www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2011, VOL. 57, NO. 3, PP. 363–368
Manuscript received July 6, 2011; revised August 2011. DOI: 10.2478/v10177-011-0049-0

Real Time Processing of Networked Passive Coherent Location Radar System

Mathew John, Michael Inggs, and Dario Petri

***Abstract*—A Passive Coherent Location (PCL) Radar system, consisting of spatially distributed transmitters and receivers is currently being integrated at the University of Cape Town (UCT). The paper investigates the feasibility of real-time processing of PCL system signals using Graphic Processing Units (GPUs), specifically a study of two distinct clutter cancellation algorithms: ECA (Extensive Cancellation Algorithm) and NLMS (Normalised Least Mean Square). Clutter cancellation is the most computationally demanding part of PCL signal processing. This investigation compares the processing speed-up achieved by GPU over CPU implementations, with very encouraging results.**

***Keywords*—Passive, PCL, real-time, GPU.**

I. Introduction

THE doubling of the number of transistors every 18 months (known as Moore's law [1]) and the demand for sophisticated computer game consoles have come together in the form of Graphic Processor Units (GPUs) and a repackaged version for research computing, known as the General Purpose GPU (GPGPU). Research on PCL systems, which gained popularity as a countermeasure to stealth and as a possible way to reduce demands on precious RF spectrum, calls for massive computational power. Large amounts of data have to be processed in real-time, using floating point representations. The processing is susceptible to parallel computation.

The PCL system that is currently used [2], [3] is based on the Universal Software Radio Peripheral (USRP) [4] as the front-end receiver and data pre-conditioner. Two clutter cancellation algorithms – the Normalised Least Mean Square (NLMS) [5] and Extensive Cancellation Algorithm (ECA) [6] have been coded. The GPU used is the Nvidia GTX480 FX [7] and the development environment is Compute Unified Device Architecture (CUDA) toolkit 3.1 [8]. The data processed was collected from near Cape Town International Airport from targets of opportunity.

The paper begins with an overview of the PCL system being implemented at UCT, for deployment in the Western Cape. The objective of this is to demonstrate whether PCL can compete at some level of performance with more traditional Air Traffic Control Radar [3], [9]. We continue to discuss the two algorithms implemented, and conclude by reporting on some of the timings achieved with respect to conventional, single thread processing.

M. John and M. Inggs are with Radar Remote Sensing Group, Department of Electrical Engineering, University of Cape Town, Rondebosch 7701, South Africa (e-mails: mathewjohnth@gmail.com, michael.inggs@uct.ac.za).

D. Petri is with RaSS Centre, National Inter-University Consortium for Telecommunications, Pisa, Italy (e-mail: petridario@gmail.com).

II. PCL System

The PCL system currently under investigation utilises the USRP and is connected to a PC via USB 2.0 at a decimated rate of 32 MBps (or 4 MHz per channel) for both the reference and surveillance channel. The acquisition code is in C++ and the sequential nature of the process limits the scope for parallel processing at the acquisition end. Figure 1 gives a system level diagram of the set-up.

The earlier RF Module available with the USRP has not demonstrated good frequency selectivity and is handicapped by a poor noise figure and a limited dynamic range. In addition, the RF units used for the surveillance and reference channels do not have synchronised local oscillators, so the frequency offset has to be determined and removed in signal processing. An improved RF front-end will be implemented in future measurements.

The preliminary CPU implementation of the processing chain, including clutter cancellation exhibited high computational intensity. The paper focusses on the parallel implementation of the parts of the PCL Radar processing chain in a heterogeneous computing platform [10]. Individual stages are identified for parallel processing by classifying processes based on arithmetic intensity and requirement for flow control. Processes that fit into the parallel processing architecture of a GPU (a highly vectorised processor) are implemented in the GPU and sequential processes are performed in CPU itself. The primary aim of the implementation is acceleration of the signal processing chain. The processes selected for parallel computation in GPU are Direct path Interference(DPI) and clutter cancellation (NLMS or ECA) and Amplitude

