MIMO RADAR HARDWARE ACCELERATION WITH ENHANCED RESOLUTION

ACCÉLÉRATION D’UN RADAR MIMO À RÉSOLUTION AMÉLIORÉE

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Eric J.A.G. Pitre, BEng
Lieutenant (Navy)

In partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Electrical Engineering

June, 2022

Acknowledgements

I would first like to acknowledge the mentorship and expertise provided to me by my academic supervisors. This thesis covers several facets of Electrical Engineering and having expert opinion on specific topics was invaluable. Dr Joey Bray, Dr Vincent Roberge, and Dr Mostafa Hefnawi provided timely advice regarding radar and RF systems, parallel processing, and MIMO techniques respectively. Additionally, the quality of this work could not have been achieved without their feedback and support.

I would also like to thank Dr. Germain Drolet and LCdr Robert Gilpin for providing me with the required tools and advice regarding the MIMO radar which was built at RMC. Troubleshooting and system understanding would have been much more difficult without their help.

Finally, I would like to thank my fiancée, Tanya, who had to share her home office with me for the past two years. The pandemic definitely brought its challenges, but having to work from home meant that she was the first one to hear about my technical problems and about how I wanted to fix them. You were there for me when I needed it the most, and I truly appreciated that.

Abstract

Classical radars steer their main beam across their desired Field of View (FOV) by physically moving the antenna or by adjusting the phase of the elements of an array. It takes time for a narrow beam to cover the entire FOV, which will in turn affect the refresh rate of the system. Recent research has seen efforts to implement Multiple Input Multiple Output (MIMO) techniques to radar. By using multiplexing techniques, the MIMO radar can illuminate the whole search sector at once and perform beamforming on receive. At the cost of higher computational complexity and longer dwell times, to compensate for lower Signal to Noise Ratios (SNR), the MIMO radar can simultaneously scan the entire FOV and thus increase the refresh rate of the radar.

Due to the increased signal processing requirements, MIMO radars have difficulty operating in real-time as the computations can take several seconds to execute. The computation cost is somewhat mitigated by using efficient algorithms such as the Fast Fourier Transform (FFT) to solve for the range, velocity, and Direction of Arrival (DOA) of the echoes. However, the FFT is shown to have less than optimal resolution when compared to other signal processing tools.

In this thesis, parallel implementations of MIMO signal processing algorithms on a Graphics Processing Unit (GPU) are proposed to allow for near real-time imaging of the field of view. In addition, two algorithms (the Chirp Z Transform and the Bartlett Beamformer) are proposed to replace the commonly used FFTs and improve the range and angular resolutions.

The result of this work yields a range resolution improvement of 24.58% and an angular resolution improvement of 24.48% when compared to the baseline FFT method. Executed in parallel, the solution provides a speed up of 454.2x on the GPU. A signal processing time of $\sim$324 ms was achieved for a Coherent Processing Interval (CPI) of 308 ms, enabling near real-time operation of the radar. Additionally, a correction method which enables the imaging of near-field target is proposed and verified.

Resumé

Les radars classiques dirigent leur faisceau principal à travers leur champ de vision en déplaçant l’orientation de l’antenne ou en ajustant la phase des éléments d'un réseau. Il faut du temps pour qu’un faisceau étroit couvre la totalité du champ de vision, ce qui affecte le taux de rafraîchissement du système. Récemment, des travaux de recherches mettent des efforts à implémenter des techniques à entrées multiples et sorties multiples (MIMO) au radar. En utilisant des techniques de multiplexage, le radar MIMO peut éclairer l’ensemble du secteur de recherche simultanément et effectuer la formation de faisceaux en réception. Cependant, cet avantage vient avec un coût de calcul plus élevé.

Du aux exigences du traitement de données, les calculs peuvent prendre plusieurs secondes à exécuter. Les radars MIMO ont donc de la difficulté à fonctionner en temps réel. Le coût de calcul est quelque peu atténué par l’utilisation d'algorithmes efficaces tels que la transformée de Fourier rapide (FFT) pour résoudre la portée, la vitesse et la direction d’arrivée des échos. Cependant, il y a des algorithmes qui génèrent de meilleures résolutions que l’analyse FFT.

Dans cette thèse, des implémentations d'algorithmes parallèles des systèmes radar MIMO sur un unité de traitement graphique sont proposées afin de réduire le temps de calcul et de permettre une imagerie en temps quasi réel du champ de vision. En plus, deux algorithmes sont proposés pour remplacer les FFT couramment utilisées dans le but d’améliorer la résolution de portée et la résolution angulaire. Les deux algorithmes sont basés sue la transformée en Z du signal chirp (CZT : Chirp Z Transform) et sur la méthode de Bartlett pour la formation des faisceaux (Bartlett Beamformer).

Le résultat de ce travail donne une amélioration en résolution de portée de 24,58% et une amélioration en résolution angulaire de 24,48% par rapport à la méthode FFT. Exécutée en parallèle, la solution offre une accélération de 454,2x sur l’unité de traitement graphique. Un temps de calcul d'environ 324 ms a été atteint pour un intervalle de traitement cohérent (CPI) de 308 ms, permettant un fonctionnement en temps quasi réel du radar. De plus, une méthode de correction qui permet l'imagerie d'une cible à courte portée est proposée et vérifiée.

Contents

List of Tables

List of Figures

Acronyms

ADC	Analog to Digital Converter
AESA	Active Electronically Steered Array
AM	Amplitude Modulation
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CDMA	Code Division Multiple Access
CPI	Coherent Processing Interval
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CW	Continuous Wave
CZT	Chirp Z Transform
DIT	Decimation in Time
DFT	Discrete Fourier Transform
DOA	Direction of Arrival
DSP	Digital Signal Processing
DSSS	Direct Spectrum Spread Spectrum
EM	Electromagnetic
EP	Electronic Protection
ESPRIT	Estimation of Signal Parameters via Rotational Invariance Technique
EW	Electronic Warfare
FDMA	Frequency Domain Multiple Access
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
FHSS	Frequency Hopping Spread Spectrum
FMCW	Frequency Modulated Continuous Wave
FOV	Field of View
FPGA	Field Programmable Gate Array
GPGPU	General Purpose GPU
GPU	Graphics Processing Units
HF	High Frequency
HPBW	Half Power Beamwidth

IAA	Iterative Adaptive Approach
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
IMU	Inertial Measurement Unit
ITU	International Telecommunication Union
LO	Local Oscillator
MA	Multiple Access
MIMO	Multiple Input Multiple Output
MUSIC	Multiple Signal Classification
MVDR	Minimum Variance Distortionless Response
OMP	Open MP
OS	Operating System
OSF	Oversampling Factor
PAR	Phased Array Radar
PRI	Pulse Repetition Interval
PRF	Pulse Repetition Frequency
RADAR	Radio Detection and Ranging
RCS	Radar Cross Section
RMC	Royal Military College of Canada
RRE	Radar Range Equation
SAR	Synthetic Aperture Radar
SDMA	Space Division Multiple Access
SDR	Software Defined Radio
SFP+	Small Form-Factor Pluggable Plus
SFU	Special Function Unit
SIMD	Single Instruction Multiple Data
SLA	Sparse Linear Array
SLL	Side Lobe Level
SM	Streaming Multiprocessor
SNR	Signal to Noise Ratio
SP	Streaming Processor
SSS	Single Snapshot
T/R	Transmit and Receive
TDMA	Time Domain Multiple Access
TOF	Time of Flight

UAV	Unmanned Aerial Vehicle
UHD	USRP Hardware Driver
ULA	Uniform Linear Array
USRP	Universal Software Radio Peripheral

1 Introduction

1.1 Background

Significant advancements have been made over the years in the fields of telecommunications and radar. With the implementations of 5G wireless networks, developers have leveraged the benefits of Massively Multiple Input Multiple Output (MIMO) to dramatically increase the throughput and robustness of communication systems [1]. Despite the advantages and scalability of MIMO communication systems, the technology comes with increased processing requirements [1]–[2]. Currently, FPGA and hardware solutions are implemented to meet these computing demands and researchers are looking into using other technologies, such as Graphics Processing Units (GPU), to perform the computations [1]–[2]. MIMO technology has piqued the interest of radar researchers as it could yield new capabilities when used with Software Defined Radios (SDR) or as an addition to current radar systems [3]. Many papers have been published regarding the implementation and testing of MIMO radars but, just like for communication systems, there is a noticeable increase in processing demands [3]. The added computations affect the scalability and real-time performance of MIMO radars [3]. MIMO radars come in 2 broad categories depending on the location of their antennas, where they are considered either co-located [4] or distributed [5]. The work contained in this thesis specifically applies to co-located MIMO radar.

MIMO, in the context of radar, can be interpreted as a different way of scanning a Field of View (FOV). Instead of relying on mechanically steering an antenna or adjusting the phase of an array to transmit and receive in a particular direction, the MIMO radar uses orthogonal waveforms to illuminate the entire FOV at once, using multiplexing techniques, and performs beamforming on receive [3]–[4]. Although the computational demands are greater, the MIMO radar can simultaneously evaluate echoes across the FOV without having to go through the lengthy scanning process required by classical surveillance radars [3], [7]. Using Software Defined Radios (SDR), a MIMO radar was built and tested at the Royal Military College of Canada (RMC) in 2021 which could operate in different multiplexing modes and yielded a relatively high refresh rate when compared to similar works [7]. However, the signal processing time was still significant, taking approximately 1.25 seconds. The radar took approximately 308 milliseconds to collect the 256-chirp Coherent Processing Interval (CPI), meaning that the signal processing takes approximately 80% of the time between radar frames [7]. Additionally, range and angular resolution measurements were taken in [7] to quantify the performance of the radar. In general, the resolution results were larger than what we expect from theory.

1.2 Problem Statement

Many recently reported MIMO radar prototypes suffer from long computation times [7]–[10]. Since the potential of the MIMO radar lies in reducing the required time to scan a FOV, this added computation time diminishes the advantage of its utilization and might not enable the radar system to run in “real-time”. Additionally, a trade-off exists between choosing computationally efficient algorithms for speed and more complex algorithms for better performance in resolution. More research is required to find computationally efficient algorithms which provide high resolution results [6].

As previously stated, the 1.25 second processing time of the RMC MIMO radar is quite long compared to its CPI of 308 milliseconds. Additionally, as shown in Table 1-1, the measured resolutions are larger than the theory would predict, even when considering the FFT window used (Blackman-Harris), which is especially true for the angular resolution.

Resolution	Desired Resolution	Measured	Expected results for rectangular FFT window
Range	0.75 m	1.75 m	0.875 m
Velocity	0.1 m/s	0.2 m/s	0.1 m/s
Angular	2°	10.5°	5.25°

Table 1-1. Measurement Results, summarized from [7]

1.3 Motivation

The motivation for this thesis is to realize a high refresh rate MIMO radar which leverages the computational power of parallel processing to the previous work done at RMC. Furthermore, algorithms that improve the resolution will also be explored and will leverage the parallel processing capability. If a successful parallel implementation of high-resolution algorithms can be achieved, RMC’s MIMO radar will be able to provide higher quality results at a higher rate than before.

The large data cube generated from the SDRs requires independent and repetitive algorithms to be performed across all time samples, pulses, and MIMO channels thereby providing significant parallelization opportunities which is well suited for parallel programming techniques.

1.4 Thesis Statement

Multicore CPU and GPU parallel processing techniques will be investigated to accelerate the high computational demands of MIMO radar signal processing. Additionally, the Chirp Z Transform and the Bartlett Beamformer will be evaluated to improve the range and angular resolutions. Resolution measurements and radar frame execution times will then be compared against the current version of RMC's MIMO radar.

1.5 Methodology

The first step is to create a simulated MIMO radar environment which can generate multiple target echoes for any given range, bearing, speed, and heading combination, given a transmit and receive array configuration. The output of the simulated environment will be the In-Phase and Quadrature (I&Q) voltage time samples for all MIMO channels.

Once the radar echoes have been properly modeled, the baseline signal processing algorithm will be written, which includes the range, doppler, and angular FFTs as well as an algorithm to generate the range-doppler and range-bearing diagrams to display to an operator. Figure 1-1 shows an example of a range-doppler diagram with a visible target at ~50 meters and a radial velocity of ~4 meters per second.

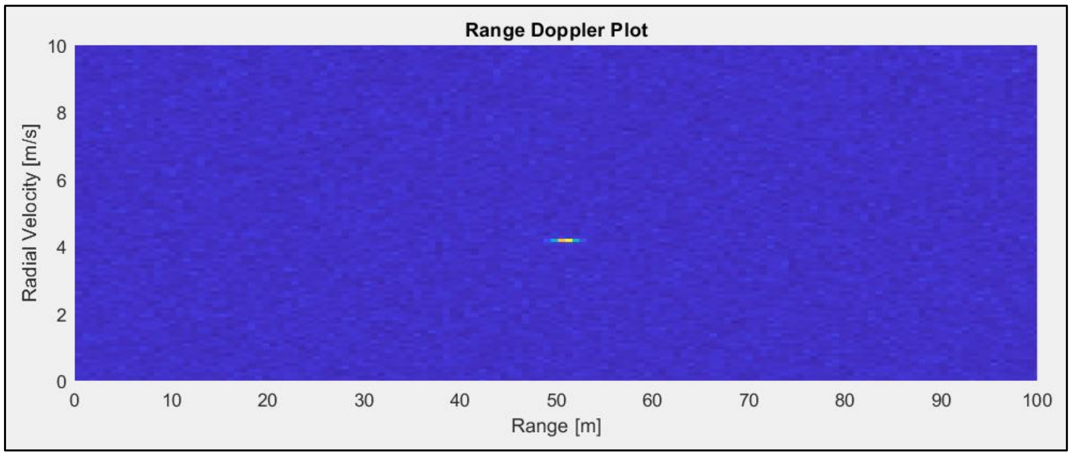

Figure 1-1. Range-Doppler Diagram Example

The next step will be to replace the existing range FFT by the Chirp Z Transform and the existing angular FFT by the Bartlett Beamformer for the high-resolution mode. Within the simulation environment, point targets are used to perform resolution measurements. Angular and range resolutions are then compared between the FFT mode and the high-resolution mode.

Subsequently, the signal processing is parallelized on multicore CPU and on the GPU. The time taken to perform each step of the computation is then measured and compared with its sequential implementation in order to quantify the computation acceleration, which will be done for both the existing 3D FFT method and the high-resolution method.

Finally, resolution measurements in range and in bearing will be taken. Quantifying the success of the high-resolution algorithm will be done by comparing the resolution measurements of the high-resolution method against those of the existing method.

1.6 Thesis Organization

This thesis is organized into 5 chapters. Chapter 2 provides a literature review of MIMO radar and parallel processing techniques. Fundamental radar theory and parallel processing techniques are covered, as well as a summary of state-of-the-art MIMO radar prototypes and parallel implementations of radar signal processing.

Chapter 3 describes the radar system on which the proposed solution will be implemented and will describe the simulation environment, the signal processing chains, and the parallel implementations of both the existing 3D FFT and the proposed high-resolution modes. Additional compensation for short range targets will also be described to accommodate beamforming techniques in the lab environment.

Chapter 4 describes the experimentation setup and will present the measurement results for the resolutions and the refresh rates. Both simulation and measurements will be presented.

Finally, Chapter 5 presents the findings and conclusions of this thesis and discusses future work opportunities regarding MIMO radar and its signal processing.

2 Literature Survey

The literature survey chapter is composed of 6 sections which cover the required background theory as well as the state-of-the-art in MIMO radar prototypes and parallel implementation of radar signal processing. Section 2.1 covers radar fundamentals and signal processing techniques. Section 2.2 describes the MIMO radar scanning mode. Section 2.3 introduces the concept of parallel processing, specifically regarding the use of multicore CPUs and GPU architecture. Section 2.4 discusses the state-of-the-art MIMO radar prototypes. Section 2.5 demonstrates recent research in parallel implementations of radar signal processing. Finally, Section 2.6 summarizes the survey on the state-of-the-art and concludes that MIMO radar should benefit from GPU implementation.

2.1 Radar Fundamentals and Signal Processing Techniques

Radio Detection and Ranging (radar) was developed and used during World War II [11]. Radars transmit electromagnetic (EM) waves and listen for echoes. By measuring the delay between the signal's transmission and the reception of the reflections, the range between the radar and the scatterers can be determined [12]. The following equation relates the elapsed time delay Δt to the range R of the scatters where c, the speed of light, is approximately equal to $3 \times 10^8\ \mathrm{m/s}$.

$$R = \frac{c\Delta t}{2} \tag{2.1}$$

Specific radar applications include: Surveillance, Tracking, Weapon Guidance, Remote Sensing, Weather Detection, and Imaging to name a few [11]. Radars have been implemented from High Frequency (HF) to millimeter waves where Table 2-1 enumerates the radar band allocations and nomenclatures according to the Institute of Electrical and Electronics Engineers (IEEE) and the International Telecommunication Union (ITU).

IEEE nomenclature		ITU nomenclature	
Radar letter designation	Frequency range	Frequency range	Band No.
HF	3 MHz to 30 MHz	3 MHz to 30 MHz	7
VHF	30 MHz to 300 MHz	30 MHz to 300 MHz	8
UHF	300 MHz to 1000 MHz	0.3 GHz to 3 GHz	9
L	1 GHz to 2 GHz		
S	2 GHz to 4 GHz		
		3 GHz to 30 GHz	10
C	4 GHz to 8 GHz		
X	8 GHz to 12 GHz		
K_u	12 GHz to 18 GHz		
K	18 GHz to 27 GHz	30 GHz to 300 GHz	11
$\mathrm{K_a}$	27 GHz to 40 GHz		
V	40 GHz to 75 GHz		
W	75 GHz to 110 GHz		
mm	110 GHz to 300 GHz	300 GHz to 3000 GHz	12

Table 2-1. Letter Band and ITU nomenclature, reproduced from [13]

The following sections aim to highlight some of the key processes which radars need to perform. Surveillance radars need to scan large volumes of space in order to find targets of interest amongst clutter [14]. Classical radars are generally fitted with high-gain antennas which are rotated in azimuth as they perform their scans. The antennas' beam pattern tends to be narrow in azimuth, for good angular resolution, but wide in elevation to maximize the sweep area and therefore its search volume [14]. The following equation is a version of the Radar Range Equation (RRE) and is thoroughly used in radar analysis [11], [12].

$$R_{max}^4 = \frac{P_T G_T G_R \lambda^2 \sigma \tau_p}{(4\pi)^3 (SNR) k T_{sys} L} \tag{2.2}$$

where:

R = range (m)
P_T = transmit peak power (W)
G_T = transmit antenna gain (unitless)
G_R = receive antenna gain (unitless)
λ = wavelength (m)
σ = radar cross section (m^2)
τ_p = pulse width (s)
SNR = signal to noise ratio (unitless)
k = Boltzmann's constant $\approx 1.38 \times 10^{-23}$ W/(K·Hz)
T_{sys} = system equivalent temperature (K)
L = system losses (unitless, with $L > 1$)

Equation (2.2) is used to predict the maximum range at which we can expect to detect a target of RCS σ, with a given probability for a given signal to noise ratio (SNR). Note that larger ranges are expected when the $P_T\tau_p$ (peak power and pulse width) product is large. Therefore, long range radars operate at long pulse widths and low PRFs to allow for unambiguous detection [11]. During a radar's design phase, all these parameters would need to be optimized for the task(s) the radar was meant for. Not only does a radar need to detect targets at a given range, but consideration must also be given to Electronic Protection (EP) and Intercept Probability (for military applications). Spectrum allocations and mitigation of competing signals which occupy the same RF band must also be a part of the design [14], [15].

