommSensLab, he co-led the Remote Sensing Laboratory, and co-leads the UPC NanoSat Laboratory. He is the PI of the first four UPC nano-satellites. He has authored or coauthored more than 194 papers in peer-reviewed journals, six book chapters and one book (860 pages), and more than 410 conference presentations. He holds ten patents, and has advised 22 Ph.D. Thesis students (+ 9 on-going), and more than 125 final project and M.Eng. Theses.