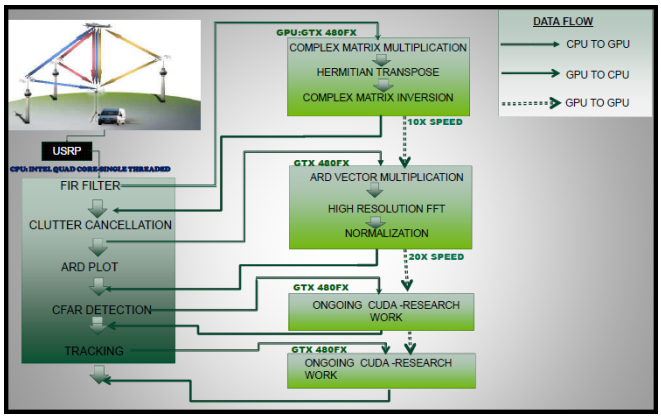

Fig. 1. PCL system consisting of a USRP front-end connected to a PC (Top-left corner image-Courtesy: Onera,France).

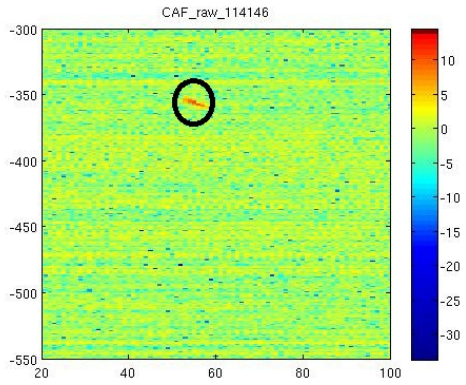

Fig. 2. Target detection (circled) at (60,-350) using GPU with DVB-T data and NLMS.

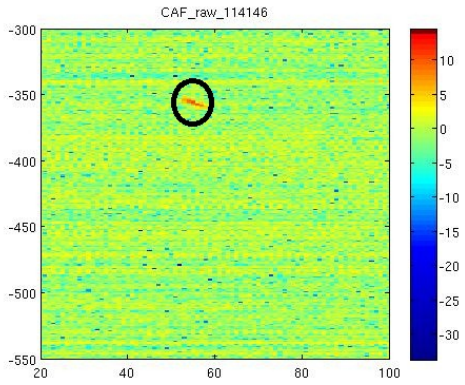

Fig. 3. Target detection (circled) at (60,-350) using CPU with DVB-T data and NLMS.

Range Doppler (ARD) Processing. Constant False Alarm Rate (CFAR) detection and tracking is part of ongoing research work [11].

III. GPU Implementation

The processing stage of the PCL system can be divided into two: DPI cancellation and clutter removal, and Matched filtering. These are described in the following subsections. In this section, the CPU is mentioned as *host* and GPU as *device* in accordance with Nvidia terminology [10].

A. Direct Path Interference Cancellation and Clutter Removal

The surveillance channel of PCL radar system contains a strong DPI component together with time-delayed reflections from clutter sources. These unwanted signals correlate with the reference channel, masking weak reflections from moving targets. In addition to physical and spatial techniques, an efficient algorithm has to be implemented for DPI cancellation and clutter removal. Two different algorithms, i.e. the NLMS and the ECA, can be used depending upon the source of illumination selected. We have tested both DVB-T (Digital Video Broadcasting-Terrestrial) and FM (Frequency Modulation) broadcast signals. The two algorithms are discussed below.

1) NLMS algorithm: The NLMS algorithm [5] is a computationally efficient algorithm with good results. The filtering model is defined by Haykin [12]. The order of the filter and the filter coefficient μ_{NLMS} [12] are the inputs to the algorithm. The CPU implementation of NLMS was coded in ANSI C [2]. Individual processes were identified and step by step migration to the GPU platform was performed with priority for maximum speed-up. The processing is carried out in the GPU with the virtue of both global and shared memory for the floating point operations, with customised kernels for individual operations. The calculation of the error correction factor [13] and calculation of the adaptation factor [12] are the main processes implemented in GPU.