2.1.1 Frequency Modulated Continuous Wave (FMCW) Radar

The FMCW radar enables a Continuous Wave (CW) or pulsed radar (using modulation on pulse) to make range measurements by adding bandwidth to the signal [16]. This is done by modulating the transmitter frequency as a function of time. As seen in Figure 2-1, any reflected signal will be delayed in time and will have a different frequency, when compared to the current frequency of the local oscillator. This difference in frequency, also called the bearing frequency, can be measured to estimate the range of the target [16]. Since the radar FMCW radar is transmitting and receiving simultaneously, it does not suffer from blind ranges in the same manner as the traditional pulsed radar [15], which is useful for detection at short ranges such as in automotive radar.

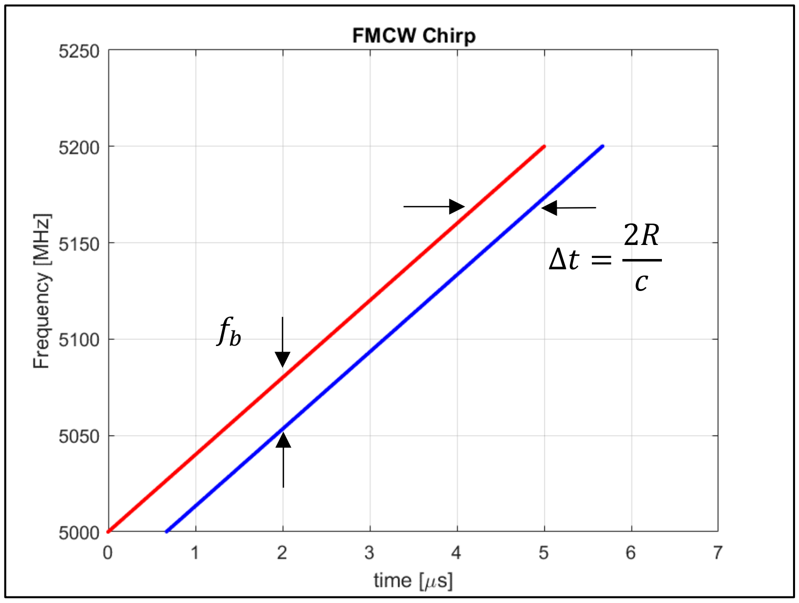

Figure 2-1. FMCW Chirp, adapted from [16]

In Figure 2-4, the target is stationary and does not produce a Doppler shift. In general, the beat frequency can be expressed as the combination of a range component and a Doppler component. This is shown in (2.3).

$$f_b = \underbrace{\frac{2k_0R}{c}}_{\text{Range Component}} - \underbrace{\frac{2f_t\frac{\partial R}{\partial t}}{c}}_{\text{Doppler Shift}} \tag{2.3}$$

where:

f_b = beat frequency (Hz)
c = speed of light (m/s)
k_0 = chirp rate (Hz/s)
R = range between the radar and the target (m)
$\frac{\partial R}{\partial t}$ = range rate (m/s)
f_t = frequency of operation (Hz)

If the radar is designed to operate in an environment where the Doppler shifts are negligible when compared to the range components, (2.4) can be used as a range estimator.

$$R \approx \frac{cf_b}{2k_0} \tag{2.4}$$

In general, FMCW radars have separate transmit and receive apertures to guarantee adequate isolation between the transmitter and the receiver. The receiver also needs a copy of the transmitted waveform to perform mixing. The mixing process generates the signal required for range and doppler measurements [16]. Figure 2-2 shows a generic FMCW radar block diagram where the frequency counter could be any process (analog or digital) which extracts the beat frequency, and the indicator represents the user interface or graphical output.

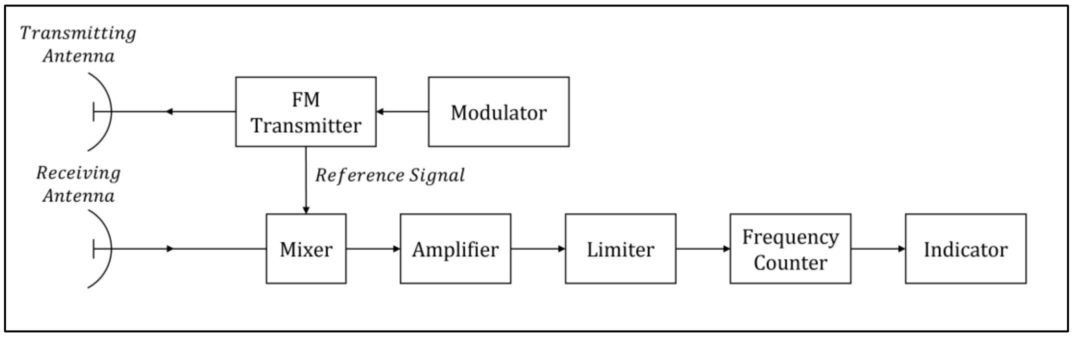

Figure 2-2. FMCW Block Diagram, reproduced from [16]

To extract target velocity information when Doppler is small compared to the range component, the dwell time must be increased so that the slowest radial velocities can be measured [16], which may be accomplished by processing several FMCW pulses in slow-time to form a Coherent Processing Interval (CPI) [3], where ramps are transmitted every $1/PRF$ (seconds). Between every ramp interval, the radial range of the target changes which translates as a measurable change of phase. The number of ramps in the CPI N_p must be selected so that the total integration time allows for the required velocity resolution. Note that the PRF is usually chosen to balance unambiguous range and blind speed requirements [12]. Therefore, for a given PRF and desired Doppler resolution, (2.6) is used to determine the number of ramps required in the CPI.

$$t_{int} = \frac{1}{f_{d\_min}} \tag{2.5}$$

$$N_P = \frac{PRF}{f_{d\_min}} \tag{2.6}$$

where:

t_{int} = required dwell time
f_{d_min} = smallest doppler shift the system is required to detect
PRF = pulse repetition frequency
N_P = number of pulses required in the CPI

2.1.2 Doppler Frequency Analysis

Modern radars use Doppler filtering and frequency transforms in order to suppress clutter and to characterize the targets with respect to range and radial velocity [12], which can then be used to triage targets based on their velocities. Figure 2-3 shows targets, clutter, and noise within the Range-Doppler space. The Fast Fourier Transform (FFT) is commonly used for this task [12], which converts time-based samples into their frequency spectrum. Since velocity information is coded in the reflection's Doppler frequency, the FFT enables the radar system to sort the echoes by velocity.

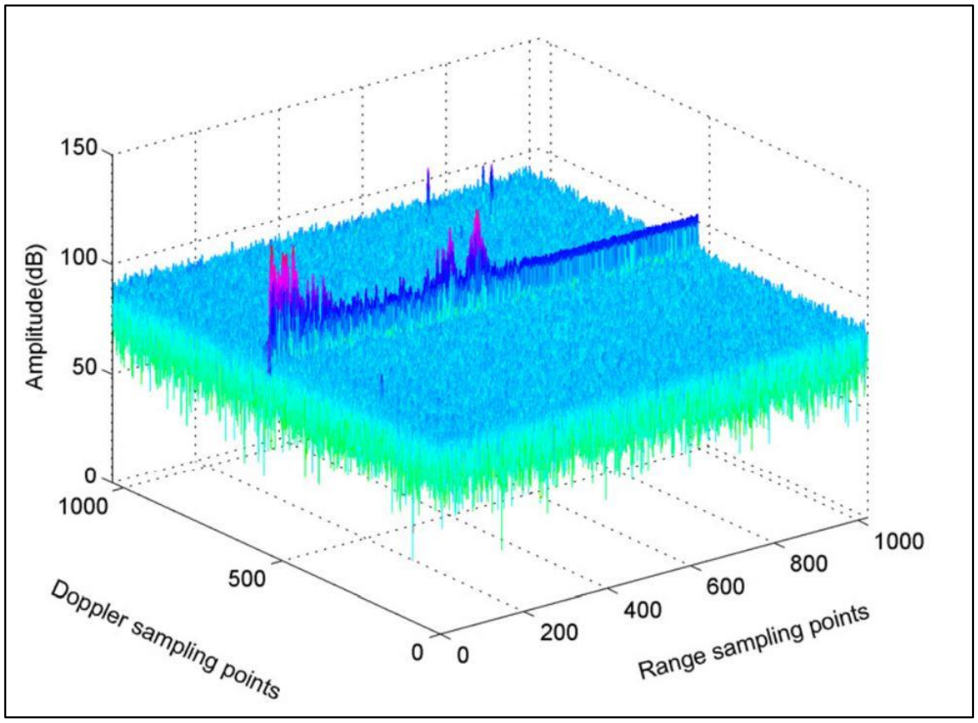

Figure 2-3. Range-Doppler Space, reproduced from [17]

The Discrete Fourier Transform (DFT) is an algorithm which transforms time domain signals into their frequency domain spectrum [18]. The frequency coverage and resolution of the DFT depends on the sampling rate of the system and the size of the DFT (number of samples to evaluate). As shown in (2.7), the DFT evaluates the sum of the input samples multiplied by complex weights, which must be performed for all frequency bins k [18].

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi nk}{N}} \qquad k = 0, 1, \cdots, N-1 \tag{2.7}$$

$$f[k] = \frac{kf_s}{N} \text{ (Hz)} \tag{2.8}$$

where:

X = complex DFT coefficients
x = complex input signal
k = frequency bin index
n = time index
f_s = sampling frequency
N = size of DFT

As an example, the following continuous signal is sampled at 10 kHz (512 samples are collected).

$$s(t) = \cos(2\pi f_0 t)\,(1 + 0.5\cos(2\pi f_1 t)) \qquad (2.9)$$

where:

$f_0 = 2000$ Hz
$f_1 = 105$ Hz

Equation (2.9) is an Amplitude Modulated (AM) signal which has a carrier frequency of 2000 Hz and is modulated by a 105 Hz sinusoid. Using the DFT, both the time domain samples and the frequency spectrum are shown in Figure 2-4. In the lower part of the figure, the carrier frequency can clearly be seen, along with the upper and lower side bands of the 105 Hz modulation. The DFT is a useful Digital Signal Processing (DSP) tool which is used across many domain [18], including the range and velocity estimations of targets in a FMCW radar system.

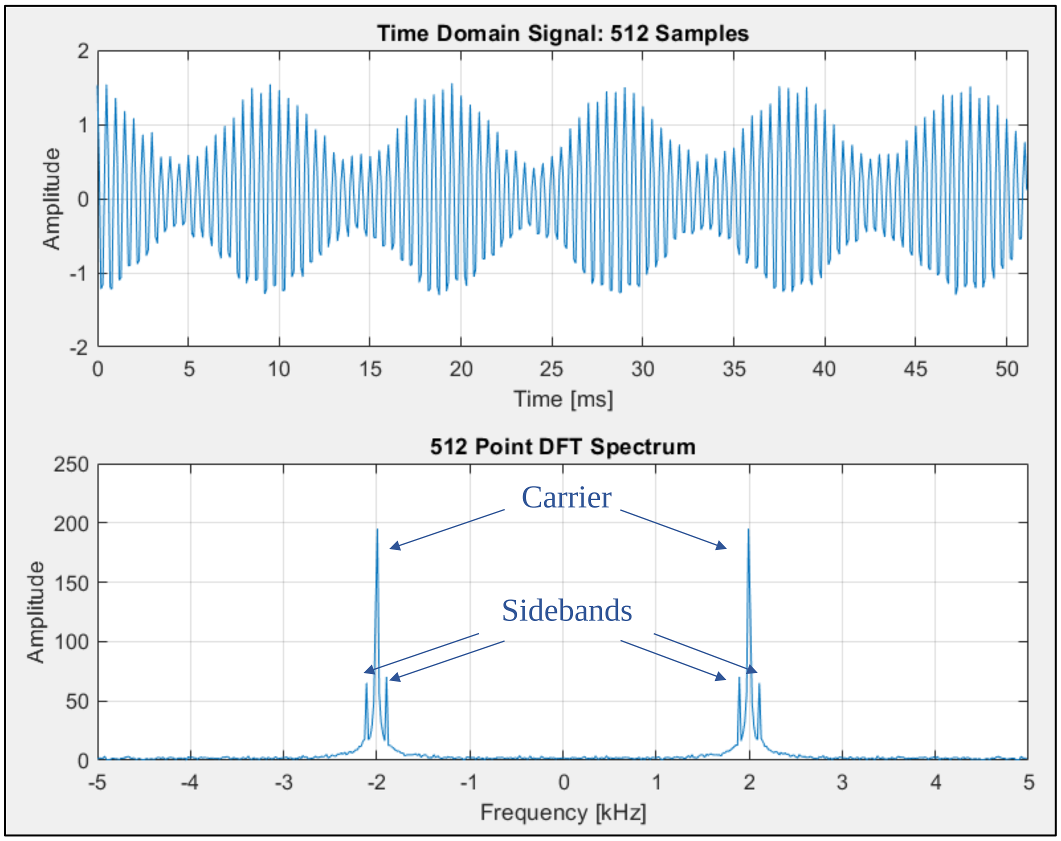

Figure 2-4. DFT Example, AM signal

The DFT as presented here is rarely used in this form, as it has a time complexity on the order of N^2. Efficient algorithms, such as the Decimation-in-Time (DIT) radix-2 method, have been developed to compute the DFT by splitting the computation in steps. These Fast Fourier Transforms (FFT) have a time complexity on the order of $N\log_2(N)$, which dramatically reduces the computation time of the Fourier Transform [18].

Sometimes, however, the FFT resolution is too coarse and cannot provide the fine spectral details required. If spectral analysis of a narrow band is required, a different algorithm will be needed. The Chirp-Z Transform (CZT) is a DFT-like algorithm which enables a user to select the frequency limits and resolution of the computations [19] and is expressed in (2.10). Figure 2-5 utilizes the Z-Plane to illustrate the frequency point distribution of the DFT and the CZT. Whereas the DFT spreads its points evenly across the z-plane, from $-\frac{\omega_s}{2}$ to $\frac{\omega_s}{2}$, the CZT can focus its frequency points between the frequencies of interest ω_1 and ω_2 with a frequency step of $\Delta\omega$.

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi(f_1+k\Delta f)n} \qquad k = 0, 1, \cdots, N-1 \qquad (2.10)$$

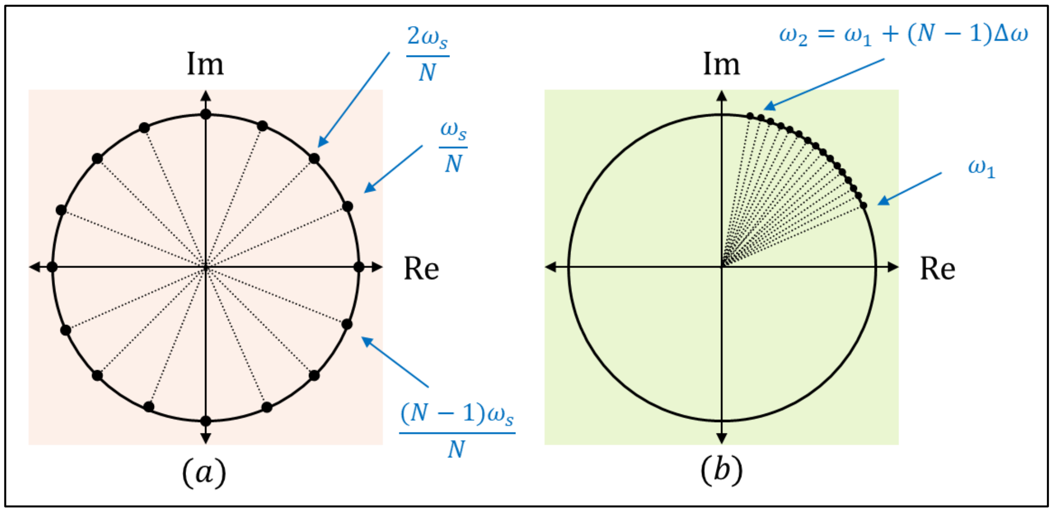

Figure 2-5. Frequency Points Comparison: (a) DFT (b) CZT

To show how the CZT can zoom-into a narrow frequency band and provide finer details than the DFT, we reutilise the example AM signal from (2.9). The signal is sampled at 10 kHz and 512 samples are collected, just as before. This time, however, the data will be sent to a 512-point DFT and a 512-point CZT where both results will be juxtaposed. The frequencies of interest for the CZT will be between 1800 Hz and 2200 Hz, which means each frequency step has a size of 0.7812 Hz. Figure 2-6 shows amplitude results for both algorithms.

