Received: 13 November 2021 | Revised: 18 January 2022 | Accepted: 18 February 2022

IET Radar, Sonar & Navigation

DOI: 10.1049/rsn2.12244

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Design of high-speed software defined radar with GPU accelerator

Wenda Li[1] | **Chong Tang**[1] | **Shelly Vishwakarma**[1] | **Karl Woodbridge**[2] | **Kevin Chetty**[1]

[1]Department of Security and Crime Science, University College London, London, UK

[2]Department of Electronic and Electrical Engineering, University College London, London, UK

Correspondence

Wenda Li, Department of Security and Crime Science, University College London, 35 Tavistock Square, Bloomsbury, London, WC1H 9EZ, UK.
Email: wenda.li@ucl.ac.uk

Funding information

Engineering and Physical Sciences Research Council, Grant/Award Number: EP/R018677/1

Abstract

Software defined radar (SDRadar) systems have become an important area for future radar development and are based on similar concepts to Software defined radio (SDR). Most of the processing like filtering, frequency conversion and signal generation are implemented in software. Currently, radar systems tend to have complex signal processing and operate at wider bandwidth, which means that limits on the available computational power must be considered when designing a SDRadar system. This paper presents a feasible solution to this potential limitation by accelerating the signal processing using a GPU to enable the development of a high speed SDRadar system. The developed system overcomes the limitation on the processing speed by CPU-only, and has been tested on three different SDR devices. Results show that, with GPU accelerator, the processing rate can achieve up to 80 MHz compared to 20 MHz with the CPU-only. The high speed processing makes it possible to run in real-time and process full bandwidth across the WiFi signal acquired by multiple channels. The gains made through porting the processing to the GPU moves the technology towards real-world application in various scenarios ranging from healthcare to IoT, and other applications that required significant computational processing.

KEYWORDS

GPU accelerator, signal processing, software defined radar

1 | INTRODUCTION

The applications of radar cover many broad and various areas, for example, the long-range airborne and weather surveillance, short-range target detection, target recognition and classification, etc. These applications have diverse demands, leading to the proliferation of highly specialised radar systems on the same platform, for example, ship, aircraft and others [1]. In addition, these platforms are also equipped with a number of other types of Radio Frequency (RF) sensors, such as communication and navigation systems. Many radar systems were implemented using hardware such as Field programmable gate arrays (FPGAs). However, a software implementation, which uses general-purpose processors is more desirable for its cost-effective, flexibility and fast development. Thus, the approach of sharing a platform with an SDRadar allows these requirements to be met [2]. However, increases in signal sampling rates (signal bandwidth) and the number of receiver channels in Multiple-Input and Multiple-Output (MIMO)/distributed radar systems, mean there are more data need to be processed. For example, the universal software radio peripheral (USRP) family [3] has sampling rate from 20 MHz up to 160 MHz for 2 to 4 channels, the DigitizerNetbox (DigitizerNetbox) [4] can operate at 5 GHz with up to 16 channels, and also Ultra-Wide Band design [5]. Consequently, this increasing requirement in computational power becomes a key parameter to be considered for an SDRadar system.

FPGA and GPU are commonly employed to accelerate computational processing. There are a number of differences between these two architectures, in terms of flexibility, power

consumption and latency etc [6, 7]. The major concern in this work is the amount of onboard memory. For example, commercialised Xilinx FPGA UltraScale product line has a maximum of 133 megabytes block memory [8]. In comparison, gaming GPUs such as NVIDIA GeForce GTX 2060 can offer onboard memory of 6 gigabytes. In this work, we focus on the high sampling rate radar signal which requires extensive data transfer via the peripheral component interconnect (PCI) Express. Therefore, GPU has larger onboard memory and allows data to be transferred and processed simultaneously, making it well-suited for this SDRadar work. Also, GPU-based accelerator means the system can be easily modified with other Software Defined Radio (SDR) devices without heavy development by FPGA-based accelerator.

With the introduction of NVIDIA Compute Unified Device Architecture (CUDA), the parallel processing capabilities of GPU becomes accessible not only for the graphics, but makes financially and technologically accessible. Another advantage of GPU-based accelerator is its ability to free-up the CPU from heavy parallel computing and focus on Data Acquisition (DAQ) and synchronisation control. The utilisation of GPU enables a real-time ability without degradation on sampling rate.

Considering these advantages, many researchers have been working on GPU-based accelerator in various systems to deal with high computational process such as fast Fourier transform (FFT) [9] and correlation [10]. An early work [11] presents an analysis on correlation with GPU acceleration and demonstrates a speed-up factor of 15 compared to CPU only process. Work [10] demonstrates a GPU-accelerated back-projection in reconstruction of Synthetic Aperture Radar. It also shows an impressive runtime with a speed-up factor between 50 and 60. However, this system does not provide a practical solution for GPU-based accelerators in real-time processing. A Raspberry Pi base SDRadar system described in [9] built for passive radar including reference signal reconstruction and two-dimensional FFT (size of 2048 × 512) with on-board CPU and GPU. In this case, GPU only shows a slight improvement by 10% due to the sequential framework design which leads to overhead for single FFT. The limitation of this system is that it operates at a relatively low sampling rate 240 kHz and is not fully compatible with the parallel framework to take full advantage from GPU acceleration. GPU has also been used to accelerate radar image processing [12] with real-time capability. This system hybrids CPU-GPU scheme to process on 40 MHz sampling rate, and can generate images at 8 fps composed of 6000 pixels. In SDRadar systems, using a GPU as an accelerator has been studied broadly in recent years. One of the most representatives is [13], which implemented a real-time Global Positioning System (GPS) receiver with adaptive beam-steering capability using a software-defined approach. This SDRadar offers sufficient computational capability to support 4-element antenna array up to 40Msps (mega samples per second). After that, a testbed [14] for anti-jamming receiver was developed embedded with Space-Time Adaptive Processing and Space-Frequency Adaptive Processing. It further deploys the batch mode to fully take advantage of the parallelism resources provided by GPU. However, both of the two SDRadars are especially developed for beamforming and are limited in flexibility and configurability. A comparison between these works and our system is summarised in Table 1.

In this work, we use the LabVIEW GPU analysis toolkit to interface the CUDA functions and embed into our previous SDRadar system [15] with a new architecture. This is achieved by switching to a parallel framework where DAQ by CPU and signal processing (partly by GPU) are processed separately. The processing speed is significantly improved with the GPU accelerator when compared to CPU-only system. To demonstrate the effectiveness of GPU accelerator in real-time, three SDR devices have been tested to quantify the advantage of proposed concepts in flexibility deployment and processing speed. The hardware specification of these SDR devices is presented in Table 2 and shown in Figure 1.

Compared to previous works [12, 17, and 18], the following contributions are made by this paper:

- This paper presents a robust high-speed SDRadar system that is capable of processing sampling rates of up to 80 MHz without sacrifice from dropping samples. The system runs with a typical passive radar processing including Cross-Ambiguity Function (CAF) and Direct Signal Interference cancellation [19].
- The proposed system can be easily tuned to run with various SDR devices. We have modified it to run with three

TABLE 1 Comparison with other SDRadar systems

Work	[13]	[14]	Our work
Mechanism	Beamforming	Nulling and beamforming	Cross ambiguity function
Configurability	Limited	Partially	Fully adjustable
SDR device	USRP2	ADC	Multiple
No. channel	4	Up to 8	Up to 16
Real-time ability	Partially	Partially	Fully adjustable
Processed sample rate (I/Q)	20Msps	20Msps	80Msps
Application	GPS sensors	Anti-jamming	Near-field monitoring

Abbreviations: ADC, analog-to-digital converter; SDR, Software defined radio.