Optimization of the code is performed based on strategies mentioned in [10]. Data transfer between individual stages are performed by internal transfer within the GPU reducing memory operations between the host and the device.

The NLMS algorithm is found better suited for DVB-T based PCL system. Figure 3 shows detection based on DVB-T data on CPU. The ARD plot for the same data on GPU is shown in Figure 2 which are found to be identical. The data used was captured near Pisa, Italy.

Table I provides a comparison of CPU and GPU processing time for NLMS algorithm. The order of filter and the data size, which is the total number of samples from both reference and surveillance channel are the major factors in Table I that determines processing time and speed-up factor. Speed-up factor is the ratio of CPU processing time to GPU processing time. Though data size determines the effective speed-up, it is observed that the variation of speed-up factor with data size is minimal. The reason for this is the number of parallel threads executed in parallel, in few of the individual process is equal to the order of the filter and is independent of the data size.

2) Extensive Cancellation Algorithm: ECA (Extensive Cancellation Algorithm) is a signal projection algorithm based on the Colone clutter cancellation algorithm [6]. The algorithm is based upon on building up a reference matrix and shifting it in Doppler. The CPU implementation of the ECA was using the Armadillo [14] library and was extremely time consuming and the scope for GPU acceleration is found crucial at this point.

The reason for the computation intensity of ECA is the calculation of estimation error calculation defined by Haykin [12] in equation 1.

$$e = s_R - X\left(X^H X\right)^{-1} X^H s_R \tag{1}$$

In equation 1, X refers to the clutter subspace matrix [12] and s_R, the received signal and e represents the re-

TABLE I
NLMS Time Consumption-CPU Vs GPU

Data Size (N^o Samples)	Order of the filter	CPU Time (sec)	GPU Time (sec)	Speed-Up Factor
409.6K Samples	100	1.94	0.22	8.85X
819.2K Samples	100	3.96	0.45	8.80X
2048K Samples	100	8.81	1.14	7.72X
409.6K Samples	200	3.63	0.45	8.06X
819.2K Samples	200	7.31	0.91	8.03X
2048K Samples	200	18.46	2.28	8.1X

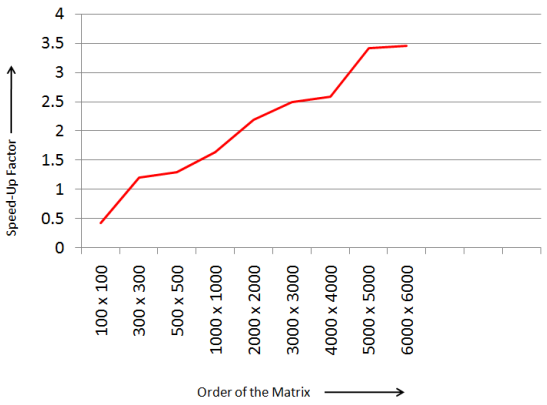

Fig. 4. Speed-up factor for GPU based complex matrix inversion for various matrix sizes.

ceived signal with zero-Doppler component cancelled.The algorithm has a complexity of $O\left[NM^2 + M^2 logM\right]$ [13] where M=(Number of Range bins $\times$ Number of Doppler bins) and N is the number of Data Samples.

The GPU implementation starts with the identification of processes that can be processed using parallel GPU threads. Algebraic operations using large data size are selected for GPU implementation. Based on this strategy, the ECA can be divided into three phases. Features of GPU implementation within each phase are discussed here.