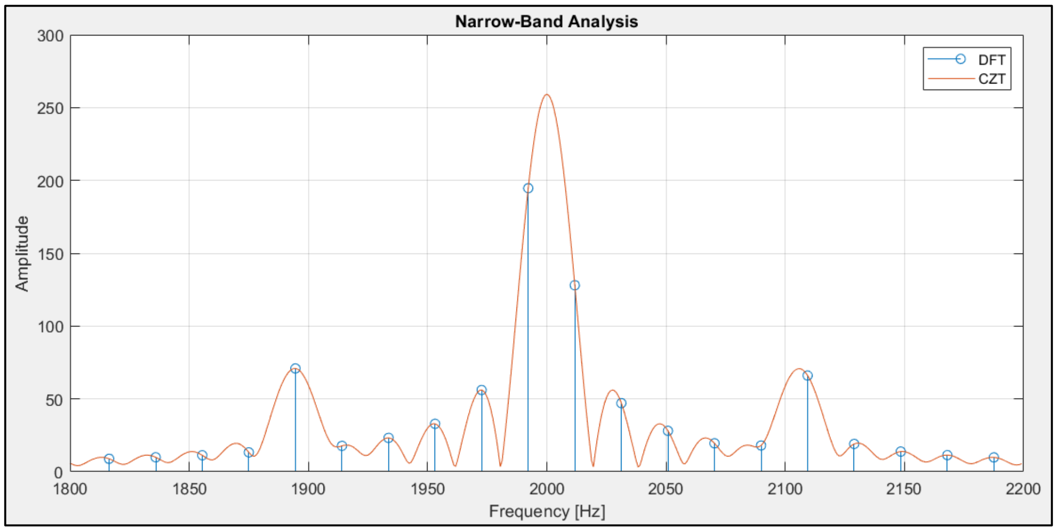

Figure 2-6. DFT and CZT Narrow-Band Comparison

Fortunately, there is also a fast algorithm which computes the CZT, as shown in (2.11) [19]:

$$X[k] = W^{\frac{k^2}{2}} \sum_{n=0}^{N-1} x[n] e^{-j\omega_1 n} W^{\frac{n^2}{2}} W^{-\frac{(k-n)^2}{2}} \tag{2.11}$$

where:

$W = e^{-j\Delta\omega}$

$\Delta\omega = \dfrac{\omega_2 - \omega_1}{N}$

ω_2 = stop frequency (rad/s)

ω_1 = start frequency (rad/s)

Although this format seems more tedious than the direct computation of the CZT, equation (2.11) is written as a convolution. Since convolutions can be done via multiplications in the frequency domain, the computational power of the FFT can be used to achieve computational complexity on the order of $N \log_2(N)$ [19]. The computation of the CZT requires two FFTs, one IFFT, and several steps of multiplications. Figure 2-7 is a visual representation of the fast CZT algorithm, in the form of a block diagram.

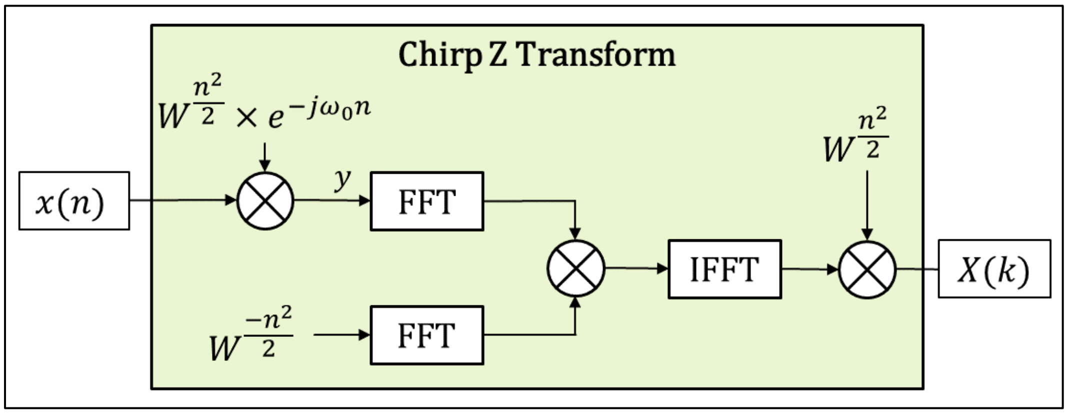

Figure 2-7. Fast CZT Algorithm: Block Diagram

2.1.3 Phased Array Radars

Phased array radars have been used in many applications due to their scalability, performance, and flexibility [11]. Using many individual antenna elements, RF energy is combined in space which creates desirable beam patterns [12]. The one-way normalized array factor for an $M \times N$ uniform linear planar array, as shown in Figure 2-8, is expressed by (2.12) [20].

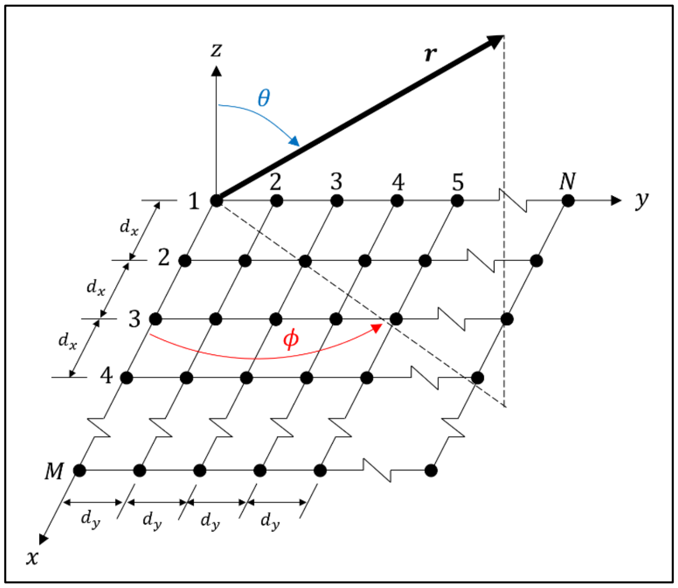

Figure 2-8. Example of an M by N Planar Array, reproduced from [20]

$$AF_n(\theta,\phi) = \left\{\frac{1}{M}\frac{\sin\left(\frac{M}{2}\Psi_x\right)}{\sin\left(\frac{\Psi_x}{2}\right)}\right\}\left\{\frac{1}{N}\frac{\sin\left(\frac{N}{2}\Psi_y\right)}{\sin\left(\frac{\Psi_y}{2}\right)}\right\} \quad (2.12)$$

where:

$\Psi_x = kd_x \sin(\theta)\cos(\phi) + \beta_x$
$\Psi_y = kd_y \sin(\theta)\sin(\phi) + \beta_y$
$k = \frac{2\pi}{\lambda}$
λ = wavelength (m)
d_x = spacing between antennas along x-axis (m)
d_y = spacing between antennas along y-axis (m)
β_x = phase progression of antennas along x-axis (rad)
β_y = phase progression of antennas along y-axis (rad)
θ, ϕ = spherical angles (rad)

In a phased array radar, the physical elements are generally fixed. Therefore, the system can use phase shifters to control the transmit phase of each radiating element [11]. The phase progression angle is represented by β_x and β_y in (2.12) and is usually given in radians [20], which steers the main beam to a desired location in θ and in ϕ [12]. Figure 2-9 shows the radiation pattern of a steered array for a given set of parameters.

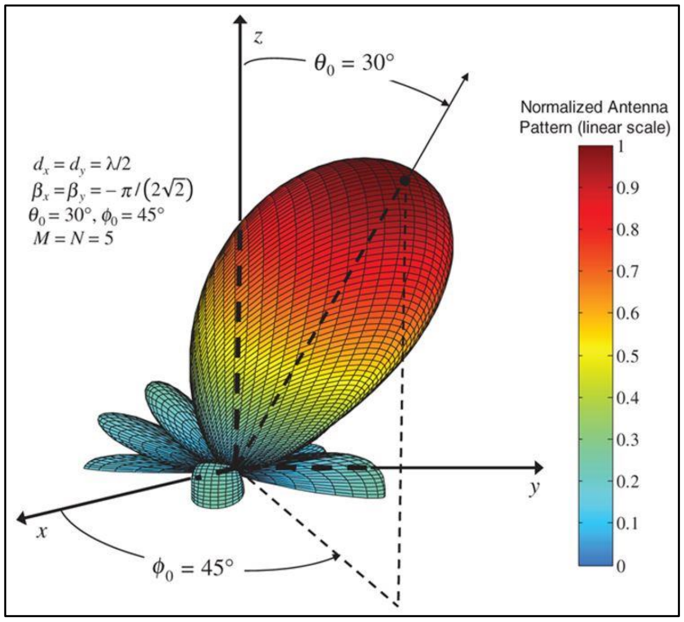

Figure 2-9. One-Way Radiation Pattern of Steered Array, reproduced from [20]

A phased array radar can therefore perform electronic scanning of a particular Field of View (FOV) by simply changing the phase progression of its array elements [11]. Some radars will use a hybrid technique where they transmit a broader beam to cover larger angular areas and then perform Digital Beamforming on receive [21], which will reduce the amount of time needed to scan a large FOV but will require significantly more processing [12], [21].

2.1.4 DOA Beamforming

Radar systems generally need to extract the azimuth information of target echoes. In DOA estimation, intercepted signals across many antennas are used in algorithms to determine the azimuth or elevation of the target [12]. As a plane wave approaches an array, it is intercepted by the array elements at different times as a function of the DOA and the positions of the elements. Figure 2-10 shows an array which receives an incident signal from an angle θ. The incident angle will generate a phase difference across each antenna element [22]. For a Uniform Linear Array, the relative phase shift φ between adjacent antennas can be expressed by (2.13).

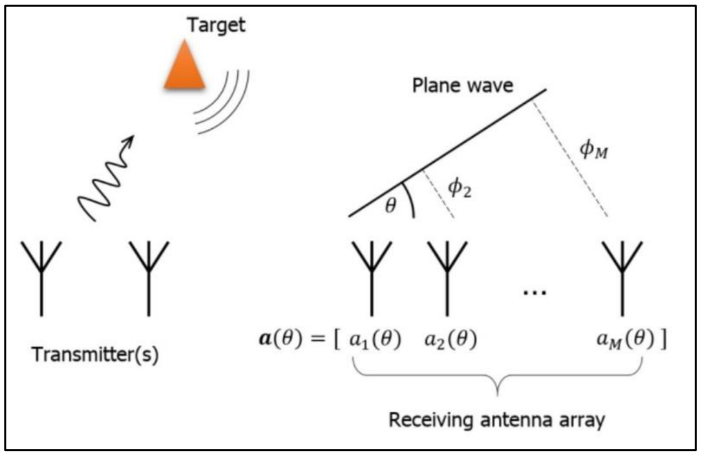
Target
Plane wave
ϕ_M
ϕ_2
θ
Transmitter(s)
$\boldsymbol{a}(\theta) = [\ a_1(\theta)\quad a_2(\theta)\quad \cdots \quad a_M(\theta)\ ]$
Receiving antenna array

Figure 2-10. Plane Wave Incident on Array, reproduced from [22]

$$\varphi = \beta d \cos(\theta) \tag{2.13}$$

where:

$\beta = 2\pi\lambda$
$d =$ distance between antenna elements (m)
$\theta =$ incident angle of wavefront (rad)

The advantage of performing digital beamforming upon reception is that the radar can form multiple beams simultaneously instead of pointing its antenna in a given direction [11]. Some modern radars, such as the Thales SMART-S MkII, scan their FOV mechanically in azimuth and using digital beamforming in elevation, allowing the radar system to generate 3D target tracks [23]. Other systems, such as automotive radar, rely on digital beamforming to find the azimuth of nearby vehicles [22].

A popular and efficient DOA estimator will simply make use of the FFT because the progressive phase difference between antenna elements is equivalent to a spatial frequency where the sampling rate is related to the element position. The resulting normalized frequency ω_n will therefore be a value between -1 and $+1$ where 0 represents a target at boresight [6]. Since (2.13) represents the phase shift across element:

$$\theta = \operatorname{acos}(\omega_n) \tag{2.14}$$

The conventional, or classical method of solving for the DOA is the Bartlett Beamformer, which uses steering vectors to generate the power spectrum across the angular FOV [24]. The steering vector is simply the expected phase shift across all elements for any given incident angle. The signal power at a given angle is given by (2.15) [25] and is simply a multiplication between the covariance matrix of the input signals and the steering vector.

$$P(\theta) = \boldsymbol{a}(\theta)^H \boldsymbol{R}_x \boldsymbol{a}(\theta) \tag{2.15}$$

where:

$\theta =$ steering angle
$P(\theta) =$ Bartlett power spectrum, for given angle
$a(\theta) =$ steering vector, for given angle
$R_x =$ signal covariance matrix
$[\quad]^H =$ complex transpose

Just as before, a target in the far field is assumed and the steering vector is equal to:

$$a(\theta) = \left[1, e^{-j\beta d cos(\theta)}, e^{-j2\beta d cos(\theta)}, \cdots, e^{-j(N-1)\beta d cos(\theta)}\right] \tag{2.16}$$

To scan the entire FOV, the Bartlett Beamformer must evaluate (2.15) for all angles θ of interest. Unlike the FFT, the number of evaluated angles is not tied to the number of antennas (i.e., a 64-point FFT will generate a spectrum with 64 equally spaced frequencies). The Bartlett Beamformer is not a High-Resolution algorithm, when compared to the Capon Beamformer or any of the subspace-based method [6]. However, it is robust, and works without a priori knowledge of the targets. Additionally, other High-Resolution algorithms do not function well in a single snapshot (SSS) setting or when multiple correlated signals are present. Because of this, the Bartlett Beamformer is often the method of choice for radar digital beamforming [24], [25]. Figure 2-11 shows a comparison of the FFT and the Bartlett DOA for an array of 64 antennas. The targets are located at $-55°$ and $-40°$ from boresight. The Bartlett DOA in this example was evaluated from $-90°$ to $90°$ with an angular step of $0.25°$, for a total of 720 angle bins.

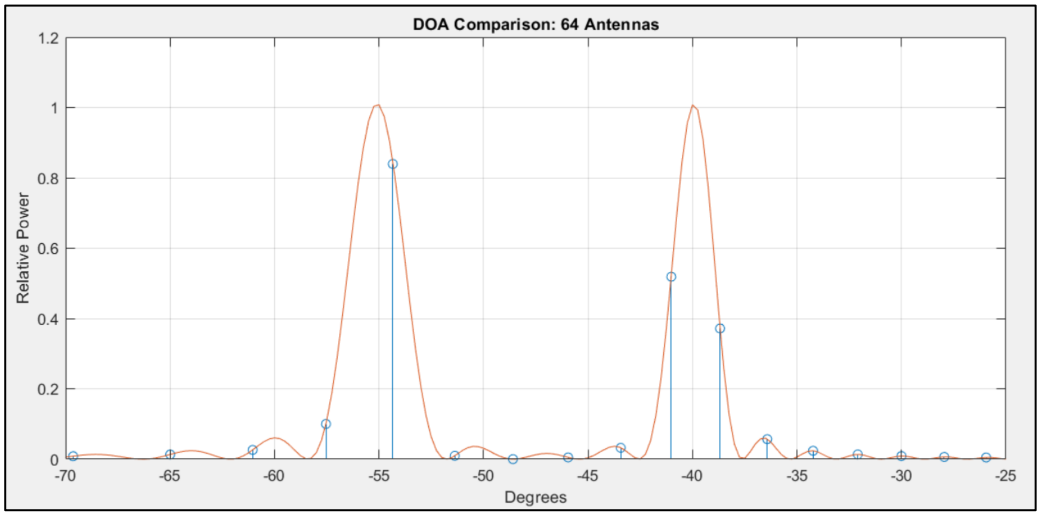

Figure 2-11. DOA Comparison between FFT and Bartlett

2.2 MIMO Radars

Like the phased array radar, the MIMO radar utilizes many antenna elements to form arrays. However, there are two main differences between them. Firstly, the phased array antennas generally perform both transmit and receive functions while the MIMO radar has dedicated transmit and receive arrays which are spatially separated [3]. Secondly, all the radiating elements of a phased array radar transmit the exact same waveform. Phase shifts and amplitude weighting are used to shape and steer the beams as required. The MIMO radar's transmitting antennas, on the other hand, all transmit different or orthogonal waveforms [3].

Because of these different transmitter waveforms, the MIMO radar receivers can identify, sort, and process each waveform individually. This means that the transmit waveforms must be orthogonal to each other [3]. When the MIMO array elements are spaced just right, the demodulated signal at the receivers create an array which is larger than the total number of antennas [6]. The convolution of transmit and receive one-way patterns to generate the two-way pattern of a radar system isn't unique to MIMO. However, the orthogonal waveforms enable the receive to discriminate between individual transmit/receive paths and combine the multiple signals in many different ways [3], whereas the phase array configuration does not. The signal processor uses the demultiplexed signals of all transmit/receive permutations for beamforming [3]. Figure 2-12 shows a 2-element transmit array and a 4-element receive array where the phase progression measured at the receivers for each transmit antenna is equivalent to that of an 8-element ULA. Note that in the figure, $\varphi = \beta d \sin(\theta)$ because θ being measured from boresight.

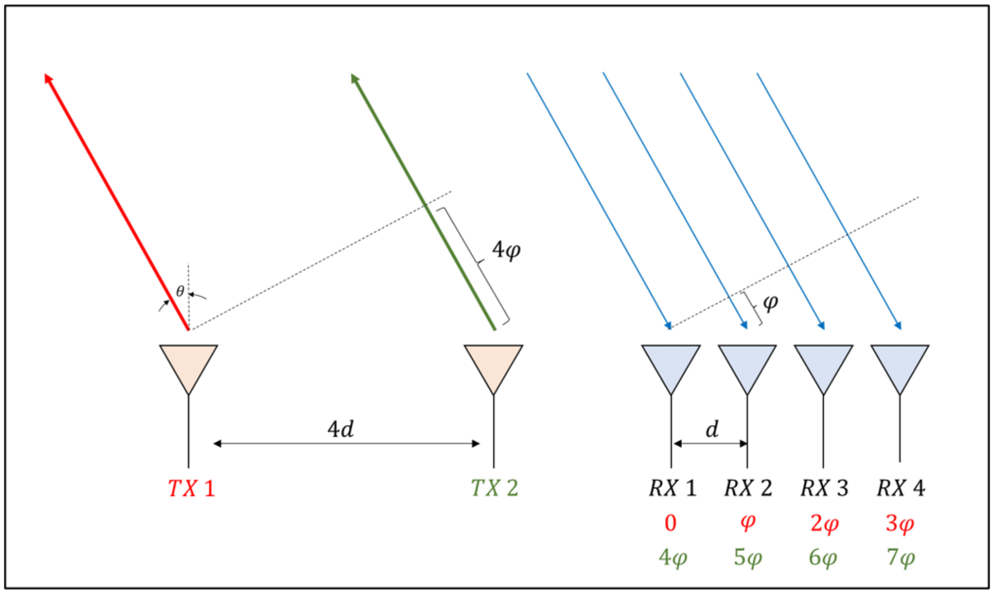

Figure 2-12. MIMO Radar Virtual Array, adapted from [6]

MIMO radar waveforms can be orthogonal in time and in frequency. Therefore, the use of Time Domain Multiple Access (TDMA), Frequency Domain Multiple Access (FDMA), and Code Division Multiple Access (CDMA) have each been investigated for the multiplexing scheme of the waveforms [3].