TABLE 2 Specification of SDR devices in this work

Device	USRP 2920 [3]	USRP 2945 [16]	DigitizerNetbox DN2.593-16 [4]
No of channels (devices)	1 (2)	4 (1)	16 (1)
Connector	Ethernet	PCIe	Ethernet
ADC resolution	16-Bit	14-Bit	16-Bit
Max. sampling rate per channel	20 MHz	80 MHz	80 MHz
Theoretical sampling rate (as a SDR device)	40 MHz	100 MHz	640 MHz
Coherent/non-coherent	Non-coherent	Non-coherent	Coherent
Data type	I/Q	I/Q	Amplitude

Abbreviation: SDR, Software defined radio.

FIGURE 1 Three Software Defined Radio (SDR) devices: (a) USRP 2920, (b) USRP 2945, and (c) DigitizerNetbox (DN)

different devices including the USRP 2920, USRP 2945 and DigitizerNetbox.
- The new system shows a marked improvement in processing speed when comparing to our previous CPU only system [15]. Three SDRadar features (full bandwidth processing, multi-channel and phased-array system) have been implemented to demonstrate the feasibility of GPU accelerator.

The rest of this article is organised as follows. Section II outlines the concepts of our SDRadar system and the associated signal processing for WiFi based passive sensing; Section III presents the design and implementation of GPU-based parallel processing; measured performance and experimental results are shown in Section IV; finally, conclusions from this study are in Section V.

2 | SIGNAL PROCESSING IN SDR

2.1 | Cross-correlation

Cross-correlation evaluates the level of similarity between two functions or signals [20], and is widely used to detect the time delay and Doppler shift for a known transmitted signal and reflected signal from objects. More specifically, according to our previous works [15, 21], cross-correlation consumes the major computational power in signal processing. In such scenarios, the speed of the cross-correlation is crucial to the overall performance of the system. In the time and discrete domains, the definition of cross-correlation between reference (transmitted) signals s_r and surveillance (received) signals s_s can be written as Equations (1) and (2):

$$(s_r \cdot s_s)(\tau) = \int_{-\infty}^{\infty} s_r^*(t) s_s(t+\tau) dt \tag{1}$$

$$(s_r \cdot s_s)[n] = \sum_{m=-\infty}^{\infty} s_r^*[m] \times s_s[m+n] \tag{2}$$

where * represents complex conjugate. The cross-correlation theorem suggested in [22] in discrete form is presented in Equation (3):

$$(s_r \cdot s_s)[n] = \sum_{m=-\infty}^{\infty} IFFT\left(FFT\left(s_r^*[m]\right) FFT\left(s_s[m+n]\right)\right) \tag{3}$$

where FFT is fast Fourier transform and IFFT is inverse fast Fourier transform.

One of the limitations to apply Equation (3) to SDRadar system is the size of s_r and s_s. Considering the sampling rate of 20 MHz (a typical bandwidth of WiFi signal at 2.4 GHz), which means there are 20 M data points that need to be processed for every second. Particularly, the FFT process on a long sequence is very slow. This makes the cross-correlation almost impossible to be processed in real-time. For this reason, batch process has been applied which segments a long sequence s into L short and equal length sequences $s = [s^1, s^2, \ldots, s^L]$. Cross-correlation with batch process can be expressed as:

$$s_r \cdot s_s = \sum_{i=0}^{L-1} IFFT\left(FFT\left(s_r^{i*}\right) FFT\left(s_s^{i}\right)\right) \tag{4}$$

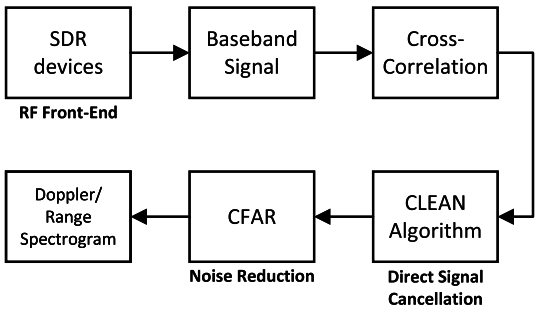


FIGURE 2 Block diagram of proposed SDRadar system

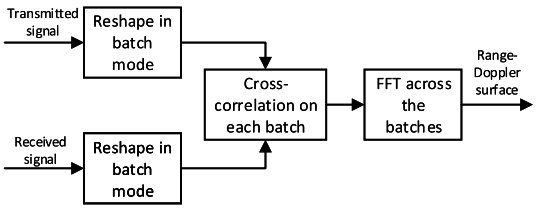


FIGURE 3 Block diagram of Cross-Ambiguity Function (CAF)

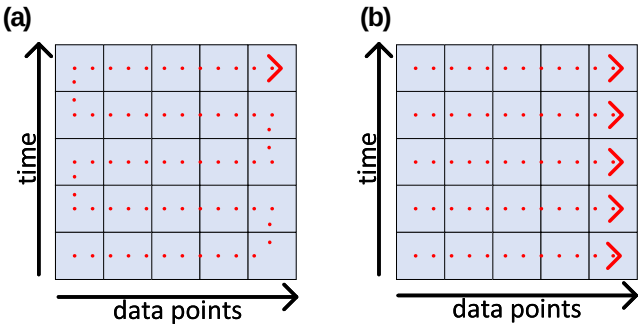


FIGURE 4 A comparison between (a) serial fast Fourier transform (FFT) processing (on CPU) and (b) parallel FFT processing (on GPU)

2.2 | Signal processing in SDR

The block diagram of the SDRadar signal processing is shown in Figure 2. The processing begins with DAQ from SDR devices. The signals from the antenna are digitised and transferred to computer for processing. The CAF calculates the distance and velocity of the target. However, the target, clutter and direct signal are mixed which reduces the sensitivity of SDR. The CLEAN algorithm [19] is used to suppress the clutter and direct signal. Afterwards, a Constant False Alarm Rate has been used to further reduce the noise.

The block diagram of the CAF process is shown in Figure 3. The complete CAF process has two inputs: transmitted signal which is broadcast by the radar system, and received signal which is reflected from target. Both signals are then reshaped into batch mode to avoid the long sequence. Afterwards, cross-correlation Equation (4) is applied to find the relative time delay within each batches. The final step is to carry out another FFT process across the batches. This is to calculate Doppler shift upon the time delay from cross-correlation. The output is a 2D range-Doppler surface S_{CAF}.

CLEAN algorithm shares similar approach to the CAF, while both of its inputs are the transmitted signal. This is to create a self range-Doppler surface S_{self} which is the representative for the direct signal. Then the cleaned range-Doppler surface S_{clean} is calculated as $S_{clean} = |S_{CAF}| - \alpha|S_{clean}|$, where $|.|$ represents the absolute value and α is the scaling factor for S_{clean} in relate to S_{CAF}.