1) Building of clutter subspace matrix X
   a) The calculation of time intervals from the sampling frequency (409.6 kHz used for data capture). The processes is done using a custom kernel to perform the division on real data.
   b) The calculation of elements in matrix X done using complex vector multiplication using cuBlas [15].
2) Calculation of Multiple matrix product $\left(X\left(X^H X\right)^{-1} X^H s_R\right)$
   a) X^H is calculated as the hermitian transpose of the complex matrix X. X has a dimension of (N^o Samples $\times$ (Doppler-bins $\times$ Range-bins)) and hence X^H is of dimension ((Doppler-bins $\times$Range-bins)) $\times$ N^o Samples). Hermitian transpose is accomplished by customising the real matrix transpose code from CUDA SDK with a subroutine for complex matrix inversion.
   b) $\left(X^H X\right)$ is performed using shared matrix multiplication extended for complex matrix multiplication.
   c) $\left(X^H X\right)^{-1}$ was the highest time-consumption portion of processing and is done fully in the GPU.
   d) $X\left(X^H X\right)^{-1} X^H s_R$ is realised with the same kernel designed for use in step (b).
3) Estimation error e [12] is calculated by Complex vector subtraction.

The GPU based hermitian transpose and parallel complex matrix inversion can be used as a standalone functions for other Digital Signal Processing(DSP) applications. Figure 4 shows the speed-up achieved by GPU complex matrix inversion. The comparison is with respect to C++ implementation in single threaded Intel Quad core with 4G RAM. Other processes in ECA, preceding equation 1 are creation of reference matrix X and its Doppler translation. Table II splits the ECA into the major processes involved and illustrates the processing time for each stage.

TABLE II
GPU TIMING – ECA WITH SUB-PROCESSES FOR RANGE BINS=48

Process	Data Size (N^o Samples)	Time(sec)
Reference Matrix Building	160K Samples	1.10
Shifting Reference Matrix in Doppler	160K Samples	1.2
Double Precision Hermitian Transpose	160K Samples	1.86
Double Precision Complex matrix inversion	160K Samples	0.38
Double Precision cuBlas Complex matrix multiplication	160K Samples	0.38-0.42
Double Precision Complex Vector Subtraction	160K Samples	0.28

The Armadillo [14] library also had limitations in increasing the number of range bins and the increase in range bins caused tremendous increase in processing time. The PCL system under deployment is expected to have a range coverage of 100 – 200km and the CPU code was unable to process such large matrix sizes. The GPU implementation solved all the above mentioned problems and the speed up achieved, increased with increase in the number of range bins. Table III compares CPU and GPU timing with increase in the number of range bins.

TABLE III
ECA TIME CONSUMPTION-CPU VS GPU FOR 240K SAMPLES

Range Bins	CPU Time (sec)	GPU Time(sec)	Speed-Up factor
32	17.97	8.38	2.14X
48	32.2	10.18	3.16X
64	57.9	14.35	4.03X
80	98.1	16.81	5.83X
96	172.4	21.10	8.17X

B. Matched Filtering

Matched filtering is the stage of PCL signal processing where, target detection is obtained in an Amplitude-Range-Doppler(ARD) plot. The stage can be considered as the search for the time-delayed and Doppler-shifted versions of the reference signal. This is achieved by correlating the surveillance signal with Doppler-shifted versions of the reference signal to form a bank of filters matched to every possible Doppler frequency of interest. ARD is calculated based on equation 2 defined in Willis book [13].

$$\left|\Psi(\tau,\nu)^2\right| = \left|\sum_{n=0}^{N-1} e(n)d^*(n-\tau)e^{j2\pi\nu n/N}\right| \qquad (2)$$

where Ψ denotes the time-delay of interest and denotes the Doppler-shift of interest [13].

The ARD is calculated using Long Integration Time(LIT) algorithm. In the GPU implementation of matched filtering, three kernels or GPU subroutine are written to realize equation 2 which are discussed in the following sections.

1) Dot-Product Calculation: Calculation of the dot-product of $d^*(n-\tau)$ and the echo signal $e(n)$ in equation 2 is performed in the following steps.

- The conjugated version of the direct signal and echo signal is copied from host to device variables.
- The number of threads per block and blocks per grid are set depending upon GPU initialized using execution syntax [10].
- The GPU kernel set as a global function [10] is called and the vector multiplication is executed in parallel with individual threads in the GPU.
- The dot product is retained in the GPU for Fast Fourier Transform (FFT) processing.