In TDMA, each transmitter is assigned a time slot to transmit its waveform. The receiver then sorts the echoes from each transmitter, sequentially [3]. Although TDMA is one of the simplest methods of implementing MIMO into a radar, it is not the most effective, as the effective PRF is reduced by the number of transmitter waveforms, reducing the data output rate and slowing the radar's possible maximum scan rate [3].

In FDMA, each transmitter is assigned a frequency channel which is orthogonal to the others. However, FDMA has poor spectrum efficiency as each channel must be as large as the radar's waveform bandwidth [3]. If there are M transmit antennas, the FDMA MIMO radar will occupy M times the bandwidth of its TDMA counterpart, thereby making hardware, ADC, and antenna selection and design difficult [3].

In CDMA, each transmitter is given a short code which is modulated onto the pulse. The transmitter can now transmit simultaneously on the same frequency channel [3]. The receiver can differentiate between each transmitter by demodulating the signals with the appropriate matched filter. It is difficult, however, to find codes which are truly orthogonal, which is especially true when pulse durations are short [3] and "even small residual cross-correlation noise can degrade the benefits of MIMO…" [3].

It is evident that each of these MIMO methods have its benefits and challenges, and selecting a method will depend on the goals and restraints of the MIMO radar being designed [12]. Although there have been recent developments in novel waveform design [26], more work needs to be accomplished in finding ways to ensure the orthogonality of the waveforms [3], [6]. Once all the waveforms have been sorted into a virtual array (meaning individual paths from each transmitter to each receiver), the channels can be added as sheets of Range Doppler, as shown in Figure 2-13. The figure also shows how each channel can then be processed by an FFT, operated in the depth (channel) direction, to transform the information into DOA information.

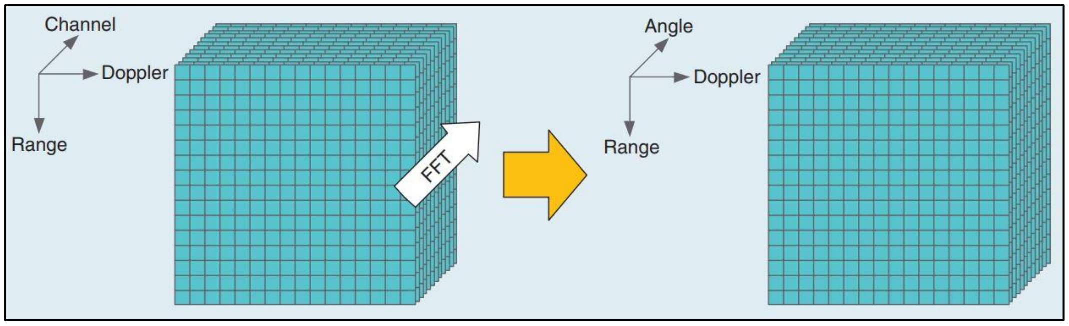

Figure 2-13. MIMO DOA Beamforming across the Data Cube, reproduced from [6]

Although Figure 2-13 shows the FFT, other methods for angle estimation exists including the CZT and the Bartlett methods that were previously described. So-called subspace DOA techniques provide high resolution and accuracy but have a much higher computational demand. Multiple Signal Classification (MUSIC), Estimation of Signal Parameter via Rotational Invariance Technique (ESPRIT), Sparse-Sensing, Iterative Adaptive Approach (IAA), and Minimum Variance Distortionless Response (MVDR) are some of the common high-resolution angle-finding techniques [6]. As discussed earlier, however, not all methods work at low

SNR, on single snapshots, for multiple correlated signals, or without a priori knowledge of the targets. Figure 2-17 shows a comparison between the FFT DOA method and the high-resolution IAA method. The sharper peaks and lower Side Lobe Levels (SLL) are clearly a desirable feature of the high-resolution algorithm [6]. Figure 2-14 is the result of a simulated Sparse Linear Array (SLA) MIMO radar.

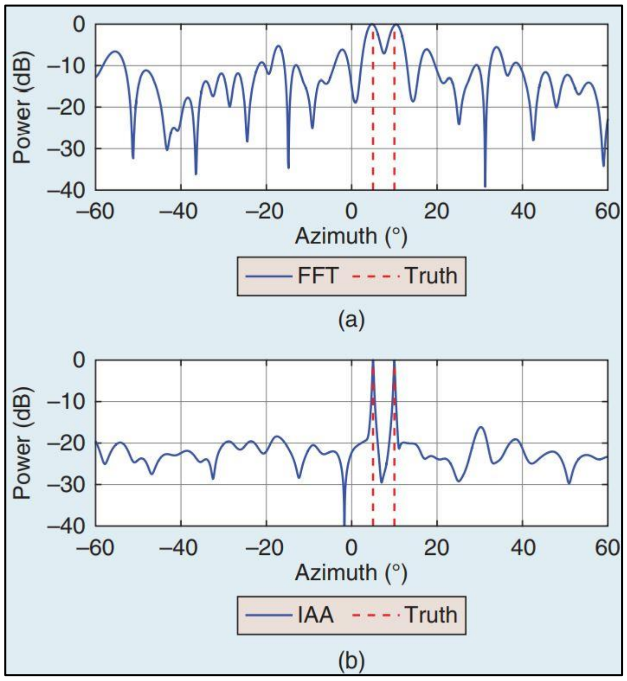

Figure 2-14. DOA Estimation: (a) FFT (b) IAA, reproduced from [6]

Even before the beamforming stage, the computation requirement of a MIMO radar is increased due to the range and doppler processing done for all demultiplexed channels. Additionally, the beamforming algorithm must be performed across all range-doppler cells. These tasks are independent and provides opportunities for parallelization.

2.3 Parallel Processing Techniques

In general, computer programs are written in sequential steps in order to be executed on Central Processing Units (CPU) [27]. As new hardware got developed, performance of CPUs increased and so did the speed of the programs. Up until recently, the performance of CPUs seemed to be correlated with Moore’s law (which predicts that transistor density doubles approximately evert 1.5 years) [28]. Since 2010, this relationship changed and the increase in single-threaded CPU performance is slowing down for each new generation of chip [28]. This is shown in Figure 2-15. Several factors such as leakage currents, heat dissipation, and energy consumption limits the increase in the clock cycles of a processing unit and therefore, CPU performance [27], [28].

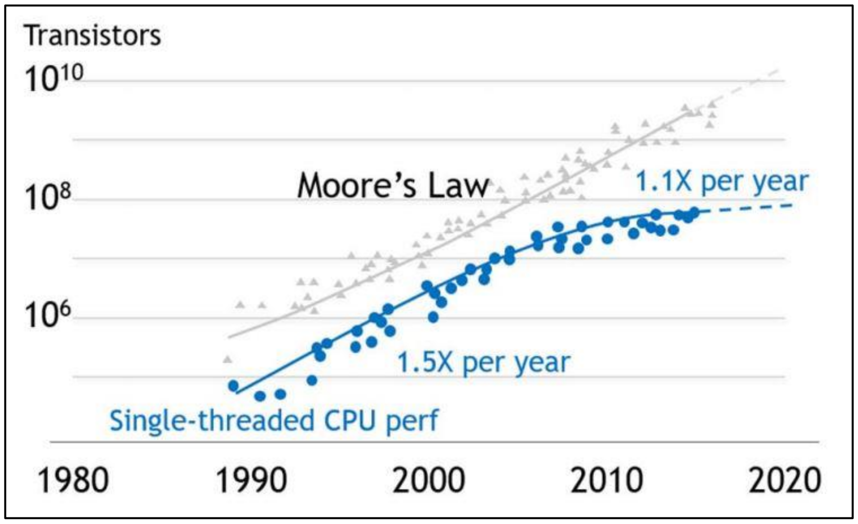

Figure 2-15. Moore's Law and Single-Threaded CPU Performance, reproduced from [28]

Instead of making the computers faster by increasing the CPU clock rates, they can make them *wider* by splitting the computing tasks onto many parallel threads preformed on multiple cores [27]. On computers, this is usually done by leveraging the power of multicore CPUs using an API like OpenMP or by using co-processors such as General Purpose Graphics Processing Units (GPGPU) [27].

2.3.1 Multicore CPU

Modern computers and laptops have multicore CPUs which make parallel programming opportunities more available than ever [27]. Using an Application Programming Interface (API) such as OpenMP, programmers can direct the CPU to perform certain tasks in parallel. Some tasks can only be done sequentially, while others are repetitive or must be executed on large amounts of data. This means that the computer program has defined sequential and parallel regions, which is shown in Figure 2-16 [29].

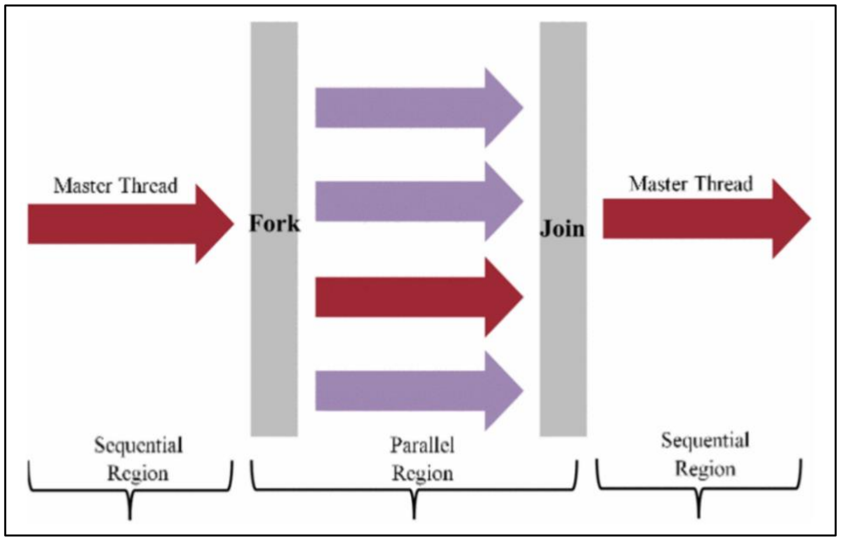

Figure 2-16. Sequential and Parallel Regions of a Program, reproduced from [29]

The acceleration which can be achieved by using multicore CPU depends on the number of cores and the portion of the code which can only be done sequentially. This is called Amdahl's law [28]. In order to maximize the speed up, given a set number of cores, the programmer must strive to maximize the ratio of parallel processing within the program. Equation (2.17) is the formulation of Amdahl's law, reproduced from [28].

$$a = \frac{1}{\left(R + \frac{1 - R}{N}\right)} \tag{2.17}$$

Where:
$a =$ multicore efficiency, or speedup ratio
$N =$ number of CPU cores
$R =$ ratio of sequential processing
$1 - R =$ ratio of parallel processing

Figure 2-17 shows the efficiency (speed up) of CPUs for different values of N and R.

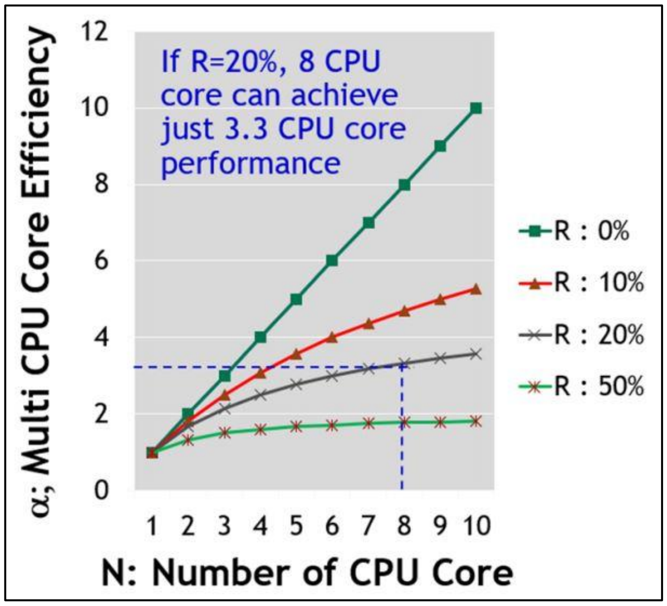

Figure 2-17. CPU Efficiency (Speed Up) as per Amdahl's Law, reproduced from [28]

2.3.2 Graphics Processing Units (GPU)

GPUs were optimized for gaming applications where developers needed to increase the throughput of operations to generate high quality graphics [27]. Since the computation required to generate image frames are highly independent, a large amount of them could be performed simultaneously in parallel [27]. The GPU has also been used in other applications where there is a lot of data to process and where the computation tasks can be parallelized, which is common in the scientific community [30].

The GPU is built to run several hundred to several thousand threads simultaneously [31]. The GPU has several building blocks containing Streaming Multiprocessors (SM) [27], [31]. These SMs all have several Streaming Processors (SP) which share control logic and instruction cache [27]. Executed threads are mapped to SMs and have access to a Shared Memory. The GPU also has a Global Memory and a Constant Memory to hold and share data send by the Host computer [27]. Figure 2-18 shows a generic GPU architecture containing 4 SMs: each with 8 SPs, 2 Special Function Units (SFU), and Shared Memory. Although the Global Memory is large, read and write operations take a long time [27] which can increase the execution time of Kernels. The Shared Memory, in contrast, is smaller than the Global Memory but the access time is much shorter [27]. Programmers should take care to minimize Global Memory access to reduce the execution times of their program.

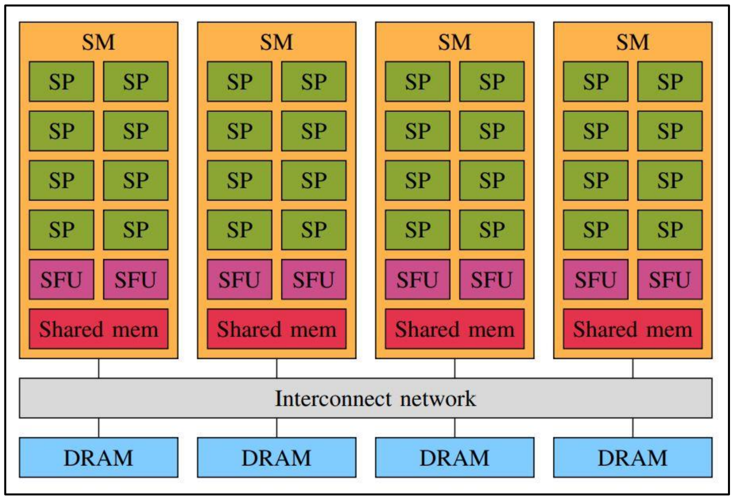

Figure 2-18. GPU Architecture, reproduced from [32]

Compute Unified Device Architecture (CUDA) is an extension to other programming languages which provide additional functionality for GPU programming [27]. With CUDA, a programmer can code sequential instructions which are to be executed on the CPU and Kernels, which are functions that run in parallel on the GPU when called from the CPU [31]. When launching a Kernel from the CPU, the GPU will execute the parallel threads in Blocks [27]. A Grid contains all the Blocks, while a Block will contain many Threads. A Block of threads is assigned to a SM, which schedule and perform the parallel computations of up to 1024 threads per Block [27]. Each individual Thread will have an ID number which will locate it within the Grid [27]. Figure 2-19 shows the CUDA Thread hierarchy.

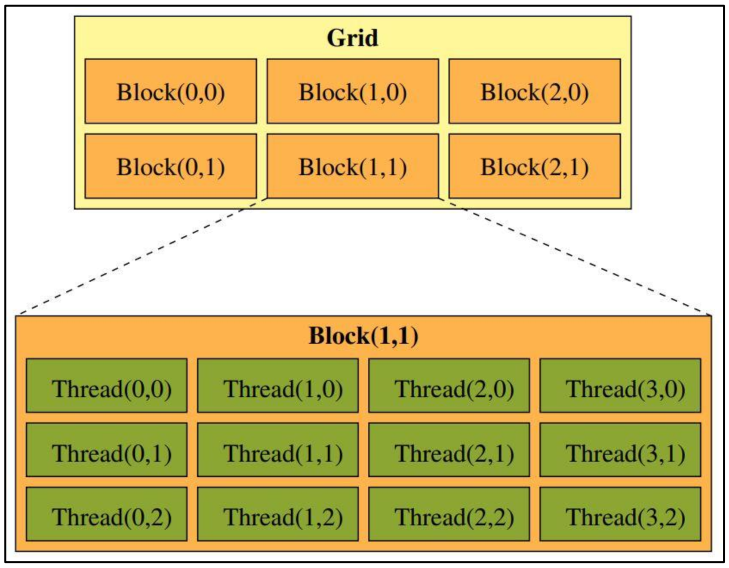

Figure 2-19. CUDA Thread Hierarchy, reproduced from [31]

2.3.2.1 Programming Considerations – Shared Memory

Access to the Global Memory is quite slow [27] and depending on the program, the use of Shared Memory could be beneficial. Shared Memory, which can be accessed much faster than the Global Memory, can be seen by all Threads of the same Block. The Threads themselves can be used to load individual elements of a data array from the Global Memory to the Shared Memory [27]. This limits the total number of Global loads and stores, which speeds up the execution time. However, the Shared Memory is much smaller than the Global Memory and caution must be taken not to over-allocate [27]. Additionally, the visibility of each memory type is limited and is summarized in Table 2-2.