3 | GPU-BASED PARALLEL PROCESSING

When comparing the hardware architectures between GPU and CPU, GPUs are specifically designed for computationally intensive calculations that have high parallelism rates. Consequently, GPUs are designed with more transistors for data processing than data caching and flow control. These differences in hardware architectures determine the different processing speed of signal processing. For operating systems, there is no affect to process the data in GPU memory which also gives better stability. As a result, accelerating the signal processing with GPU is considered a powerful and fast development option for SDRadar systems [10, 17].

CUDA is a general purpose parallel computing platform and programming model that leverages the parallel compute engine in GPUs to solve many complex computational problems in a more efficient way than on a CPU [23]. Moreover, the LabVIEW GPU Analysis Toolkit can access the CUDA function, and DAQ software in LabVIEW to control SDR devices.

A comparison of FFT performed (in batch mode) by CPU and GPU is shown in Figure 4. The X-axis represents data points from SDR devices, and Y-axis represents the time (a batch can be considered as data points from each specific time period). A conventional SDRadar system uses a CPU to perform FFT in a sequential order that sweeps across all data points in each batch as shown in Figure 4a. In this paper, we propose to use GPU to perform simultaneous multiple FFTs upon a plan of data points as presented in Figure 4b. The high capacity of parallel computation is expected to significantly accelerate the cross-correlation of SDR.

3.1 | Mapping the data points to GPU grid

Figure 5 presents the structure of mapping the data points to GPU grid. An SDRadar system with inputs from N channels at S_r sampling rate in T_i integration time, has a data size of $N \times S_r \times T_i$. Within each channel, data points are segmented equally into L batches with a batch length of L_B. Batches from each channel are combined together and downloaded into GPU memory for cross-correlation. In the CUDA framework, a given sequence of instructions is called *kernel*. Each kernel controls a group of *blocks* which process the data in parallel.

The block index (red context in Figure 5) indicates the number of parallel block N_B, where $N_B = N \times L$. To make it simple, we use the processing rate P_r to measure the actual data that processed by the system. The number of blocks, overall data size downloaded by GPU and processing/sampling rate can be expressed as:

$$N \times S_r \times T_i = N_B \times L_B = P_r \tag{5}$$

whereas the left part represents the processed data points in CPU memory, the middle part represents the data points in GPU memory. The processing rate can also be calculated as the product of $N_B \times L_B$. Note that, the value of parameters in Equation (5) is not constant. They vary depending on the SDR device and the maximum throughput of DAQ.

3.2 | System integration

Figure 6 schematically displays the system integration of the GPU-accelerated cross-correlation processing for our SDRadar system. There are three major threads including raw data DAQ (Thread1, T1), multi-channel cross-correlation by GPU (Thread2, T2) and spectrogram generation (Thread3, T3), respectively. The solid arrows describe the main data stream, and the red arrows indicate the data flow inside the GPU.

In T1, the raw RF signal is acquired by the SDR device and transferred into the host computer's memory through the Ethernet/PCIe port (10 GHz). Afterwards, there is an initial preparation to sort the data into 2D matrix. Then, it can be saved into the hard drive for off-line processing or downloaded into the GPU memory through the PCIE X 16 3.0 interface for subsequent online processing. In T2, the whole process is operated for each cross-correlation in parallel by the GPU accelerator, and the obtained data are uploaded to host memory from GPU memory again in the end. In T3, the spectrogram is generated by the CPU from the data loaded in the host memory for graphic user display or storage into the hard drive for further processing for example, activity recognition, identification, etc.

Recall the Equation (5), taking integration time T_i of 1 second as example, the total amount of received data points are $N \times S_r$. The detailed description relating to the cross correlation processing on the GPU is given below:

1. Reshape the received vector data (complex values) into parallel blocks $N_B \times L_B$. This process is done by CPU.
2. Download the batch data from CPU to GPU memory.
3. Complex conjugate is implemented as a GPU function and applied on the received signal $(N_B - L) \times L_B$ in time domain, by changing the sign of imaginary parts of the complex number. This is because the transmitted signal does not do complex conjugate.
4. FFT: forward FFT is performed on all the batches $N_B \times L_B$.
5. The transmitted signal $L \times L_B$ and every $L \times L_B$ received signal, both are complex numbers in the Fourier domain, are multiplied. This gives size of $(N_B - L) \times L_B$.
6. IFFT: inverse fast Fourier transform is performed on all the batches $(N_B - L) \times L_B$.
7. Upload the processed data from GPU to CPU memory.