2) FFT Calculation: CUFFT library [16] is used to calculated complex to complex one dimensional in place FFT [16] mentioned in equation 2. The GPU FFT kernel constitutes the following steps.

- The FFT input and output are declared in cufftcomplex data type and the FFT size
- The parameters in the CUFFT plan [16] are FFT size depending upon frequency resolution required.
- When the function is called the plan is executed and the output is obtained as a device variable.

3) Absolute Value Calculation of the ARD: Absolute value calculation is done using a kernel that is similar to dot-product calculation with suitable change in the variables for conversion to a logarithmic scale.

For the ARD Calculation, the DPI and clutter cancelled echo signal is sent to the GPU by a device to device transfer, or within the same device. Optimisation of the code was done by converting all data transfer between the kernels by device to device transfer which further increase the total speed-up. Table IV compares the ARD processing gain in the GPU. It is observed that the speed-up factor increase with increase in the number of range bins and data size up to a factor of 18.67X for 300 range bins and 819.2K Samples. The data size is equivalent to 2 seconds of observation at a data capture frequency of 409.6 kHz. This value can be used as an optimum value for PCL system considering a range resolution [13] of 700 m/range bin giving an effective range of 210km which is sufficient for the expected maximum range.

TABLE IV
ARD TIME COMPARISON-CPU VS GPU

Process-Range Bins	Data Size (N^o Samples)	CPU Time (sec)	GPU Time (sec)	Speed-Up Factor
ARD-100	409.6K Samples	6.38	0.82	7.79x
ARD-100	819.2K Samples	19.70	1.54	12.8x
ARD-100	2048K Samples	52.62	3.68	14.3x
ARD-300	409.6K Samples	20.83	2.48	8.4x
ARD-300	819.2K Samples	81.99	4.39	18.67x
ARD-300	2048K Samples	161.7	9.24	17.5x

We note that the CPU could well be optimised to produce better results using multithreading

Care must be taken when making comparisons between CPUs and GPUs. In most cases, the CPU code is single threaded, and does not exploit the multicore nature of most PCs. It is likely that a very careful implementation of the code on the CPU would produce a significant improvement in performance.

C. Optimisation Measures

- To minimise data transfer between the host and the device, the output at intermediate stages is retained in the device and is used by the next process. Only the control variables for that particular process is transferred from the host. Hence device to device transfer is exploited to the maximum.
- Sequential process like loops with comparatively less iterations are performed in the CPU .This includes delaying reference signal in ARD processing.
- CUFFT [16]can calculated FFT for up to 8 million data in a single instance. But the intention of program is to track the target at each second which corresponds to nearly 400 thousand Samples/ ARD surface. This data rate is used for test deployment using FM illumination. But testing using DVBT data was at a sample rate of 8 Million samples /ARD surface utilising the maximum throughput of the the device.
- Instruction throughput is achieved by utilising the same kernel for few vector operations. But most of the kernels are custom written for that particular operation.

IV. TESTS AND RESULTS

Table I, Table III and Table IV compares the processing time between a CPU and a GPU for different data size and for different processes. The speed-up achieved is comparable to real-time PCL software implementation at the University of Warsaw using Nvidia GTX 280 [17].

Data from Flexible Extensible Radar Simulator (FERS) [18] was used to validate range and Doppler Calculation. The algorithm was tested for different target sizes. The results were satisfactory with clear detections for both big and normal targets.