Memory Type	Scope	Lifetime
Local Memory	Thread	Kernel
Shared Memory	Block	Kernel
Global Memory	Grid	Application
Constant Memory	Grid	Application

Table 2-2. GPU Memory Types and Scope, reproduced from [27]

2.3.2.2 Programming Considerations – Memory Access Patterns

Once relevant data is loaded from the host to the device, it is stored in the Global Memory (constant values may be stored in the Constant Memory). When a large amount of data needs to be loaded, it is best to do so in a coalesced manner [27], especially when done from the Global Memory as it is slower. If data from the Global Memory needs to be accessed in a random or non-sequential manner, the program will benefit from first performing a coalesced load onto a faster and more local memory (i.e., the Thread’s Local Memory or a Bock’s Shared Memory). From there, non-sequential access to the required data will have less of a detrimental effect on the Kernel’s execution time [27]. Note that as shown in Table 2-2, data that is stored in a Thread’s Local Memory can only be seen by that thread, just like the data stored in a SM’s Shared Memory can only be accessed by Threads within the same Block. Therefore, a coalesced store may be required to transfer the Kernel’s results back onto the Global Memory and, eventually, back to the Host [27]. Figure 2-20 shows the access model of CUDA device memory.

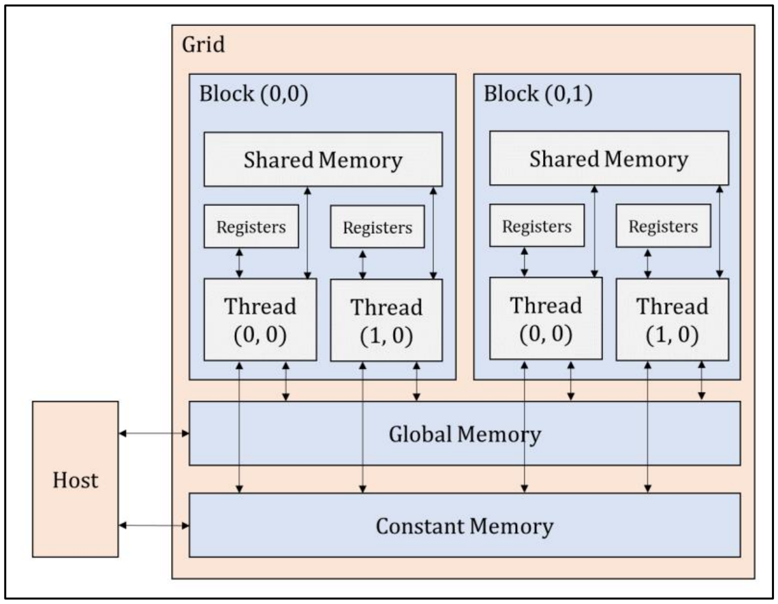

Figure 2-20. CUDA Device Memory Model, reproduced from [27]

2.3.2.3 Programming Considerations – Control Flow Divergence

The GPU will run sets of 32 Threads at a time, called Warps [27]. The SM "is designed to execute all threads in a warp following the Single Instruction, Multiple Data (SIMD) model" [27]. This means that all Threads within a Warp need to execute the same instructions, due to the shared control logic of the SM. As an example, if the code contains an **if/else** statement, the Warp will perform well so long as all the Threads have the same outcome (**if** or **else**). However, if some Threads have to perform the **ifs** and the others perform the **elses**, the SIMD hardware must execute them sequentially [27]. This problem is called Control Flow Divergence and will increase the execution time of a program [27].

2.3.2.4 Programming Considerations – Memory Transfer

Kernels, which perform computations in parallel, can be considerably faster than their sequential counterparts. However, the data first needs to get copied onto the GPU before the Kernel launches. Additionally, once the data ahs been processed, the results must be copied back to the Host [27]. The transfer of large data sets between the Host and the Device takes time and should be minimized. It is therefore wise to perform many tasks and computations on the GPU before copying the results back to the CPU [27].

Parallel programming techniques have been used in many fields to accelerate programs, signal processing, and run scientific simulations [27]. The acceleration provided by parallelizing signal processing tasks of the MIMO radar should reduce execution times and increase the system's refresh rate.

2.4 MIMO Radar Prototypes

This section will look at recent MIMO radar prototypes and describe their functionality, their performance, and their refresh rates.

2.4.1 Prototype A: TDMA MIMO Radar on FPGA and DSP

A 2D (range and azimuth) 4×4 FMCW MIMO Radar was demonstrated in [33] which leverages the parallel structure of the collected data. The system has 4 transmit antennas , 4 receive antennas, performs the fast-time Range FFT on FPGA, and performs the digital beamforming on a dedicated DSP module [33]. Using a TDMA multiplexing scheme, the radar creates 16 virtual MIMO channels. Each of these channels have 1024 time samples which are sent to an FFT range estimator, prior to being beamformed for imaging. To reduce computation and to generate an image of a reasonable size, the radar user can select the range of distances which will be sent for imaging. In the paper, 44 out of the 1024 range cells (representing ranges from 20 to 100 meters) are sent to the beamformer for imaging [33]. The raw data flow is shown in Figure 2-21.

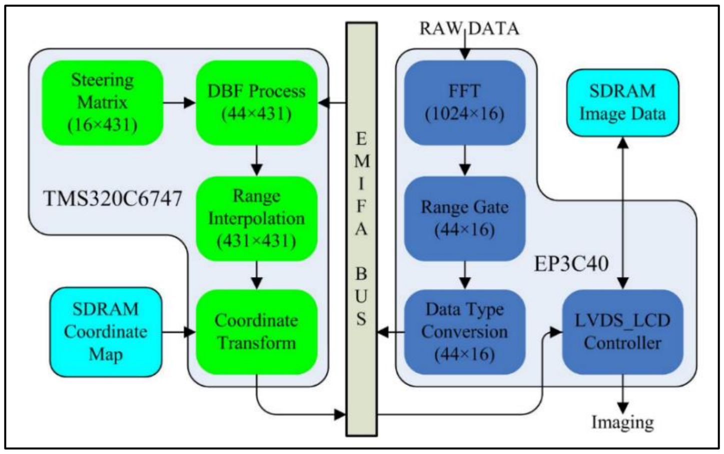

Figure 2-21. Image Processing using FPGA and DSP, reproduced from [33]

Within the DSP module, the Bartlett beamformer is executed. All the required steering vectors are pre-stored in the L2 memory, for quick access. Beamforming is done on a field of view spanning $-21.5°$ to $21.5°$ with an angular step of $0.1°$, meaning that there are 431 total angular bins to compute. It appears that only one chirp (per MIMO channel) is collected for processing, since no CPI was defined nor was there any indication of integration or Doppler processing. Overall, the FPGA/DSP configuration takes approximately $110\,\text{ms}$ to execute the signal processing. Each FMCW chirp has a duration of $20.6\,\text{ms}$ and is followed by a $19.4\,\text{ms}$ delay which enabled the channel switch to occur. The total data collection time T_s takes:

$$T_s = 16 \times (20.6 + 19.4) = 640 \text{ ms} \tag{2.18}$$

The signal processing could be executed at the same time as pulse collection, so the refresh rate of approximately $1.56\,\text{Hz}$ was purely a function of T_s. Figure 2-22 shows the functional block diagram of the radar. Figure 2-23 shows the antenna positioning of the transmit and receive arrays. Figure 2-24 shows an example of the radar image output.

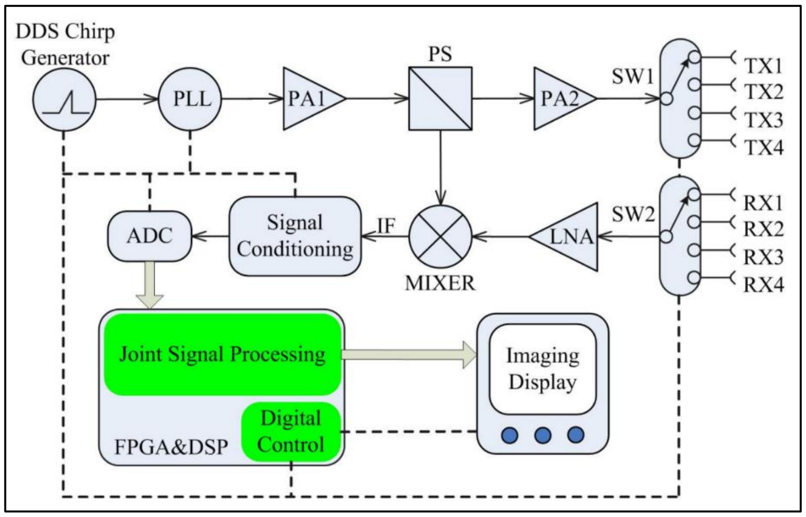

Figure 2-22. System Block Diagram, reproduced from [33]

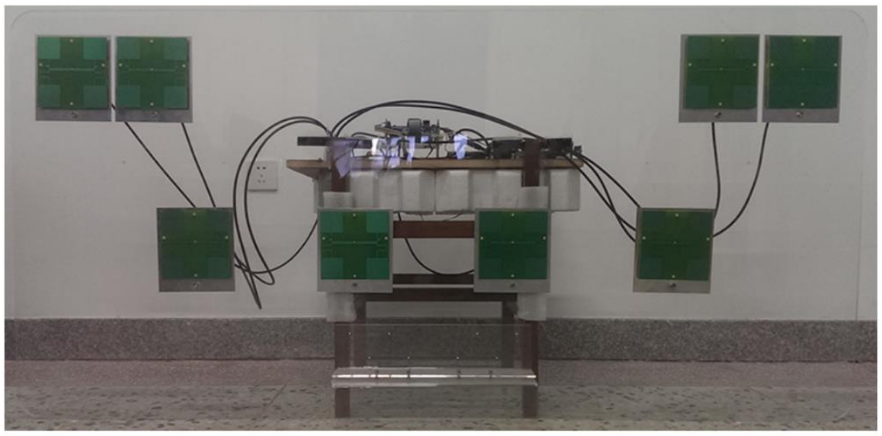

Figure 2-23. Antenna Placement, reproduced from [33]

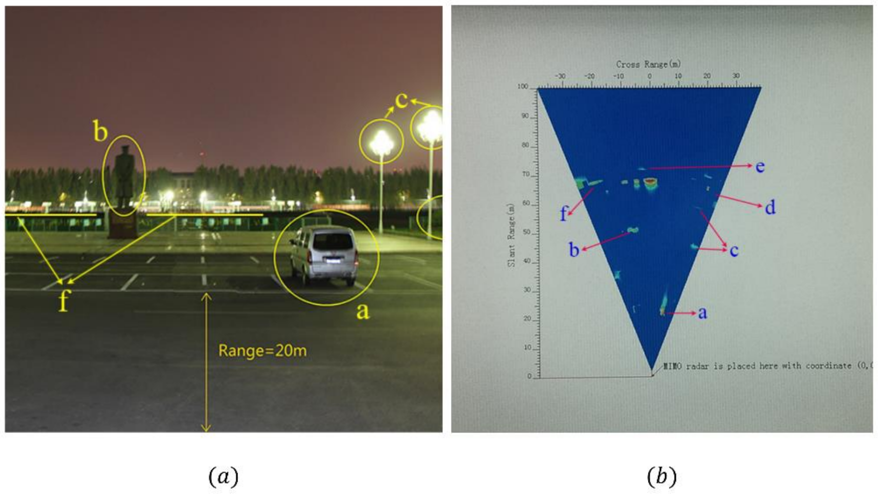

Figure 2-24. Radar Image Output: (a) Field of View (b) Radar Image, reproduced from [33]

2.4.2 Prototype B: OFDM MIMO Radar

A real-time 3D (range, azimuth, and velocity) MIMO radar was demonstrated in [9] using an OFDM waveform. The radar has a configuration of 4×4, as shown in Figure 2-25, and performs range/bearing localization and velocity measurements [9]. To reduce the cost and challenges associated with the design and acquisition of RF components and hardware, the authors opted to leverage new Software Defined Radio (SDR) technology as their RF front end [9].

The antennas are connected to four X310 radios, from Ettus Research, each of which will stream the baseband samples to a computer for processing [9]. Once demodulated, the OFDM radar requires the same signal processing as an FMCW radar for range and velocity estimation. Therefore, the computation is comprised of three FFTs [9]. The first FFT, across the subcarriers, will generate range information. The second FFT, across the OFDM symbols, will generate target velocity information. The third FFT, across the 16 MIMO channels, will generate the DOA information.

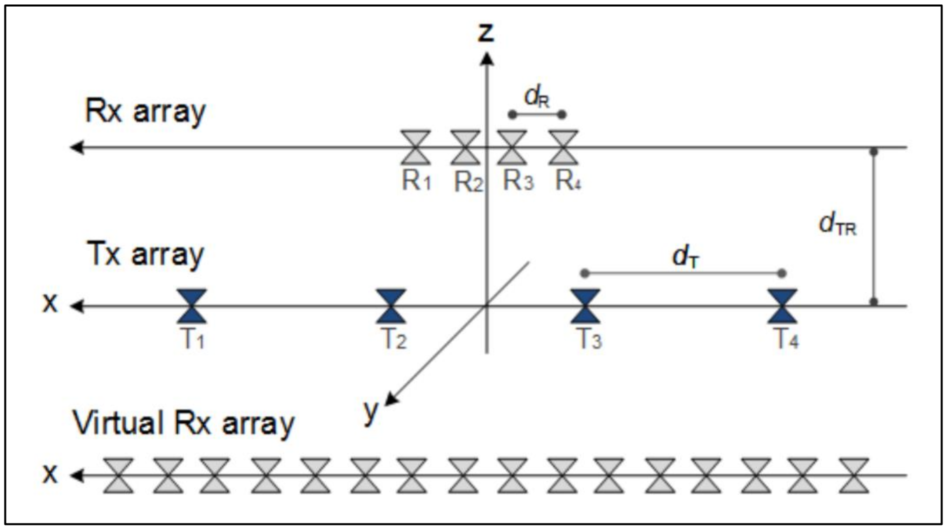

Figure 2-25. MIMO Radar Antenna Configuration, reproduced from [9]

The radar system was designed so that the demodulated OFDM data is sent to a high-performance computer for signal processing. The computation is done offline on MATLAB and takes approximately 5 seconds to perform. During the experiments, the following results were found:

Metric	Result
Range Resolution	1.5 m
Velocity Resolution	22.6 km/h
Angular Resolution	7°
Processing Time	5 seconds

Table 2-3. Performance Results, summarized from [9]

Figure 2-26 shows an example of a radar image output. The FOV has 3 reflectors placed in front of the radar at different ranges and different angles. The processed radar image clearly identifies Target A (12.5 m), but is not capable of neatly separating Targets B and C. However, the combined return of Targets B and C are in the right locations and, along with the return from Target A, it is shown that the OFDM MIMO radar works as intended [9].

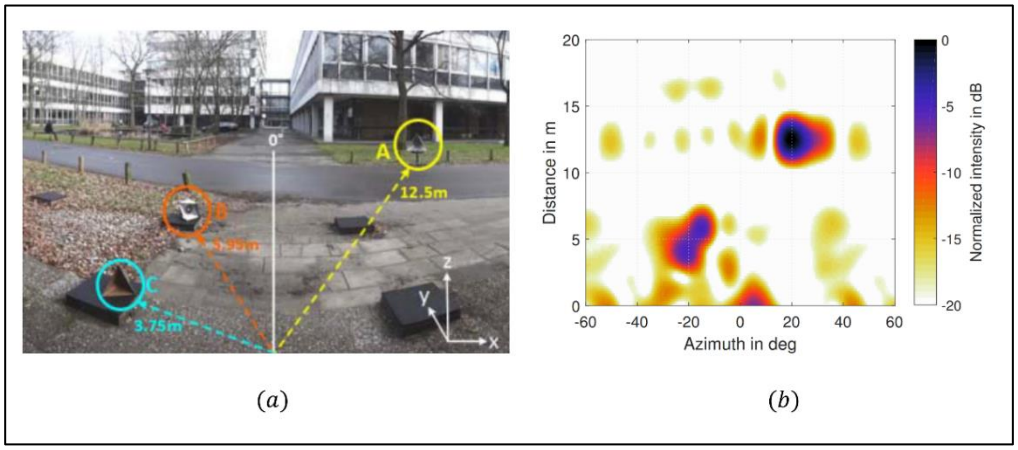

Figure 2-26. OFDM MIMO Radar: (a) Field of View (b) Radar Image after Signal Processing, reproduced from [9]

By increasing the number of OFDM symbols, the integration time was able to be set to 11.8 ms, which enabled a velocity resolution of 22.6 km/h [9]. A van travelling away from the radar at a speed of ~30 km/h was detected by the radar and its speed was correctly displayed, as shown in Figure 2-27.

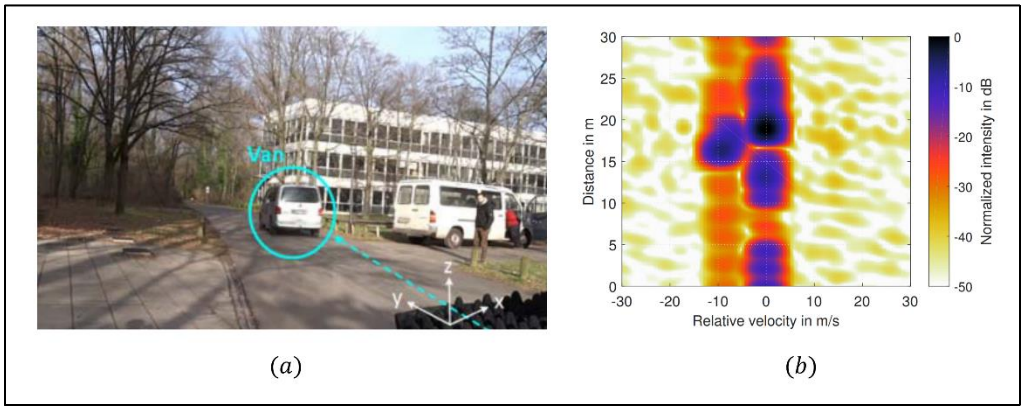

Figure 2-27. Moving Target: (a) Field of View with Moving Van (b) Radar Image after Signal Processing, reproduced from [9]

2.4.3 Prototype C: 3D TDMA MIMO Radar

A 3D (range, azimuth, and elevation) TDMA MIMO radar was demonstrated in [8]. The radar has a 24×24 configuration, uses a FMCW waveform, and performs the signal processing on MATLAB [8]. Unlike the previous 2 prototypes, this radar system's antenna arrays are located around the perimeter of the desired aperture. There are 2 arrays of 12 transmit antennas, located on the upper and lower perimeter walls. Additionally, there are 2 arrays of 12 receive antennas located on the left and right perimeter walls. This configuration creates a 2D virtual array of 576 elements and is shown in Figure 2-28 [8]. With this configuration, the angular resolution should approach 2.5° in both elevation and azimuth. Note that the spacing between the antenna elements is slightly larger than $\lambda/2$, which should increase the resolution and reduce antenna coupling [8]. Grating lobes are introduced, however, but their locations fall outside the main lobe of the radiating element ($\pm 25°$). Beamforming computations are therefore only done within this FOV [8].