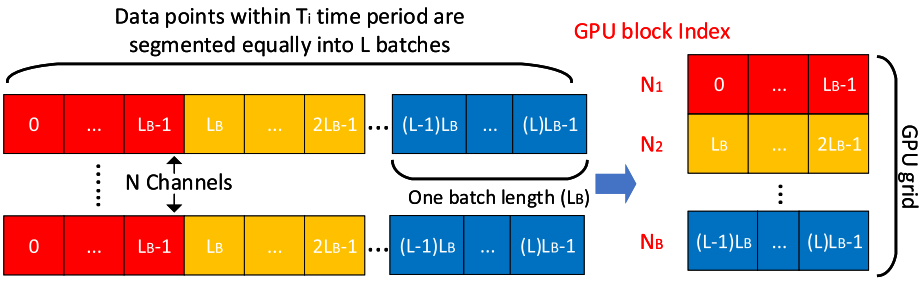


FIGURE 5 Mapping data points on the GPU

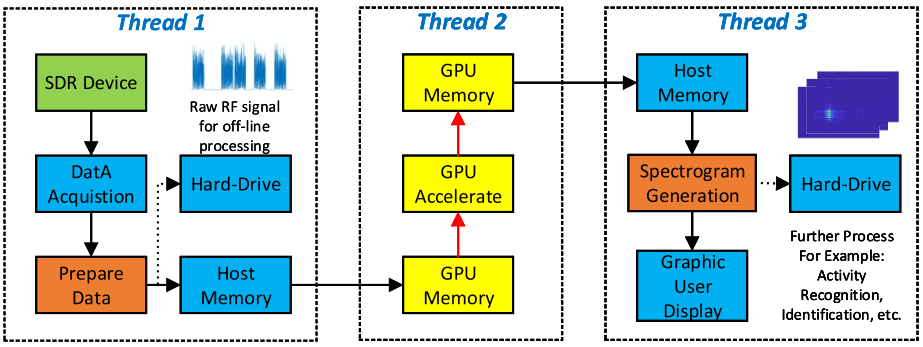


FIGURE 6 Flow chart of GPU-accelerated raw Radio Frequency (RF) data processing

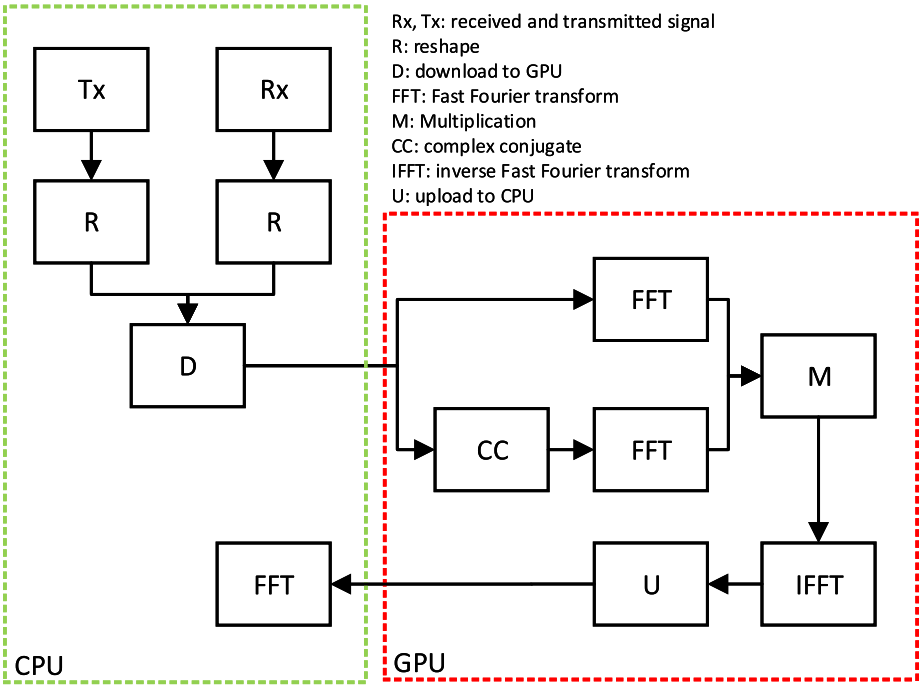


FIGURE 7 Cross-correlation on the GPU

8. Intercept first 30 samples within each batch $(N_B - L) \times 30$. Because the redundant batch length L_B is not necessary in terms of the sensing distance.
9. The final step is to do another short FFT across all the batches on CPU. This is to calculate the Doppler shift upon the time delay.

The implementation of cross-correlation on GPU is shown in Figure 7.

Finally, to ensure different SDR devices are working in a similar fashion, a 'two-process' design has been used, where the DAQ and signal processing run separately as shown in Figure 8. A queue is used to link together DAQ and signal processing and follows First-In-First-Out order. The queue length control is to ensure the queue is not too long to be processed. If the queue length is over a pre-defined value, the queue will be erased. This design minimises the interference of data overflow in DAQ to the real-time signal processing.

3.3 | Performance comparison on CPU and GPU

Here we compare the processing time of cross-correlation on CPU and GPU. The host computer is equipped with an AMD Ryzen 3600@3.59 GHz CPU, an NVIDIA GeForce RTX2060 GPU with 6 GB graphics memory. Random generated data was used in this comparison with data type of CDB (16bytes/element). Upon each test, both processing methods were performed 100 times, and calculated the averaged processing time. To match with the DAQ flow from SDR devices, we simulate 1 second of data but with varied sampling rates, that is

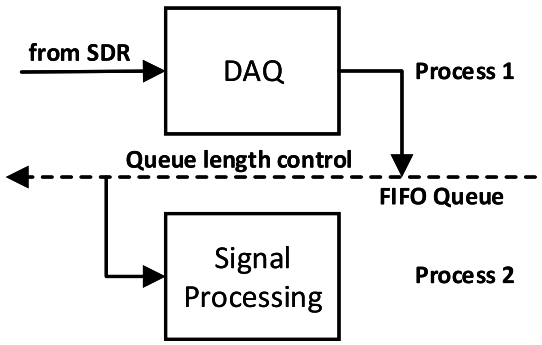


FIGURE 8 Two-process design for SDRadar system

different numbers of data points. Due to the specification of SDR devices, we consider the maximum sampling rate of 50 MHz and up to 16 channels.

Figure 9 shows the performance comparison results between the CPU and GPU based cross-correlation processing upon various data sizes. It can be seen from Figures 9a and b that the processing time of CPU and GPU on either set of cross-correlation increases in proportion to the number of data points, irrespective of the increase in the block number of batch length. However, the processing of the GPU-based method is much faster than the CPU-based processing for any values of block number or batch length, even though the CPU-based processing has been optimised with matrix multiplication. Figures 9c and d show the ratio of processing time of the CPU-based processing over GPU-based processing for each test. It is observed that the processing acceleration of the GPU-based method over the CPU-based method ranged from 2 to 5 times. In addition, both Figures 9c and d clearly

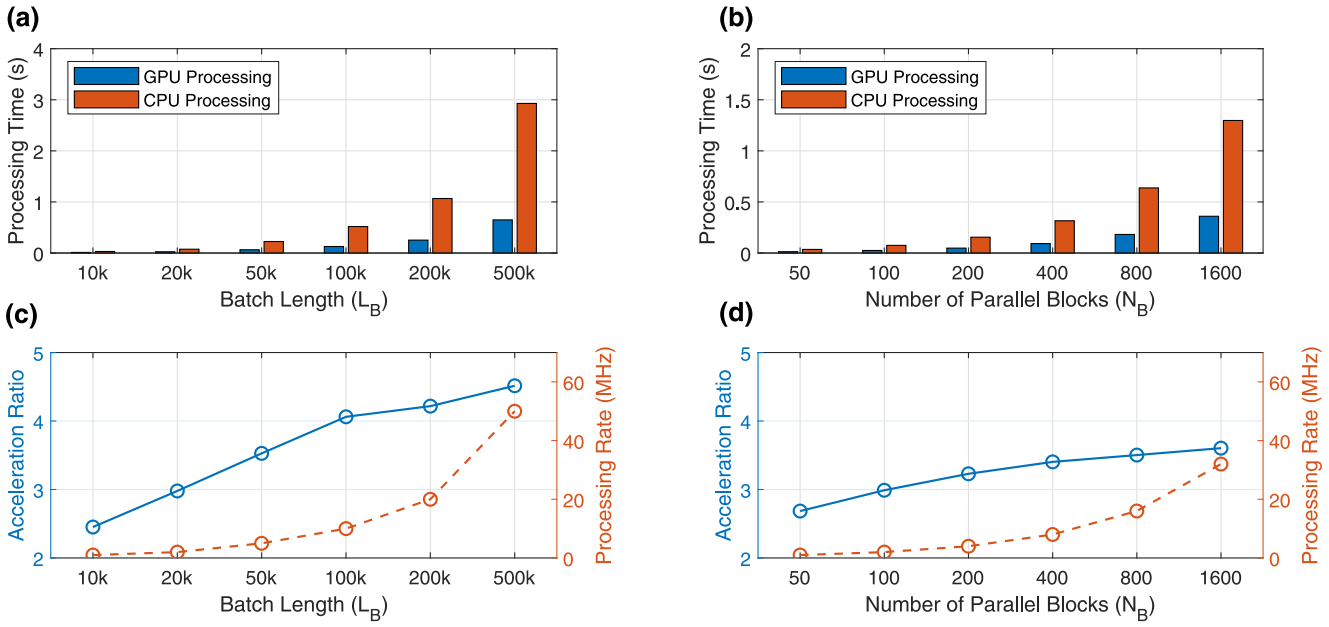


FIGURE 9 Performance comparisons between CPU and GPU based cross-correlation processing on various sizes of batch length and block number. (a) Processing time comparison on 100 blocks with varying batch length. (b) Processing time comparison on varying block numbers with a fixed batch length of 20 k. (c) and (d) GPU–CPU processing acceleration ratio and processing rate corresponding to (a) and (b), respectively

demonstrate that this acceleration ratio increases with the size of batch length and block number. The comparison between Figures 9c and d demonstrates that the GPU accelerates the cross-correlation based on two factors: 1. Faster FFT processing over longer data in each block; 2. More efficient parallel processing due to the highly paralleled core hierarchy in GPU.