A. Algorithm Comparison

The Comparison between NLMS and ECA algorithm has to be done with respect to two aspects – Computational Efficiency and Clutter Cancellation Efficiency:

TABLE V
GPU TIME COMPARISON-NLMS AND ECA

Algorithm	Data Size (N^o Samples)	Time(sec)
NLMS	409.6K Samples	0.45
NLMS	2048K Samples	2.28
ECA	409.6K Samples	9.52
ECA	2048K Samples	48.17

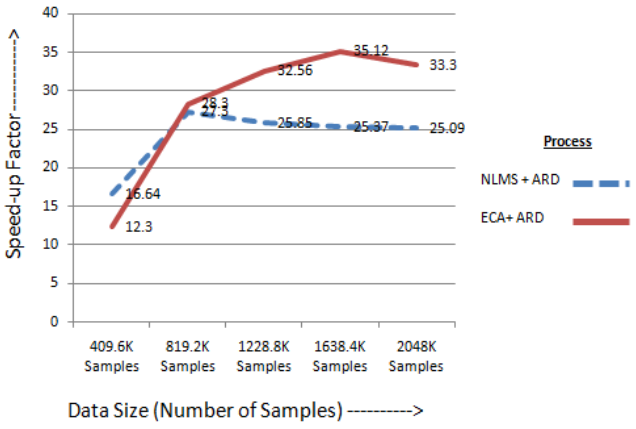

Fig. 5. GPU speed-up factor for PCL signal processing. We note that the CPU (Intel Quad core) could well be optimised to produce better results using multithreading.

1) Computational Efficiency : Table V compares GPU processing time for NLMS and ECA. The NLMS algorithm was able to achieve 1.5 times real time processing in a single GPU. The ECA algorithm is computationally more demanding than NLMS. But the ECA as a batch process is better suited for parallel computation. The number of parallel threads that can be implemented for NLMS algorithm is depended on the order of the filter which extends up to 200 only. But the number of parallel threads in the ECA depends largely on the size of the the captured data and hence will achieve better speed-up factor than NLMS. The graph illustrating total speed-up achieved by GPU implementation for the entire PCL signal processing chain in Figure 5 validates this point. The higher speed-up factor achieved by ECA also provides evidence for the scalability of the algorithm in a multi-GPU platform enabling real-time processing.

The CPU used is an Intel Quad core single threaded with 4GB of RAM. In Figure 5, a NLMS filter of order 100 and ECA with a range bin of 112 is used together with ARD processing for 200 range bins and 300 Doppler bins.

2) Clutter Cancellation Efficiency: Though computationally efficient, under strong clutter environment, the the NLMS algorithm was not effective in preventing the masking effect of clutter on weak targets for FM illumination.

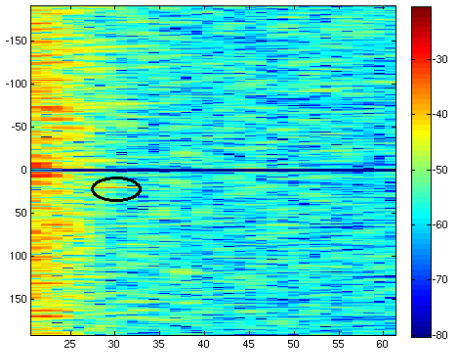

Fig. 6. Target (circled) at (24,30) detected in ECA processing.

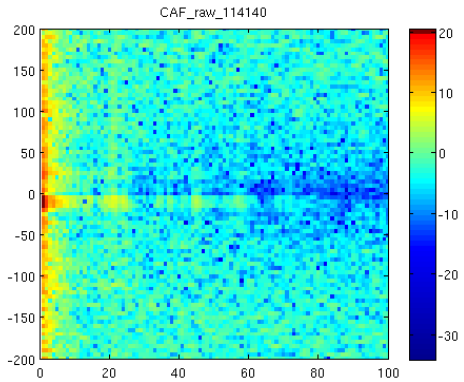

Fig. 7. Target masked by Clutter in NLMS processing.

Figure 6 shows a target detected at 30km in ECA. The source of illumination was FM. NLMS processing was done in the same data set, but the weak target is masked by clutter in the ARD plot shown in Figure 7. Algorithm comparison with respect to long range detection (>100km) is mentioned in Subsection IV-B.