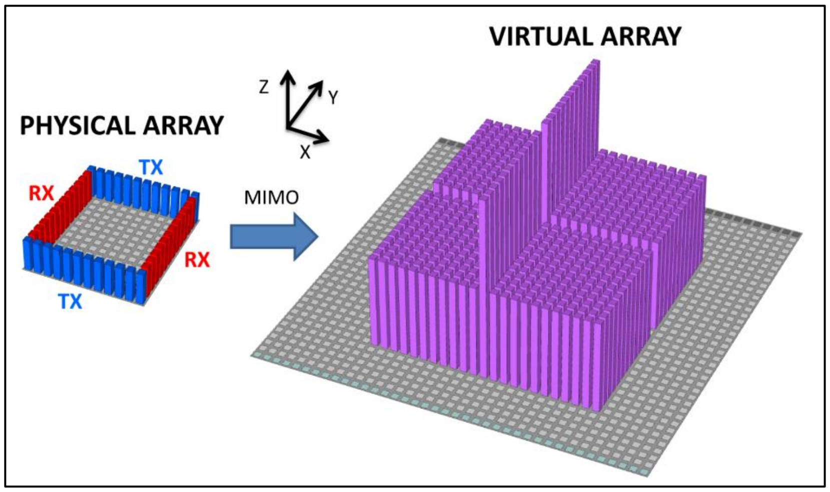

Figure 2-28. Physical and Virtual Array Configuration, reproduced from [8]

To implement the TDMA technique, each transmit element has a turn to transmit its FMCW waveform, while the 24 receive antennas simultaneously receive [8]. Transmitter boards were used which include the antennas, calibration ports, and a RF switch chain which is controllable by the system's firmware. Figure 2-29 shows the transmitter board alongside a simplified circuit diagram [8], illustrating the cascade of switches along the branches to enable the switching of the transmitter to each of the antennas.

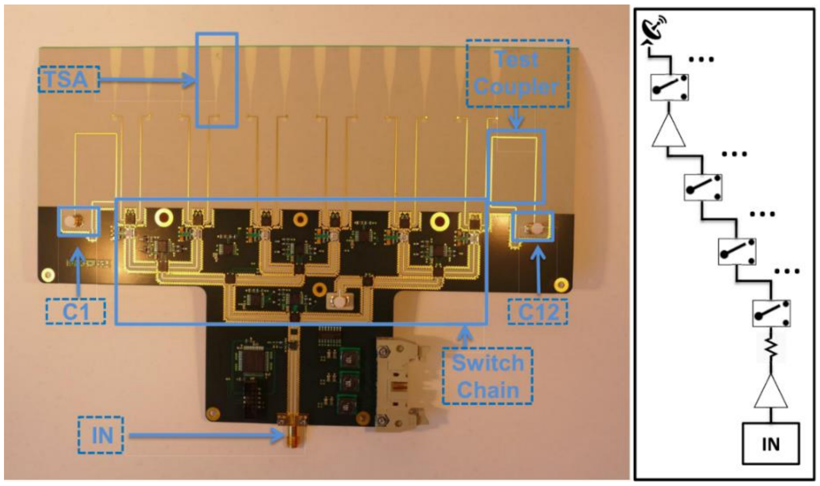

Figure 2-29. (Left) Photograph of Transmitter Board. (Right) Circuit Diagram, reproduced from [8]

Once all 576 MIMO channels have been sampled and sent to the processing computer, MATLAB reads the file and performs the computations. A range FFT is performed on each channel, followed by conventional beamforming for the azimuth and elevation estimation [8]. Note that Hanning windows are used with the FFT to reduce the Side Lobe Levels (SLL), at the cost of resolution. When including the sampling, binary file writing on the DSP, transfer to the host computer, and signal processing via MATLAB, it takes approximately 11 seconds to collect data and render a radar image frame [8].

Table 2-4 summarizes the performance results of this MIMO radar prototype.

Metric	**Result**
Range Resolution	27 cm
Azimuth Resolution	2.7°
Elevation Resolution	2.7°
Processing Time	11 seconds

Table 2-4. Performance Results, summarized from [8]

Figure 2-30 shows an experiment setup where four targets are placed in a field. The MIMO radar images the FOV and it is clear that all four targets are visible and at the correct location [8].

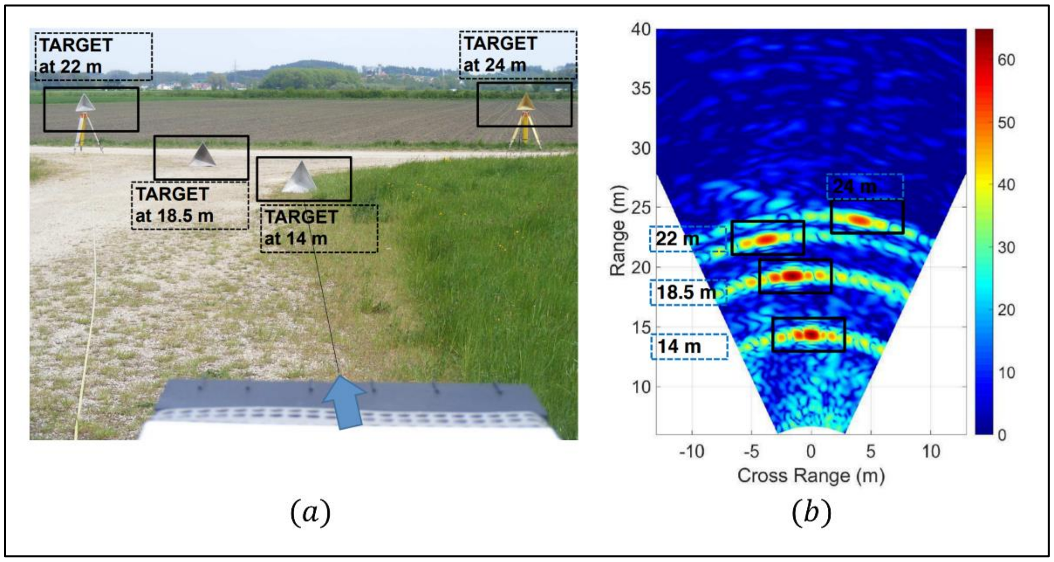

Figure 2-30. Field Test of MIMO Radar, reproduced from [8]

2.4.4 Prototype D: Distributed FMCW MIMO Radar

A 2D (range and azimuth) TDMA MIMO radar is demonstrated in [34] which operates at long ranges using a 15×5 antenna configuration. The radar system comprises of 15 transmitter antennas and 5 receiver antennas, which generate a virtual MIMO array of 75 elements [34]. The antenna positioning can be seen in Figure 2-31, alongside the rest of the radar components. Note that for simplicity, only 7 transmitter antennas are drawn.

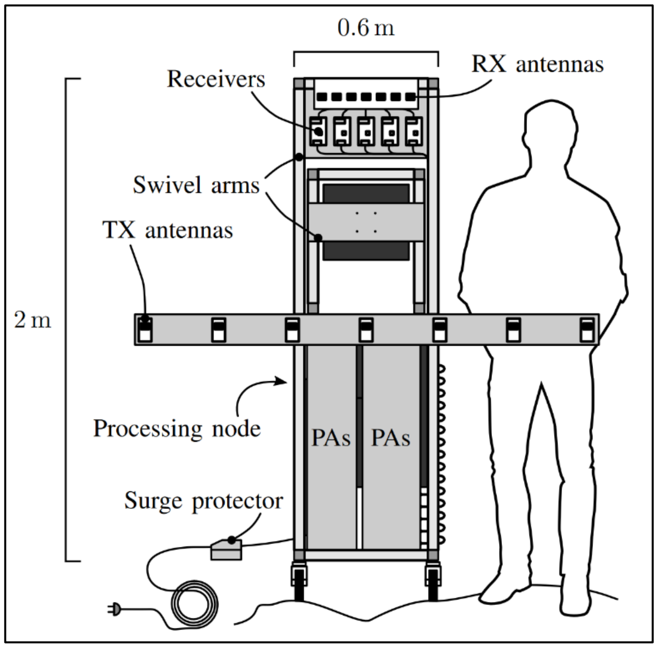

Figure 2-31. Radar Components, reproduced from [34]

The receiver and transmitter boards were developed for this project and perform front end RF functions such as amplification, mixing, filtering, and sampling [34]. The radar transmits FMCW waveforms and uses TDMA to achieve orthogonality. The received waveforms are sent to the Processing Node over high-speed Ethernet where the signal processing is performed. Unfortunately, the signal processing architecture and algorithms were not mentioned [34]. The radar, however, only processes one MIMO chirp at a time (one chirp from each transmitter) and can generate radar images at a refresh rate of 10 Hz [34]. When tested in a maritime environment, the radar was able to detect a large ship (50 meters in length) at a range of 3 km and a smaller vessel (8 meters in length) at a range of 1 km. These measurements were achieved with an instantaneous transmitter power of only 1 W [34]. Figure 2-32 shows an image of the radar prototype. The 15 transmitter antennas and 5 receiver antennas are clearly visible. Figure 2-33 shows the measurement setup with the radar located on the edge of a Greek coastline, facing the water. Figure 2-34 shows the radar image of a 50 meter long ship being detected at a range of 1200 meters.

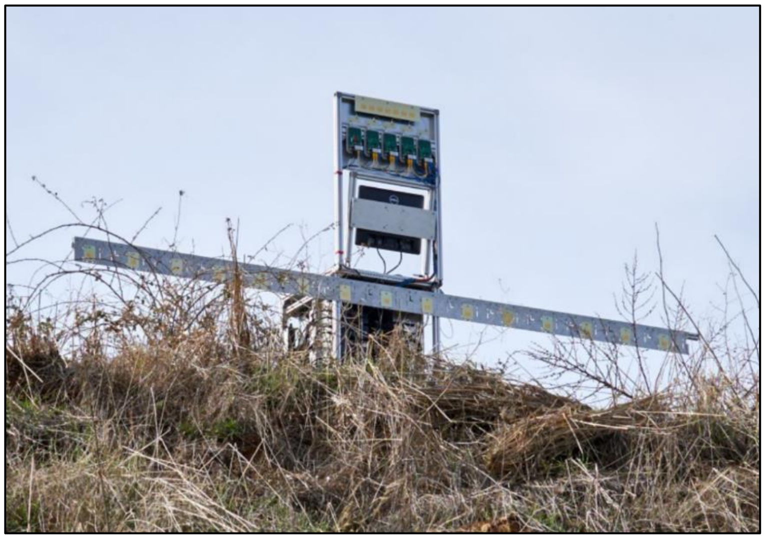
Figure 2-32. Image of Radar Prototype, reproduced from [34]

Figure 2-33. Measurement Location of Radar, reproduced from [34]

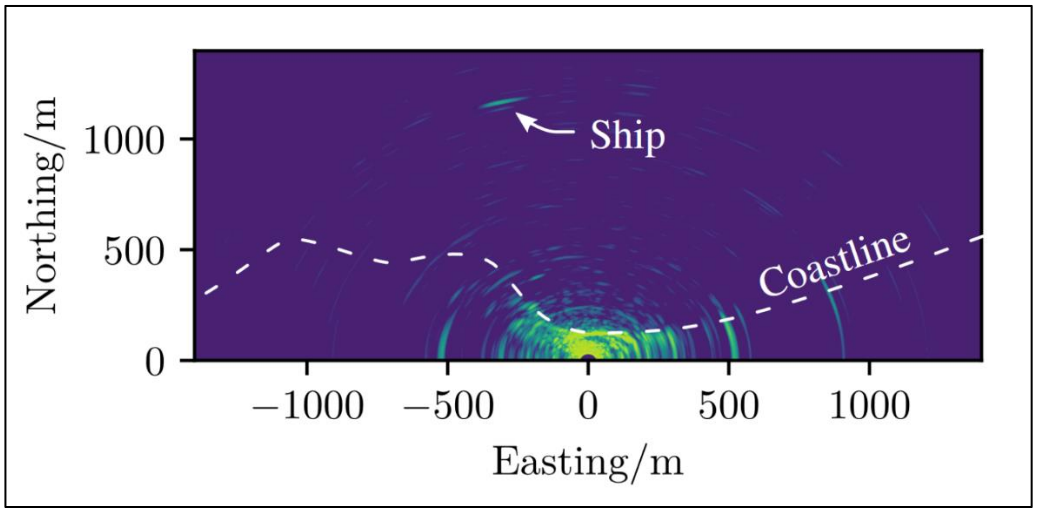

Figure 2-34. Radar Image of Ship at 1200 meters, reproduced from [34]

The radar operates at a frequency of 3 GHz, and its waveform has a bandwidth of 50 MHz. With the 15×5 configuration, an angular resolution of 2.4° was measured. The results are summarized in Table 2-5.

Metric	Result
Frequency of Operation	3 GHz
Bandwidth	50 MHz
Angular Resolution	2.4°
Refresh Rate	10 Hz

Table 2-5. Performance Results, summarized from [34]

2.4.5 Prototype E: Real Time MIMO Radar using SDRs

A real-time 3D (range, azimuth, and velocity) MIMO radar using SDR technology was demonstrated in [7]. In contrast to the other prototypes discussed so far, this radar can operate in different MIMO modes (TDMA, FDMA, CDMA) which can be selected by the user [7]. The radar has an 8×8 configuration which yields 64 MIMO channels. The SDRs mix the received signals with the Local Oscillator (LO) and sends to sampled data to a workstation via 10G Ethernet for signal processing and user display [7]. Figure 2-35 shows the MIMO radar. It is comprised of the transmit and receive arrays, the SDRs and synchronizing clock, the processing workstation, and the user interface.

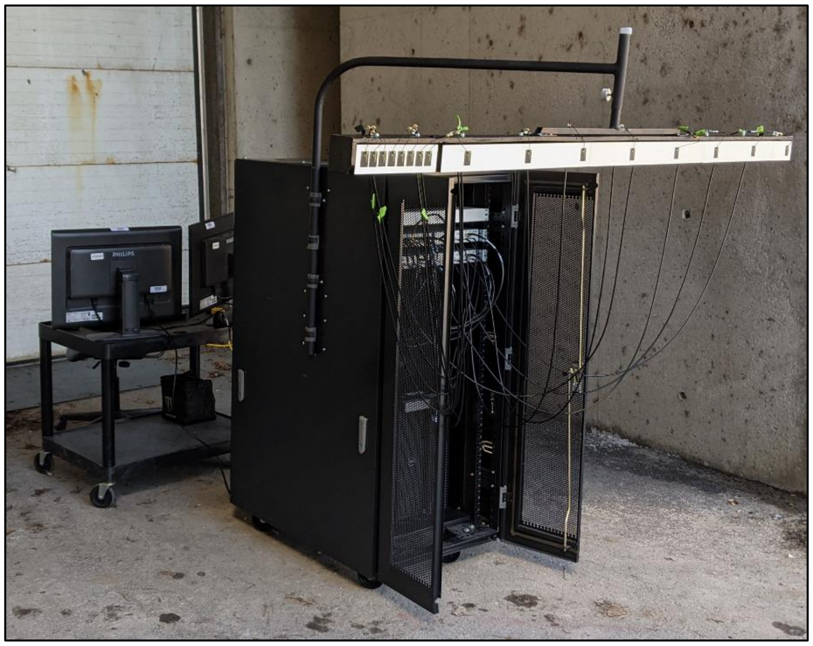

Figure 2-35. MIMO Radar photograph, reproduced from [7]

The control of the radios and the collection of samples is managed by a c++ program using the USRP Hardware Driver (UHD) Application Programming Interface (API). GNU Radio was chosen as the signal processing program and performs demultiplexing, MIMO radar tasks, visualization, and data recording [7]. The MIMO signal processing chain is done by performing the Range FFT, Doppler FFT, and Angular FFT in sequence. Finally, the data is prepared for visualizing the range-doppler and range-bearing graphs. Figure 2-36 shows a simplified block diagram of the signal processing chain.

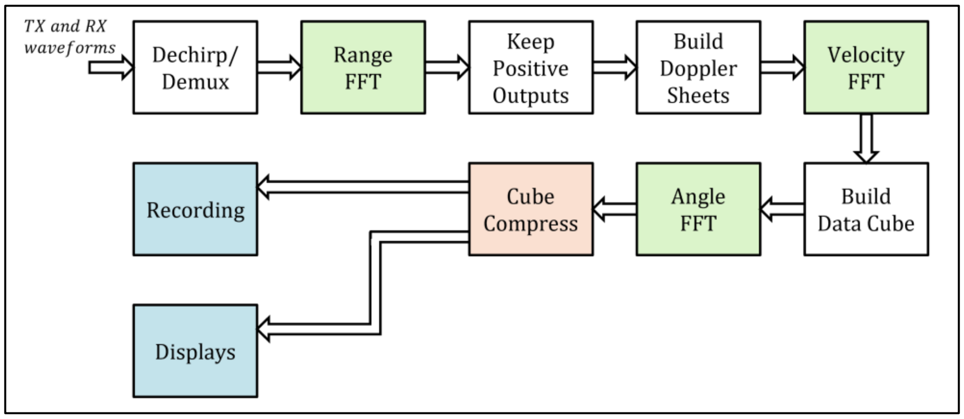

Figure 2-36. Simplified Signal Processing Diagram, adapted from [7]

Several measurements were performed on the radar, operating in different MIMO modes (TDMA, FDMA, CDMA) to verify the functionality of the radar. Resolution measurements were also taken to qualify the performance of the radar. Due to the selected antennas and the power of the radios, target detection was limited to approximately 22 meters [7]. Figure 2-37 shows an image generated by the MIMO radar for a single target.