The simulation is based on the data from 1 second, which means latency will be induced when the processing time is longer than 1 second, thus not suitable for real-time processing. Besides, CPU also has another heavy load task: DAQ from SDR devices. From Figure 9b, it can be seen that CPU has more than 1 second processing time in the test of 200 k, 500 k batch length and 800, 1600 blocks, while GPU in all tests are less than 1 second. In addition, the overall acceleration ratio in this work is not as significant as in work [18] which ranged from 10 to 30 times faster. The reason is because we also simulated the process of downloading and uploading data to the GPU memory which revealed longer processing times than the cross correlation instead. This has a particular effect in processing time especially with a large data size.

It is worth noting from Figure 9a that the processing time of 10k(L_B) × 100(N_B) can be accelerated to 2 times faster than CPU processing. This is the minimum sampling rate (1 MHz) used in this work; GPU performs 2 times faster than CPU despite it having a short processing time. In comparison, at 500k(L_B) × 100(N_B) (the maximum sampling rate), GPU has contributed more than 4 times acceleration ratio. From Figure 9b, GPU processing also has much shorter processing time when dealing with multiple channels. The processing time of 20k(L_B) × 50(N_B) has been accelerated up to 3 times faster than CPU processing. While, there is only slightly acceleration has been seen at 20k(L_B) × 1600(N_B) at 3.7 times, but still lower than the 1 second of processing time. These results *indicate* an attractive performance delivered by GPU accelerator which is a feas of high sampling rate processing.

4 | SDRADAR APPLICATIONS

A typical radar system requires at least two channels: one transmitter and one receiver. Two USRP 2920 have been used since each of them only has 1 channel. More details about the two USRP 2920 setup can be found in our previous paper [24]. In comparison, USRP 2940 (4 channels) and DigitizerNetbox (16 channels) can fully function as an SDRadar system. Additional channels can be employed for distributed sensing and angle-of-arrival detection.

4.1 | Advantages in sensitivity (USRP 2920)

One of the important measures for SDRadar system is it's ability to detect targets against the background clutter and noise. This is defined by multiple factors, for example, the signal strength, antenna beam pattern, etc. In terms of range-Doppler surface, the sensitivity can be measured as the Peak Signal-to-Noise Ratio (PSNR), where peak represents the pulse relating to the object and noise represents the sidelobes. High PSNR means the object can be easily identified.

Here we provide examples of range-Doppler surface based on WiFi signals. In this measurement, two USRP 2920 s were used as receivers, one channel was connected directly to the WiFi router to measure a 'Reference' signal while the other channel was connected to an antenna to record corresponding

signal reflections from the environment. The antenna was configured as the same direction towards the WiFi router under a stationary environment without any Doppler shifts. Both USRP 2920 were operated at 20 MHz for full bandwidth of WiFi signal at 2.4 GHz channel. To demonstrate the sensitivity performance of SDRadar system in different signal scenarios, three different WiFi frame rates were used as 20 Hz, 100 Hz and 1 kHz per second. The integration time T_i was set at 1 s, the block number was constant at 100 that gives the maximum batch length of 20 $M/100 = 200\ k$. Thus, the processing rate is given as $(2 \times 100) \times 200k = 40\ MHz$, where 2 represents two channels. The SDRadar system made a full CAF processing upon various batch lengths.

Figure 10 presents the range-Doppler surface on three frame rates at different of batch lengths. As expected, frame rate of 1 k has the best performance compared to frame rates of 20 and 100 Hz. This is because of the effective signal depending on the WiFi frame rate as discussed in our previous paper [15]. However, the effective signal may not be captured if the batch length is not of sufficient length, even under a high frame rate. Consequently, improvements can be observed in range-Doppler surfaces with increasing batch length and frame rate. In the worst case, 20 Hz frame rate with 10 k batch length results in significant difficulties with identifying the preferred peak. This situation gets improved when the batch length is increased to 200 k. Range-Doppler surface at 100 Hz frame rate with 200 k batch length, has almost similar performance compared to the same batch length at 1 k frame rate. In comparison, all peaks in 1 k frame rate can easily be distinguished.

Figure 11 presents the PSNR versus batch length from 1 k to 200 k for three frame rates. The red dash line indicates the maximum CPU processing ability at 100 k, however, the GPU accelerator can easily process more than 200 k. The highest PSNR of 20 Hz frame rate can reach up to 17 dB, whereas 100 frame rate and 1 k frame rate both can reach more than 27 dB. PSNR values are getting closer after batch length of 160 k between 100 and 1 k frame rate. This indicates that WiFi signal at 100 Hz can deliver high performance when sufficient data points have been processed, this can be also observed from Figure 10. In addition, in 1 k frame rate, there is only little improvement after batch length of 80 k. Processing on additional data points will not have extra benefit on PSNR. Thus, the trade-off between the amount of processing and PSNR threshold needs to be identified depending on the frame rate.

4.2 | Distributed channels (USRP 2945)

Distributed channels, which measure the object from different angle of aspects, can generate multiple range-Doppler surfaces simultaneously. This can bring spatial diversity in Doppler information and deliver higher recognition accuracy [25]. However, there are many challenges for such systems, for example, the clock/time synchronisation among different channels, also the much higher sampling rate when compared with single channel SDR.

In this measurement, a USRP 2945 was used as the receiver with total 4 channels. Among them, one channel was used to recreate transmitted signal, and other three channels