Furthermore, The NLMS filter should be adapted to the source of illumination each time for efficient DPI and clutter cancellation. This reduces the reliability of NLMS algorithm in dynamic testing conditions. ECA is independent of source or change in testing conditions, and hence preferred for PCL system due to the robustness of the algorithm.

B. Test Deployment

Preliminary deployment of the system was done near Cape Town International Airport in December 2010. The source of illumination selected was FM and target detection using both NLMS and ECA algorithms was achieved and shown in Figure 8. The presence of clutter in zero Doppler after the extent of cancellation range-bins can be observed in Figure 8. The processing time for individual processes within ECA corresponds to the data in Table II. The data was processed in sets and split ARD's where obtained. The target under observation is a landing flight and the change in Doppler

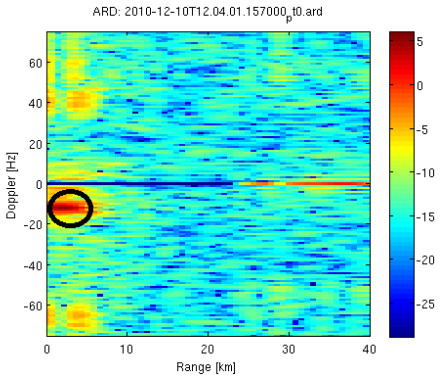

Fig. 8. Target detection (circled) detection using ECA with FM illumination.

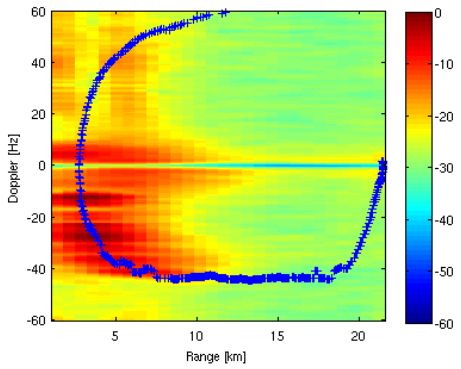

Fig. 9. Overlapping of actual flight path (Line Style '+' and colour blue) on calculated flight path (red) in compound ARD between –12Hz and –42Hz Doppler [3].

and range was observed from the split ARD's. The need for increasing the FFT resolution was identified at this stage. The results were compared to the actual flight path obtained from a DBS transponder on the aircraft and a commercial receiver.

Figure 9 shows the overlapping of actual flight path on the compound ARD, made by overlapping individual ARD frames. The calculated flight path from the PCL system and the actual flight were found to be identical.

An optimized version of the algorithm was used for second phase deployment and the the ARD obtained is shown in Figure 10. Multiple targets were detected at short range (30 – 35km) and the figure also shows target detections at 130km. NLMS algorithm was able to detect the short range targets but the target at 130km was visible only with ECA processing. This provides evidence for the superior clutter cancellation efficiency of ECA. Ongoing work is focusing on comparing the Signal to Clutter Ratio (SCR) of both algorithms.

V. Conclusions and Future Work

The results obtained have to be considered with respect to two aspects: Computation time and reliability. Real-time processing of PCL data with the NLMS algorithm can be realised at less computational cost than the ECA. However, since ECA is based on signal projection, it eliminated the need for adapting the filter to the source of illumination

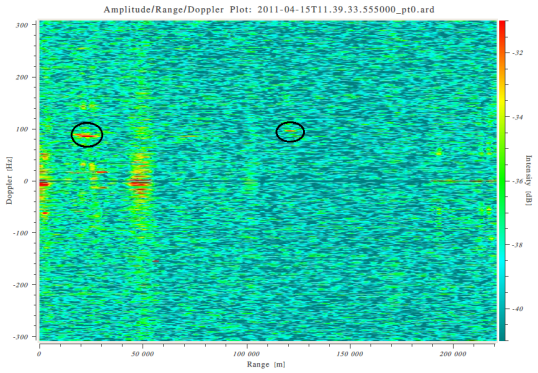

Fig. 10. Multiple targets (circled) detected in short-range (20,90) and long-range (130,90) after clutter cancellation using ECA [3].

used, and is more reliable. The clutter-cancellation ability of ECA was found superior to NLMS when it comes to strong-clutter environment and long-range detection providing evidence for the robustness of ECA. The processing speed-up achieved can be further increased by upgrading to a multi-GPU platform. Ongoing work on this plan, focusses on splitting the captured data into separate GPUs such that scalability of the processing in a cross-platform computing environment can be achieved. Streamlining the ARD to real-time plotting software, improvements in ECA in a multi-GPU environment and improving the adaptivity of the NLMS algorithm is the plan and scope for future work in this field.