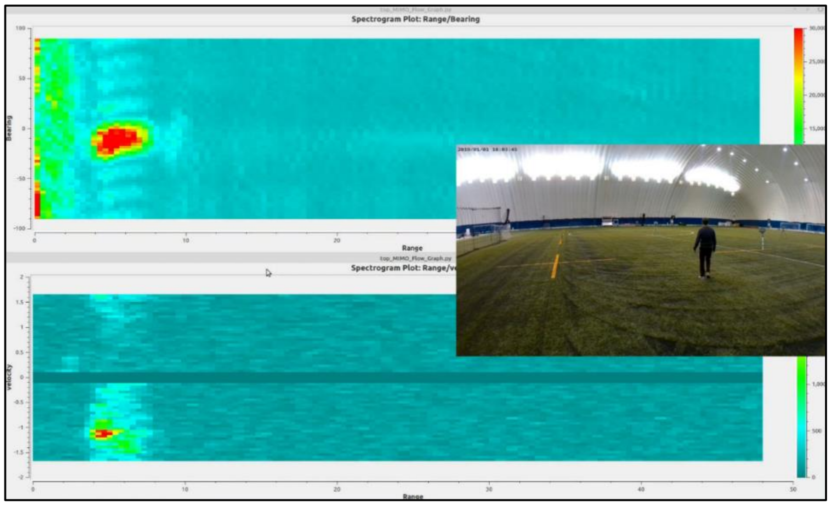

Figure 2-37. Radar Image for Single Target, reproduced from [7]

The design parameters of the MIMO radar are listed in Table 2-6. Note that since a doppler resolution of $0.1\ m/s$ is required, a CPI of 256 chirps is needed. Each SDR receiver collects 1024 samples at a rate of $250\ MSa/s$. This means that the data cube which needs to be processed for one frame has a size of $1024 \times 256 \times 64$ or $\sim\mathbf{17\ million}$ samples [7]. The GNU Radio signal processing chain is designed as a pipeline and leverages the multiple cores of the workstation's CPU by performing several sections of the pipeline simultaneously. Despite the size of the data cube, a radar frame takes ~1.25 seconds to process [7]. Table 2-7 shows the performance measurements of the radar.

Parameter	**Value**
Maximum Range	150 m
Range Resolution	0.75 m
Velocity Range	-10 m/s to +10 m/s
Velocity Resolution	0.1 m/s
Angular Resolution	2°
Processing Time	"real-time"

Table 2-6. Desired Parameters, summarized from [7]

Parameter	Measured Result
Maximum Range	21.75 m
Range Resolution	1.75 m
Velocity Resolution	0.2 m/s
Angular Resolution	10.5°
Processing Time	1.25 s

Table 2-7. Performance Results, summarized from [7]

In summary, the MIMO radar capable of functioning in real-time for all three modes of multiplexing (TDMA, FDMA, CDMA). It is clear, however, that there is a discrepancy between the desired and measured performance of the radar.

2.5 Accelerated Radar Signal Processing

This final section will investigate recent research in radar signal processing acceleration. Specifically, to see how effective GPU acceleration can be when dealing with radar data and processes. Two specific papers will be summarized, to highlight the effectiveness of GPU acceleration. Additionally, other current research regarding MIMO radar signal processing will be briefly summarized.

2.5.1 SAR Motion Compensation

In Synthetic Aperture Radar (SAR), all computations are designed around a linear flight path. In reality, there will be perturbations in the aircraft’s trajectory which will introduce errors in Slant Range and in Squint Angle [35], which distorts the SAR image and require correction [35]. Because of the weight, size, and power consumption of high performance Inertial Measurement Units (IMU) they are not the ideal solution for airborne SAR, especially for Unmanned Aerial Vehicles (UAV) [35]. Instead, using the raw radar data to estimate the aircraft’s movement itself was proposed [35]. Given that the additional required computation is in addition to an already demanding application, the tasks were performed on a GPU. Figure 2-38 shows the flight path of an airborne SAR in comparison to the ideal linear trajectory.

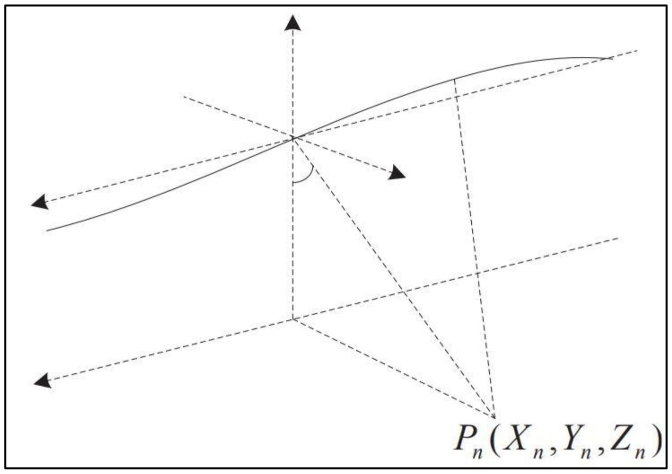

Figure 2-38. Flight Path of Airborne SAR, reproduced from [35]

The large synthetic array was subdivided into several sub-arrays. Once the radar echoes were collected for a sub-array, the data was sent to a GPU to run a kernel which estimates flight path error [35]. Figure 2-39 shows the GPU implementation approach. Each kernel launched (labeled GPU kernel 1, 2, 3, …, M) performs the algorithm for each of the sub-arrays. The computations output the corrections to the flight path required at each sub-array location which, when applied to the image processing, will refocus the image [35].

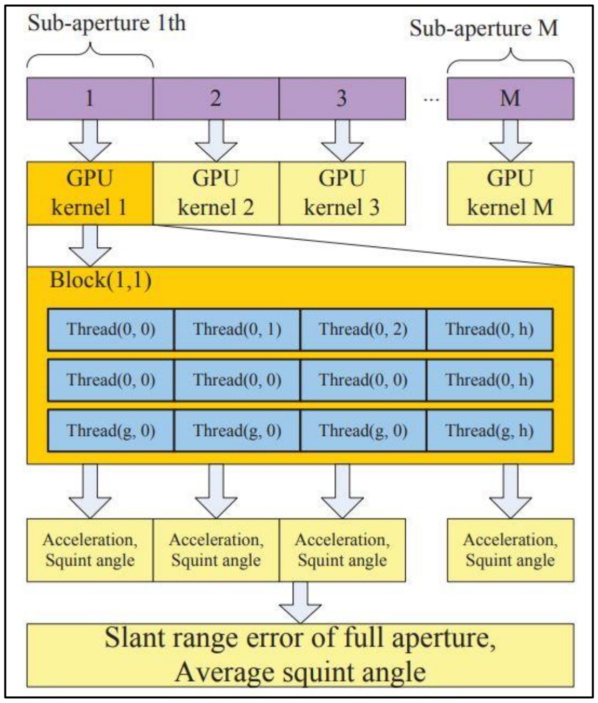

Figure 2-39. Error Estimation Method on GPU, reproduced from [35]

The CPU took 7313 seconds (~ 2 hours) to perform all the SAR processing whereas the GPU took 92.15 seconds (~ 1.5 minutes) to perform the same operations, resulting in an acceleration of ~79.36x [35]. The following SAR computations were performed:

- Range Compression
- Azimuth Compression
- Motion Parameter Estimation
- Motion Compensation
- Autofocus Algorithm
- Speckle Filtering
- Image Generation
- Distortion Correction
- Geocoding

Seeing as the GPU can perform this many algorithms much faster than the CPU, it should be more than capable of executing MIMO radar signal processing. Additionally, subdividing a large dataset and performing the algorithms in parallel enabled efficient use of the GPU. Similar techniques could be applied to MIMO radar signal processing.

Figure 2-40 shows the difference in the quality of a SAR image when Motion Compensation is used.

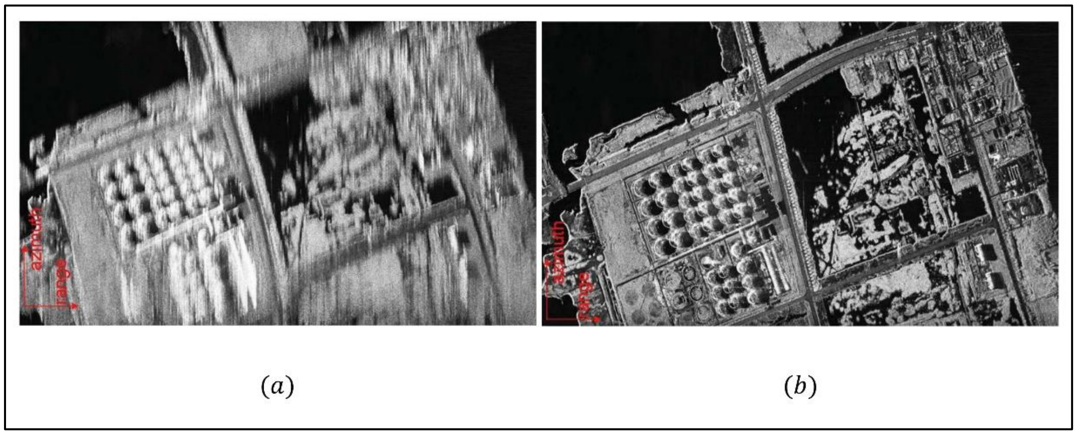

Figure 2-40. SAR Image Comparison: (a) Without Compensation (b) With Compensation, reproduced from [35]

2.5.2 Radar Signal Processing of Weather Radar

The signal processing of a dual polarization FMCW weather radar was executed on a GPU in [36]. Several processes need to happen on the weather radar such as FFTs (range and doppler processing), clutter suppression, power calculations, spectrum smoothing, mean doppler velocity calculations, and reflectivity depolarization ratio to name a few [36]. These calculations must all be done on each of the polarization permutations of the radar (horizontal-horizontal, vertical-vertical, and horizontal-vertical).

Several CUDA kernels were implemented to perform key tasks in parallel such as applying windowing to input signals, taking averages and complex conjugates of signal samples, performing FFTs and IFFTs, and squaring the values of samples [36]. These types of operations are required across many different radar algorithms and can add significant delay to the signal processing time as they are usually performed on a per-sample basis. The ability to execute these tasks in parallel on a GPU should help reduce the execution time. Unfortunately, it was not mentioned if the signal processing for each polarization mode was also done in parallel.

The weather radar functions were performed on a CPU and on a GPU to evaluate the execution times and to determine the acceleration [36]. For the experiment, an Intel® Core™ i5-6200U (2 cores) CPU and an NVIDIA® GeForce® 930M GPU were used [36]. Unfortunately, there was no detail on the size of the processed data nor the parameters of the radar itself. The following are the measured execution times:

- CPU: 15426.43 ms
- GPU: 6133.19 ms

Parallelization of signal processing tasks of the weather radar was successful, and an acceleration of approximately 2.5x was achieved [36]. Applying the techniques in [36] as well as parallelizing the signal processing of demultiplexed channels should yield positive results when used in MIMO radar.

2.5.3 MIMO Radar Signal Processing

Several relevant papers have been recently published which deal with the signal processing aspect of this work. For one, parallel implementations of MIMO radar signal processing have been investigated. However, much of the work have been focused on implementing solutions on multiple-core DSPs or on FPGA boards, yielding modest results [37], [38]. GPU parallelization of MIMO radar signal processing is relatively novel, and current implementations only use Fast Fourier Transforms (FFT) for the analysis of range, doppler, and bearing information [39]. The acceleration currently achieved from performing this processing on a GPU is approximately 150x, when compared to its execution on MATLAB [39].

Developments in beamforming techniques are also of great interest in MIMO radar systems. Particularly, high-resolution beamformers which function on single-snapshot datasets are highly sought-after. Modifying known super-resolution techniques (such as MUSIC) for a MIMO configuration yields better results than the Single Input Multiple Output (SIMO) counterparts [40]. Unfortunately, as discussed in section 2.2, most super-resolution beamforming techniques have unique requirements which may not be met, such as having a priori knowledge of the targets or guaranteeing that multiple targets are **not** correlated. However, in order to address the added complexity of some of these high-resolution methods, attempts have been made to parallelize them on GPU [41]. Unfortunately, the parallelization of the MVDR algorithm on GPU (via MATLAB) did not yield good results, where it was concluded that larger speed ups could be achieved by developing the GPU code through CUDA [41].

2.6 Summary

As seen in section 2.4, there has been great success with MIMO radar prototypes. However, a common issue is the increased computational requirement of MIMO signal processing. This is especially true when doppler processing is required and the radar needs to collect large CPIs. Not all prototypes executed in real-time, but those that did resorted to one (or several) of the following techniques to increase the radar's throughput:

- Choosing a faster algorithm at the cost of resolution (i.e., the FFT);
- Reducing the quantity of data to be processed (i.e., compute only some angles and some ranges); or
- Offloading the computation on FPGA or DSP.

In section 2.5, GPU acceleration of radar signal processing was investigated. It was found that large acceleration is achievable across many different algorithms, due to the quantity and parallelism of the radar data. GPU acceleration is therefore proposed as a solution to accelerate MIMO radars. Specifically, the proposed algorithm will aim to not only accelerate the baseline MIMO radar signal processing chain, but also incorporate different algorithms which will increase the quality of the produced images.

3 Methodology

The methodology chapter provides a full overview of the proposed signal processing and its implementation. Section 3.1 outlines the simulated environment and the generic program for the MIMO signal processor. Section 3.2 presents an overview of the physical constraints of the radar environment and describes the configuration setup. Section 3.3 describes the sequential and parallel implementations of the Chirp Z Transform, the Bartlett Beamformer, and the Cube Compression algorithm. Section 3.4 defines the problem of beamforming at short ranges and describes the required corrections. Finally, Section 3.5 describes an optimization method which enabled CPU caching when executing the Bartlett Beamformer on the CPU.

The proposed solution is independent of the type of MIMO waveform multiplexing (i.e., TDMA, FDMA, CDMA, etc.) as it is applied on the baseband signal after demultiplexing. The proposed solution is also designed for the FMCW waveform. Using the proposed solution with a pulsed waveform is possible, but the first series of FFTs must be modified as to perform matched filtering of the pulses.

3.1 Simulation Environment

Before signal processing solutions were designed and tested on the physical radar, a Simulated Environment was developed. Its role is to simulate target echoes, given a MIMO configuration for varying target ranges, bearings, speeds, and headings. Since resolution measurements will be performed on the signal processing solutions, the simulated environment must be able to generate echoes from at least 2 targets. The scope of this thesis is restricted to accelerating already proven algorithms for MIMO radar, therefore the output of the simulated environment will be the ideal baseband I and Q voltages after de-chirping and demultiplexing the MIMO waveforms. Since the radar in question uses a FMCW waveform, the baseband signals have a beat frequency and a relative phase offset, which will depend on the pulse number and the MIMO channel number.

To demonstrate a simple case, a single target's range is evaluated for a 1×1 antenna configuration. Figure 3-1 shows the geometry of this configuration for a target located at some range R and some azimuth angle θ from the radar's reference point $P(0,0)$.

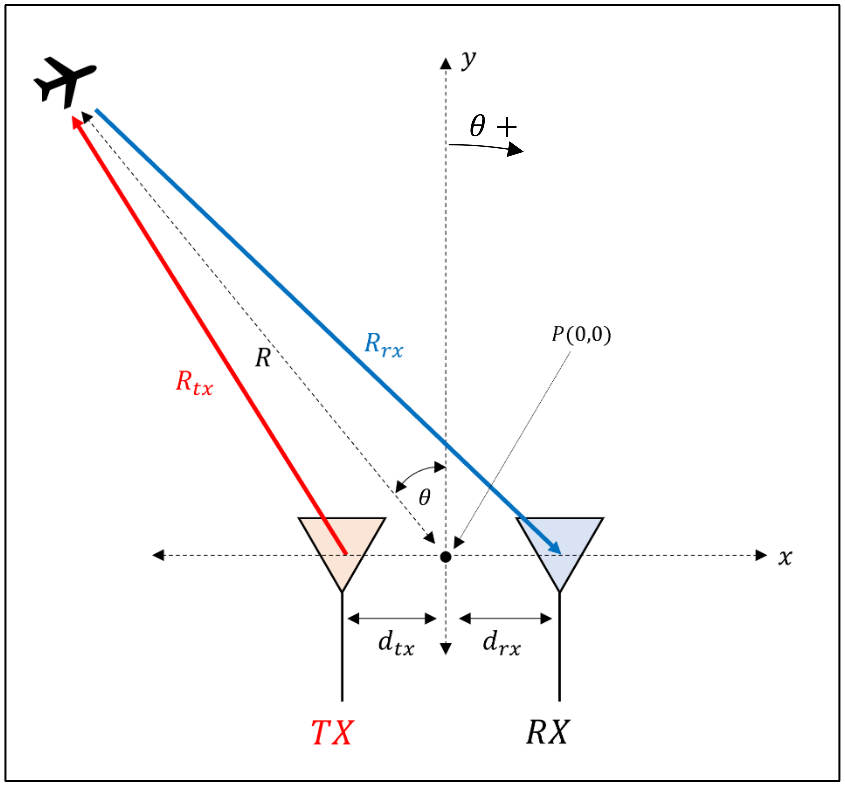

Figure 3-1. Antenna and Target Geometry, Single Target

If (2.1) is manipulated to include the variables from Figure 3-1, the expression can be arranged to isolate the time delay in this bistatic configuration as:

$$\Delta t = \frac{R_{tx} + R_{rx}}{c} \tag{3.1}$$

where:

Δt = round trip delay (s)
R_{tx} = range between target and transmitting antenna (m)
R_{rx} = range between target and receiving antenna (m)
c = speed of light (m/s)

For the simulated target echoes, the calculations of R_{tx} and R_{rx} is done for every permutation of transmit and receive antenna, as a function of the MIMO array configuration. Additionally, as the targets moves due to their speed and heading, the range measurements will change. This is important as the radar collects several pulses to form its CPI, which will enable doppler processing.

For each simulated chirp the positions of the targets are calculated, and baseband I and Q voltages are generated at the correct beat frequencies using (2.3). Additionally, relative phase offsets are added to properly simulate the doppler shifts and the different MIMO channel ranges resulting from the different Tx and Rx antenna locations. The output of the Simulated Environment is a data cube, and its size will depend on the radar parameters (i.e., number of transmit antennas, receive antennas, sampling rate, chirp time, number of chirp per CPI). Figure 3-2 shows an example data cube for a radar with 10 virtual channels, 40 I and Q samples per channel, and a CPI of 20 chirps. Note that each dot represents a sample.