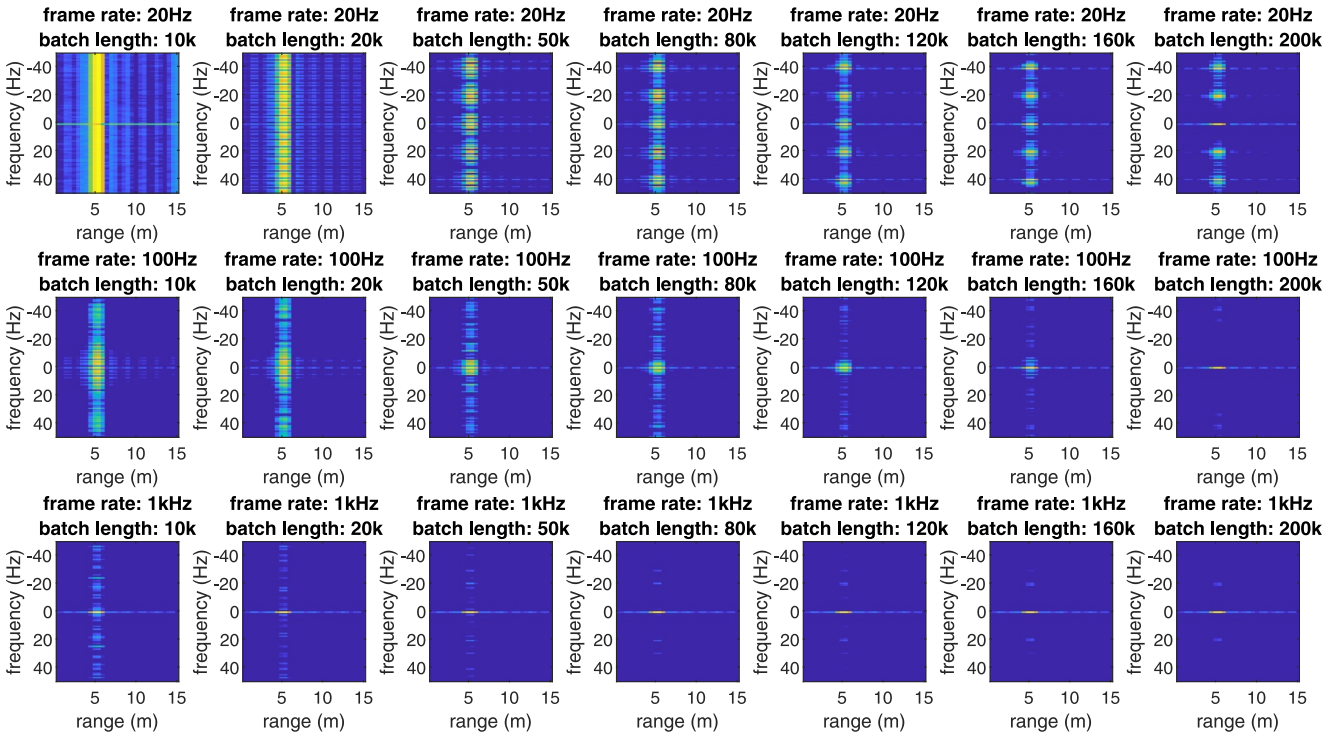
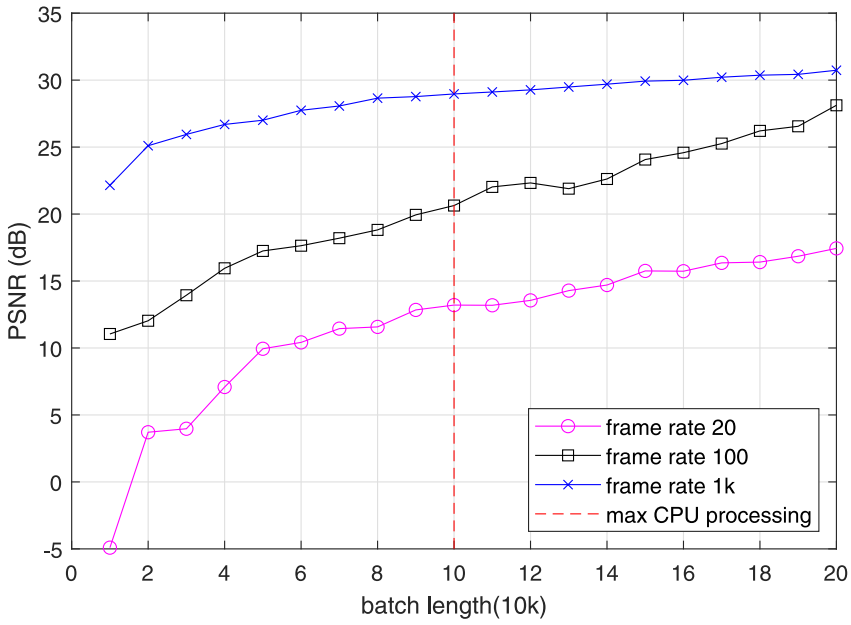


FIGURE 10 Range-Doppler surface for a WiFi signal: rows 1, 2 and 3 illustrates frame rates of 20 Hz, 100 Hz and 1 kHz, respectively. Column 1–7 are batch lengths of 10 k, 20 k, 30 k, 50 k, 100 k, 150 k and 200 k, respectively. The *x*-axis plots range and the *y*-axis plots Doppler

FIGURE 11 PSNR versus batch length



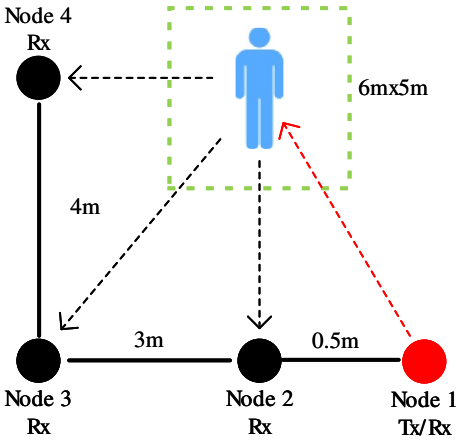


FIGURE 12 Experiment setup: distributed channel

are for the surveillance. A participant was asked to walk around randomly within an area of 6 × 5 m with pauses during the experiment as shown in Figure 12. A WiFi router was aligned with the walking path and marked as 0°. Afterwards, three antennas (one for each channel) were set to different angles towards the walking path to demonstrate the Doppler signatures at different angles. The WiFi router was fixed at 100 frame rate, SDRadar system had been set up with constant parameters as 100 block number and 20 MHz of sampling rate. The full size of processed data point is $(4 \times 100) \times 200k = 80\ M$.

Figure 13 shows a 30 s Doppler spectrogram for walking, captured at different angles. Because the USRP 2945 has three surveillance channels, we measure the walking activity repeatedly for three times. It can be seen that the real-time Doppler record for each period of walking can be distinguished from the others. Doppler signatures are varied in each node due to the variations in monitoring angle. This difference provided by spatial diversity is very important for many machine learning tasks to improve their accuracy like activity recognition, localisation, people counting etc. Additionally, unlike the CSI-based systems [22], the phase noise of our SDRadar exhibits better stability and can more easily be extracted to provide an indication of target direction, where the positive pulses represent the person moving towards the antenna and negative pulses represent away from the antenna. Moreover, we can also observe the micro-Doppler caused by limbs movement, which indicates the system is very sensitive even for small movements.

Afterwards, we record the processing time within each step to demonstrate the actual time spent by the GPU and CPU during real-time processing. Three processing rate tests had been conducted including CPU-only at 20 MHz, and GPU at both 20/80 MHz. Based on observations, CPU at 20 MHz and GPU at 80 MHz are the maximum processing rates that can be operated without time latency on current hardware. We run 100 s of processing for each test, and record the average processing time.

Table 3 summarises the three tests with the processing time for each step. As it can be seen, the CPU at 20 MHz reaches 1329.3 ms in overall time which is close to that in GPU at 80 MHz at 1389.5 ms. Also, GPU at 20 MHz has an overall time of 654.4 ms which is only half to the CPU-only processing time. These results indicate our GPU accelerator can improve the processing rate up to four times than the CPU-only SDRadar system. Reshape process is to convert the long vector data into 2D matrix which depends on the size of data. The highest processing time for CPU-only are FFT and IFFT due to the serial processing. In comparison, GPU has much lower processing time in FFT and IFFT in both 20 and 80 MHz. There are additional download and upload processes for GPU to transfer the data with CPU memory. However, GPU are still faster than the CPU-only processing even when including these extra steps.