Acknowledgment

The authors would like to thank S. Heunis [2] and Yoann Paichard (University of Cape Town) for the work they have done in this field, and supplied the basis for the CPU to GPU conversion. The authors express their gratitude to Gunther Lange and Craig Tong for conducting the ongoing field measurements and continuous support. We would also like to thank the SA National Defence Force and the UCT University Research Committee for ongoing student support.

References

[1] E. Mollick, "Establishing Moore's Law," in *Annals of the History of Computing, IEEE*, vol. 28, no. 3, September 2009, pp. 62–75.
[2] F. S. Heunis, "Passive Coherent Location Radar using Software-Defined Radio Techniques," Master's thesis, University of Cape Town, Private Bag, Rondebosch, 7701, South Africa, May 2010.
[3] C. Tong, M. R. Inggs, and G. E. Lange, "Processing design of a networked passive coherent location system," in *Proceedings of the 2011 IEEE Radar Conference*, May 2011.
[4] M. Ettus, *USRP User's and Developer's Guide*, Ettus Research LLC, Matt Ettus, Ettus Research LLC.
[5] Scott.C.Douglas, "A Family of Normalized LMS algorithms," in *IEEE Signal Processing Letters*, vol. SPL-1, no. 3, 1994, pp. 49–51.
[6] F. Colone, "A multistage processing algorithm for disturbance removal and target detection in Passive Bistatic Radar," in *IEEE Trans. On Aerospace and Electronic Systems*, vol. 45, no. 2, 2009, pp. 698–721.
[7] *Tuning CUDA Applications for Fermi ver. 1.2*, NVIDIA, Santa Clara, CA, July 2010.
[8] *NVIDIA CUDA Reference Manual ver. 3.0*, NVIDIA, Santa Clara, CA, February 2010.
[9] M. R. Inggs, Y. Paichard, and G. E. Lange, "Networked PCL System," in *Proceedings of the 2010 Cognitive Systems with Interactive Sensors (COGIS 2010)*. IET United Kingdom, 2010.
[10] *NVIDIA CUDA Programming Guide*, 3rd ed., NVIDIA, February 2010.
[11] N. Morrison, R. T. Lord, and M. R. Inggs, "The Gauss-Newton Algorithm in Passive Aircraft Tracking using Doppler and Bearings," in *Proceedings of the IET International Conference on Radar Systems (RADAR 2007)*. Institution of Engineering and Technology, October 2007.
[12] S. Haykin, *Adaptive Filter Theory*, 4th ed., T. Kailath, Ed. Prentice Hall, 2002.
[13] N. Willis, *'Bistatic radar' chapter 25 in Radar Handbook1*, 2nd ed., M. Skolnik, Ed. McGrawHill, 1990.
[14] C. Sanderson, *An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments*, NICTA, St Lucia, Australia, September 2010.
[15] *CUDA CUBLAS Library*, NVIDIA, Santa Clara, CA, March 2008.
[16] *CUDA CUFFT Library ver. 1.1*, NVIDIA, Santa Clara, CA, October 2007.
[17] K.Szumski, "Real-Time Software Implementation of Passive Radar," in *Proceedings of the 6th European Radar Conference*, September 2009, pp. 33–36.
[18] M. J. Brooker, "The design and implementation of a simulator for multistatic radar systems," Doctoral Thesis, University of Cape Town – RRSG, Jun. 2008.