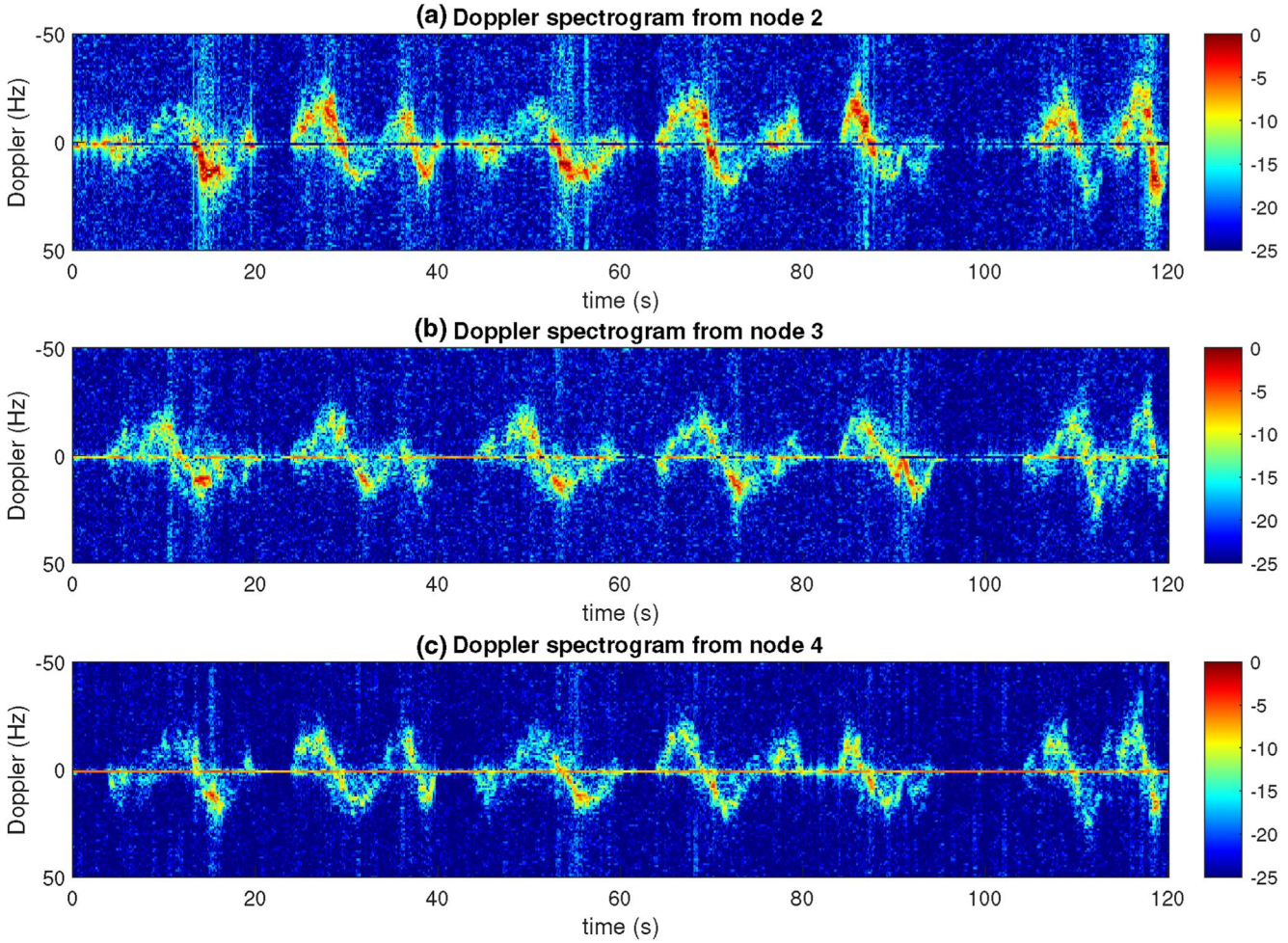


FIGURE 13 Doppler spectrogram captured by 3 distributed receivers at different angle (random direction walking)

TABLE 3 CPU & GPU real processing time comparison

Device	USRP 2945(CPU)	USRP 2945(GPU)	USRP 2945(GPU)
Channel	4	4	4
Sampling rate (S_r) per channel	5 MHz	5 MHz	20 MHz
Processing rate (P_r)	20 MHz	20 MHz	80 MHz
Block number (N_B)	400	400	400
Batch length (L_B)	50k	50k	200k
Processing time (ms) for 1 s of data			
Reshape	287.5	258.6	567.5
Download to GPU	0.0	74.2	125.7
FFT	467.2	54.6	152.8
IFFT	380.8	54.9	148.7
Upload to CPU	0.0	40.7	129.8
Others	193.8	171.4	265.0
Overall	1329.3	654.4	1389.5

Abbreviations: FFT, fast Fourier transform; IFFT, inverse fast Fourier transform.

4.3 | Angle finding (DigitizerNetbox)

The 16-channel DigitizerNetbox was originally designed for waveform spectrum analyser, whereas we use it as a MIMO-SDR. We integrated a phased array antenna to the DigitizerNetbox and performed an Angle-of-Arrival (AoA) analysis over the measurements from all 16 channels. Let the RF signal received at ith channel be $s_i(t)$, the sampling data from DigitierNetbox can be written as $S(t) = (s_1[t]\ s_2[t]\ \ldots\ s_i[t])$, $i \leq 16$. For T period of sampling time, AoA is calculated as the sum of the signal amplitude from the signal source as $\theta(t) = FFT\left(\sum_{t=0}^{T-1} S(t)\right)$. Since there is only a single FFT process, we slightly modified the architecture shown in Figure 7 by removing the multiplication and second FFT process. Here, the AoA analysis aims to search the direction of the incoming signals by the signal source.

Due to limitations of the Ethernet cable creating a data flow bottleneck, we reduced the DAQ rate to 3 iterations per second (96 M data points received every second). The sampling rate was set at 80 MHz for all 16 channels. We ran the system with and without GPU accelerator to show the difference in angular detection. To ensure the system remains in real-time processing, only the CPU was running at a block number of 0.4 k and batch length of 20 k (8 M data points processed every iteration), while GPU accelerator was running at a block number of 1.6 k and batch length of 20 k (32 M data points processed every iteration). According to the angle resolution $\Delta\theta = \frac{D}{\lambda}$ where D is the size of antenna, it is expected that 16-channel system should provide 4 times higher resolution than the 4-channel system.

In this measurement, a WiFi access point (AP) was located at 50° towards the phased array antenna at a distance of 3 m Figure 14 presents an AoA plot for a signal source using data from 4-channel processed by CPU (as the baseline) and data from 16-channel systems processed by GPU. As expected, the angular resolution has been largely improved by the 16-channel with a clear peak at 50°, whereas the 4-channel system gives a much coarse estimation. This indicates the performance gain using a GPU accelerator can sufficiently improve the angular resolution at no additional cost on the computing unit.

5 | CONCLUSIONS

This paper presents a high-speed design for an SDRadar system by using a GPU accelerator to speed up the cross-correlation process. The idea is that CPU can handle raw Data DAQ and post-processing which are non-parallel threads, while GPU can handle the parallel threads involving FFT process. The proposed GPU accelerator demonstrates high flexibility and extensibility, and is able to work with three different SDR devices. Experimental results show that the proposed GPU accelerator can speed up the system by up to four times than the CPU-only system. There are significant improvements in PSNR (Figure 10) and angular resolution (Figure 14) by using the GPU accelerator to process more samples.

Future work will focus on the joint FPGA and GPU implementation for ultra-speed SDRadar systems. This could save more computational power from the CPU and reduce the sampling rate from computer side as some processing can be performed by FPGA. It is envisioned that such systems could be a solution for many industry-based radar applications and can be compatible with AI systems for mission-critical applications that require very low-latency, such as autonomous vehicles and manufacturing operations.

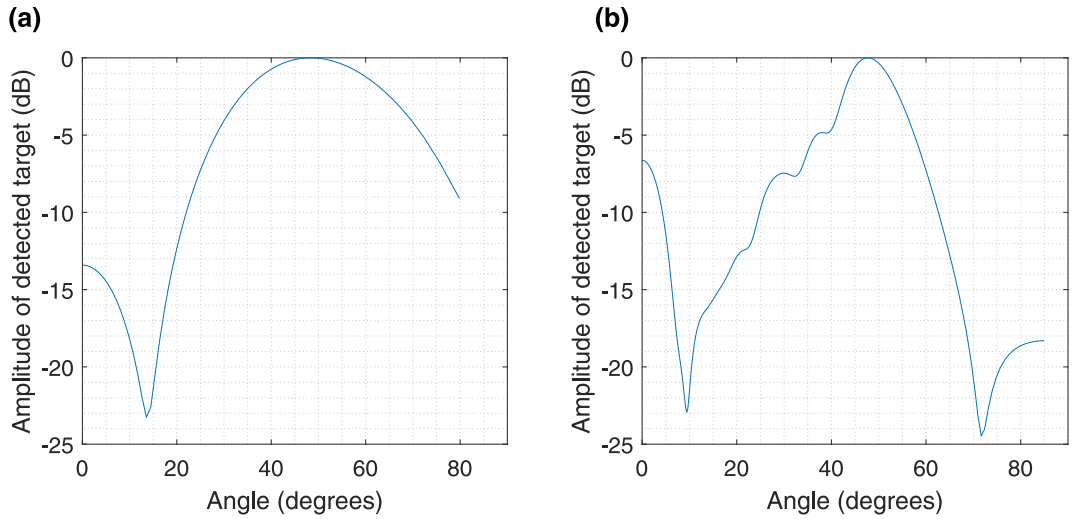


FIGURE 14 Performance gain: angle-of-arrival plot for a signal source measured at 50° to the antenna, (a) data from 4-channel and processed by CPU and (b) data from 16-channel and processed by GPU

ACKNOWLEDGEMENT

This work is part of the OPERA project funded by the UK Engineering and Physical Sciences Research Council (EPSRC), Grant No: EP/R018677/1.

CONFLICT OF INTEREST

All authors listed in this paper declare that they have no conflicts of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Wenda Li https://orcid.org/0000-0001-6617-9136
Shelly Vishwakarma https://orcid.org/0000-0003-1035-3259

REFERENCES

1. Debatty, T.: Software defined radar a state of the art. In: 2010 2nd International Workshop on Cognitive Information Processing, pp. 253–257. IEEE (2010)
2. Jondral, F.K.: Software-defined radio—basics and evolution to cognitive radio. EURASIP J. Wirel. Commun. Netw. 3, 652784 (2005)
3. Ni usrp 2920. [Online]. https://www.ni.com/en-gb/shop/hardware/products/usrp-software-defined-radio-device.html
4. Digitizernetbox. [Online]. https://spectrum-instrumentation.com/en/digitizernetbox
5. Anderson, C.R., et al.: Analysis and implementation of a time-interleaved adc array for a software-defined uwb receiver. IEEE Trans. Veh. Technol. 58(8), 4046–4063 (2009)
6. Nurvitadhi, E., et al.: Accelerating binarized neural networks: comparison of fpga, cpu, gpu, and asic. In: 2016 International Conference on Field-Programmable Technology (FPT), pp. 77–84. IEEE (2016)
7. Fowers, J., et al.: A performance and energy comparison of convolution on gpus, fpgas, and multicore processors. ACM Trans. Archit. Code Optim. 9(4), 1–21 (2013)
8. Xilinx: [Online]. UltraScale Architecture and product ata Sheet: Overview
9. Moser, D., et al.: Design and evaluation of a low-cost passive radar receiver based on iot hardware. In: 2019 IEEE Radar Conference (RadarConf), pp. 1–6. IEEE (2019)
10. Fasih, A., Hartley, T.: Gpu-accelerated synthetic aperture radar back-projection in cuda. In: 2010 IEEE Radar Conference, pp. 1408–1413. IEEE (2010)
11. Gembris, D., et al.: Correlation analysis on gpu systems using nvidia's cuda. J. Real. Time. Image. Process. 6(4), 275–280 (2011)
12. Garcia-Rial, F., Ubeda-Medina, L., Grajal, J.: Real-time gpu-based image processing for a 3-d thz radar. IEEE Trans. Parallel Distr. Syst. 28(10), 2953–2964 (2017)
13. Seo, J., et al.: A real-time capable software-defined receiver using gpu for adaptive anti-jam gps sensors. Sensors. 11(9), 8966–8991 (2011)
14. Xu, H., Cui, X., Lu, M.: An sdr-based real-time testbed for gnss adaptive array anti-jamming algorithms accelerated by gpu. Sensors. 16(3), 356 (2016)
15. Li, W., et al.: Passive wifi radar for human sensing using a stand-alone access point. IEEE Trans. Geosci. Rem. Sens. (2020)
16. Ni usrp 2945. [Online]. https://www.ni.com/en-gb/support/model.usrp-2945.html
17. Zhang, C., Yang, Q., Deng, W.: High frequency radar signal processing based on the parallel technique, 2015
18. Li, J., Xiao, Y.: Gpu accelerated parallel fft processing for Fourier transform hyperspectral imaging. Appl. Opt. 54(13), D91–D98 (2015)
19. Chetty, K., Smith, G.E., Woodbridge, K.: Through-the-wall sensing of personnel using passive bistatic wifi radar at standoff distances. IEEE Trans. Geosci. Rem. Sens. 50(4), 1218–1226 (2011)
20. Yoo, J.-C., Han, T.H.: Fast normalized cross-correlation. Circ. Syst. Signal Process. 28(6), 819–843 (2009)
21. Li, W., et al.: Physical activity sensing via stand-alone wifi device. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2019)
22. Cross-correlation theorem. [Online]. https://mathworld.wolfram.com/Cross-CorrelationTheorem.html
23. Cuda. [Online]. NVIDIA CUDA C Programming Guide
24. Li, W., Tan, B., Piechocki, R.J.: Wifi-based passive sensing system for human presence and activity event classification. IET Wirel. Sens. Syst. 8(6), 276–283 (2018)
25. Fioranelli, F., et al.: Feature diversity for optimized human micro-Doppler classification using multistatic radar. IEEE Trans. Aero. Electron. Syst. 53(2), 640–654 (2017)

How to cite this article: Li, W., et al.: Design of high-speed software defined radar with GPU accelerator. IET Radar Sonar Navig. 1–12 (2022). https://doi.org/10.1049/rsn2.12244