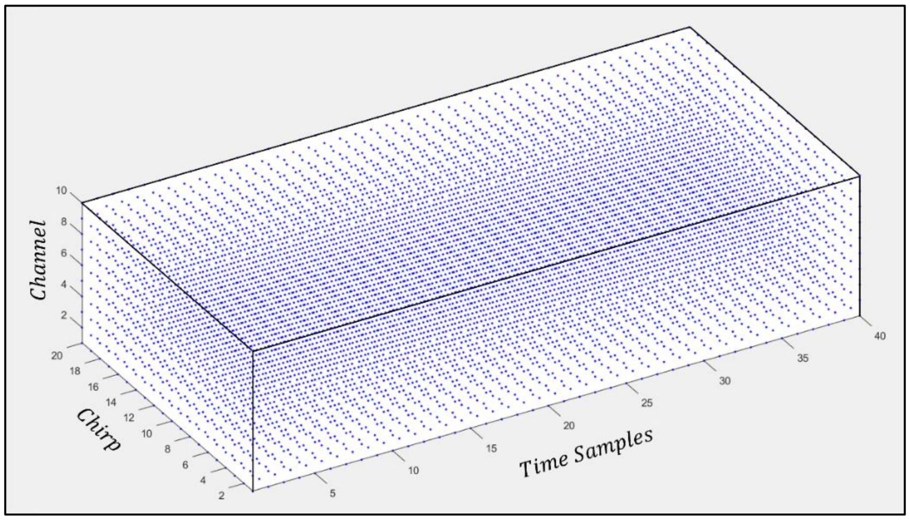

Figure 3-2. Data Cube Example, $40 \times 20 \times 10$

Once the Simulated Environment generates the echoes for a specified number of targets, the data cube can be sent for signal processing. Once the signal processing has been tested and verified, the Simulated Environment for target echo generation can simply be replaced by the real-time acquisition of radio samples from the SDRs, using UHD commands as in [7].

3.2 Configuration Setup

The algorithms developed herein is designed to replace the existing GNU Radio signal processing chain on the physical radar [7]. Therefore, the radar design parameters, given in Table 3-1, and settings are key to the algorithm’s implementation. All parameters have been chosen in order to meet the performance requirements listed in Table 2-6 (Section 2.4.5) [7]. The data cube collected by the radar has a size of $1024 \times 256 \times 64$, meaning that for each radar frame, nearly 17 million complex samples must be processed to produce the range-doppler and range-bearing images.

Parameter	Value
Maximum Range	150 m
Frequency of Operation	5 GHz
Waveform Type	Linear FMCW
Waveform Bandwidth	200 MHz
Radio Type	Ettus Research™ N3xx
Number of Radios	4
Sampling Frequency	250 MHz
Chirp Duration	4.096 µs
Number of Tx Antennas	8
Number of Rx Antennas	8
Number of MIMO Channels	64
Number of Samples per Channel	1024
Number of Chirps per CPI	256
CPI duration	308 ms
Chirp Repetition Frequency	830.5 Hz

Table 3-1. MIMO Radar Parameters, summarized from [7]

The Local Oscillators (LO) of each N320 radio is synchronized from the N321 SDR LO outputs. Additionally, a CDA-2990 Clock Distribution Unit supplies a common time and reference signals to the 4 radios. Each radio also has a 10G SFP+ Ethernet connection to the Host Processor, as shown in Figure 3-3. Detailed connectivity of the MIMO radar setup can be found in [7].

The Workstation (also known as the Host Processor) is a Dell Precision 7920 containing 2 Intel® Xeon Gold 5120 processors (14 Cores each) and an NVIDIA GeForce 2080 Ti GPU. Figure 3-4 illustrates the workstation along with its hardware.

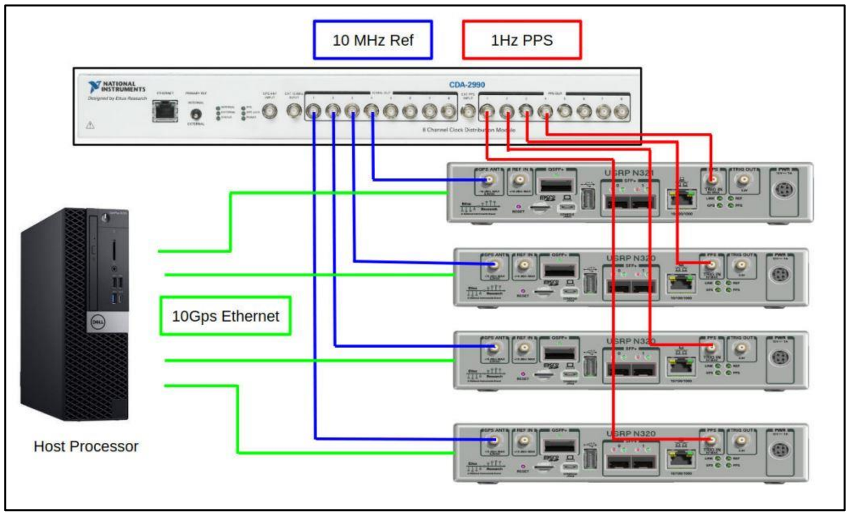

Figure 3-3. Connectivity Diagram, reproduced from [7]

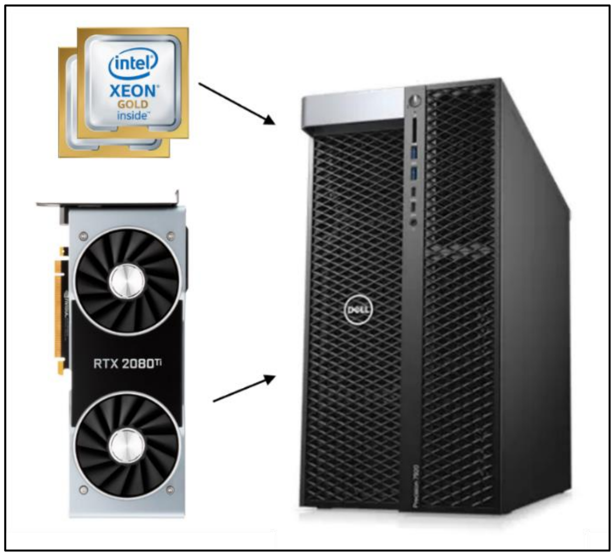

Figure 3-4. Dell Precision 7920 Workstation

Several commercial software packages are required and are listed in Table 3-2. Note that the UHD version on the Host Processor and on the SDRs must match. The process to change the version of UHD of the radios can be found on the Ettus Research™ website [42].

Software Name	Notes	Version
Linux Mint Cinnamon	Operating System (OS)	20.1
CLion	Integrated Development Environment (IDE)	2021.2
MATLAB	Scientific programming platform	2021b
UHD	Software API which supports USRP devices	3.15.0
FFTW	Performs efficient DFT subroutines	3.3.10
CUDA Toolkit	Libraries, compilers, and development tools for CUDA programs	10.1.243

Table 3-2. Software Versions on Workstation

The C++ program controls the radios and receive the sampled signals using the UHD library. Signal processing will be performed either on the CPU and on the GPU using the appropriate library or toolbox (FFTW and CUDA). OpenMP (OMP) enables control over the threads and allows parallelization on multicore CPU. Graphics will be generated using the **pcolor** function via the MATLAB Engine API [38]-[39]. Figure 3-5 shows a simplified block diagram showing the software elements (green) within the radar architecture.

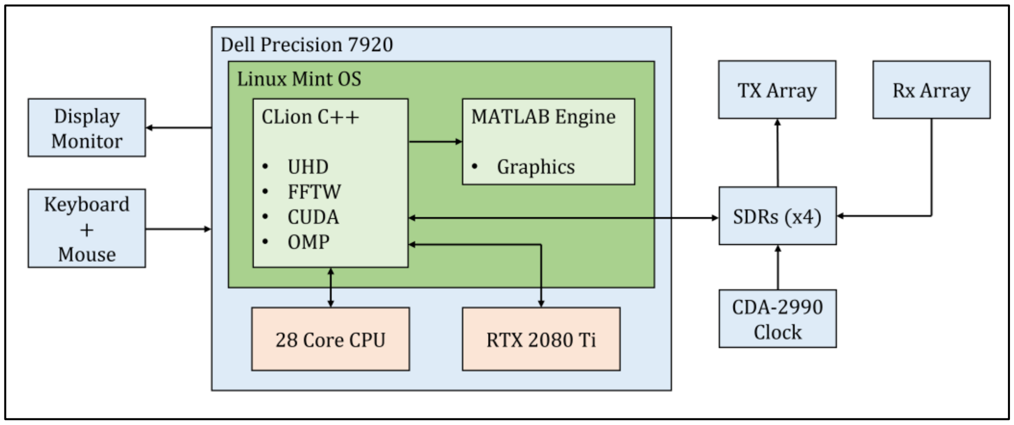

Figure 3-5. Simple Software Diagram

3.3 Algorithms

This section will cover the MIMO radar signal processing chain. First, the existing 3D FFT method will be described, followed by the implementation of the CZT and the Bartlett Beamformer, and concluding with the parallelization of the algorithms.

Regardless of the method is used to generate the MIMO radar images, five distinct steps must be performed in the signal processing chain, namely:

- Initial Setup
- Sample Collection
- Radar Signal Processing
- Display Update
- Program Shut Down

The Initial Setup initializes the program with regards to the radar parameters and any user entries. Then, communication with the SDR radios is established and an instance of MATLAB is opened via the Engine API. Additionally, Host and Device array memory, required for the computation, is allocated.

The Sample Collection stage can mean either the generation of target echoes from the Simulation Environment, or the collection of baseband samples from the radios. Either way, the output of this step is demultiplexed I and Q samples required for signal processing.

The Radar Signal Processing stage can include any process that is performed on the data cube, such as the baseline 3D FFT and cube compression to generate the range-doppler and range-bearing data matrices [7]. In general, other radar functions could be added here such as CFAR or tracking functions.

The Display Update stage refreshes the image seen by the radar user, which involves calling the **pcolor** function on the newly processed sets of data or “frame”. Note that steps 2 through 4 are done iteratively for as long as the radar is running.

The last stage would be to do a proper shut down of the program. This could mean freeing Host and Device memory, clearing the FFT plans, and closing the instance of MATLAB. Figure 3-6 shows a block diagram of these steps for the baseline 3D FFT method.

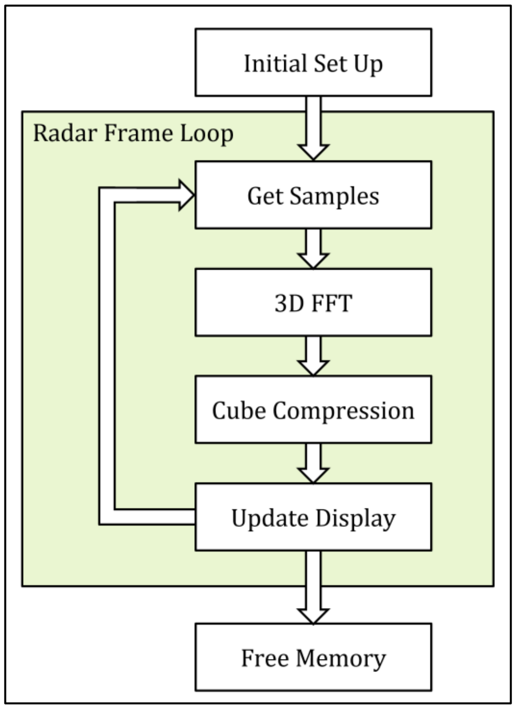

Figure 3-6. Stages of Computing the Radar Frames for the Baseline FFT Method

3.3.1 Baseline Method: 3D FFT

The Fastest Fourier Transform in the West (FFTW) is a C library which performs fast DFT subroutines on input data arrays having arbitrary size and dimension [45]. The data can be real or complex and the DFTs support different precisions (i.e., float or double). With the FFTW API, computing a 3D FFT is relatively simple. An FFT plan must be created which defines the location of input and output data memory (as pointers), and the type of FFT (FFT or IFFT). Once data is collected and stored in the correct memory location, the FFT can be executed by calling the **fftw_execute** function. A list of FFTW functions used for 3D FFTs is found in Table 3-3.

Function	Notes
fftw_plan_dft_3d( )	Creates an 3D FFT plan. The size of the data cube must be specified, as well as the pointers to the input and output arrays. There is also an option to specify the FFT or the IFFT.
fftw_execute( )	Takes an FFT plan as input and performs that FFT. Can be executed as often as required.
fftw_destroy_plan( )	This function destroys the FFT plan and frees memory allocated to it once the program is done.

Table 3-3. FFTW functions used in Baseline FFT method. Notes from [45]

Once the FFTs have been completed, the next step in the signal processing chain is to compress the 3D cube into 2D images for range-doppler and range-bearing visualisation. The data cube is composed of many range-bearing sheets (one for each doppler bins) or from many range-doppler sheets (one for each bearing bin). To calculate the range-bearing pixels, the power sum of the range-bearing bins is taken across all doppler sheets, and the process is repeated in the bearing direction to produce the range-doppler pixels, as illustrated in Figure 3-7 and described by (3.2) and (3.3).

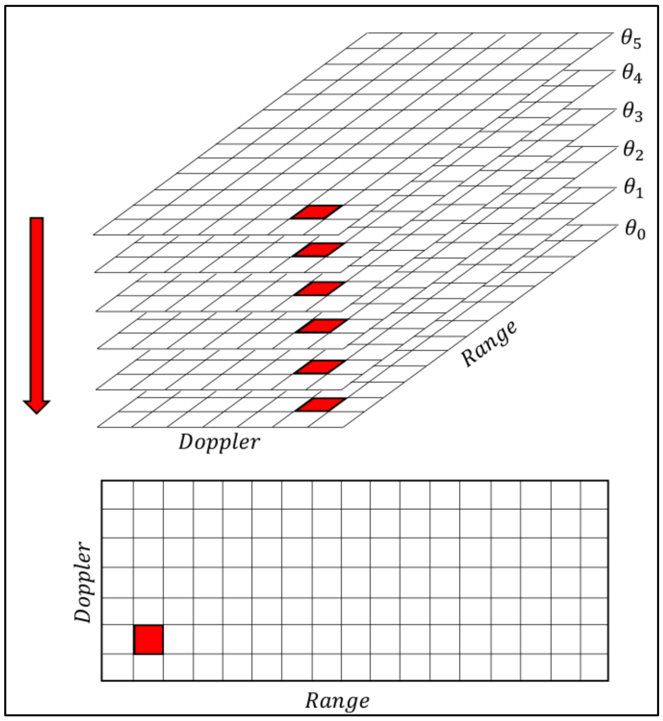

Figure 3-7. Visualization of Cube Compression

$$Out[r,b] = \sum_{d=0}^{N_d-1} |x[r,b,d]|^2 \tag{3.2}$$

$$Out[r,d] = \sum_{b=0}^{N_b-1} |x[r,b,d]|^2 \tag{3.3}$$

where:

Out = pixel amplitude (real positive value)
N_d = number of doppler bins
N_b = number of bearing bins
r = range bin
b = bearing bin
d = doppler bin
x = processed cube samples (complex values)

In [7], the processed data cube is very large. There are 512 range samples, 256 doppler bins, and 64 bearing bins for a total of ~8 million complex samples. The range-bearing image contains 32,768 pixels and the range-doppler image contains 131,072 pixels. The 3D FFT and Cube Compression operations, being computationally intensive, are slow to execute on a CPU and are expected to benefit the most from parallelization.

The radar collects its CPI of 256 pulses at a rate of 830.5 Hz, which takes approximately 308 ms. Therefore, even if the signal processing could be done instantaneously, the refresh rate of the radar has an upper limit of 3.24 Hz. When executed as a pipeline on GNU Radio in [7], the signal processing took 1.25 seconds which translated to a refresh rate of only 0.64 Hz.

3.3.2 Enhanced Resolution: CZT and Bartlett

As described in Chapter 2, the resolution of the FFT is sometimes too coarse for the intended application. To increase the resolution of the radar, the Chirp Z Transform will replace the range FFT and the Bartlett DOA will replace the bearing FFT, as illustrated in Figure 3-8.

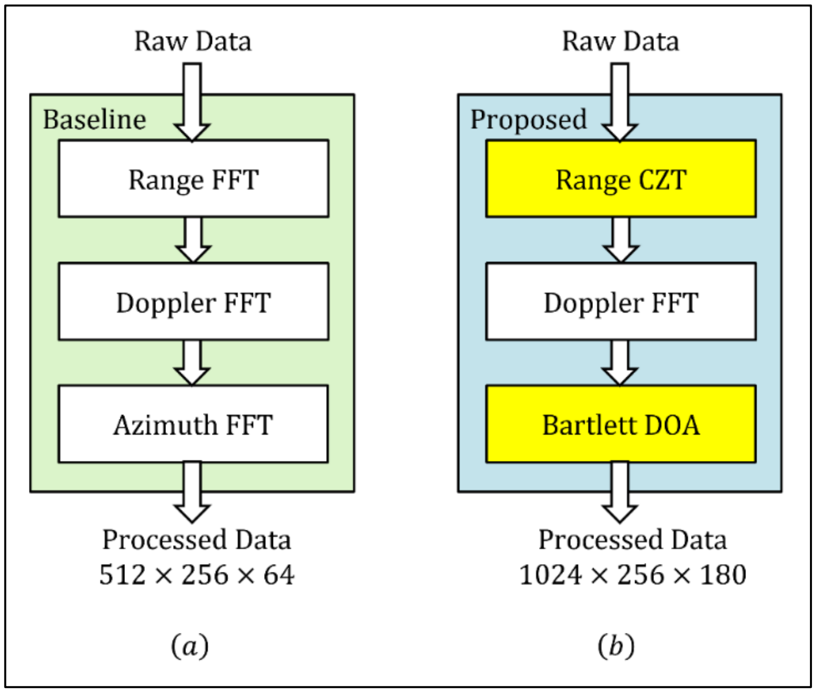

Figure 3-8. Algorithm Substitutions: (a) Baseline 3D FFT (b) Proposed

In the existing baseline solution, the range FFT computes the 1024 range bins and keeps the positive half of the results (as negative range values are nonsensical). By combining (2.4) and (2.8), the expression for the range axis, after the FFT, is obtained:

$$R[i] = \frac{cf_s}{2k_0} \times \frac{i}{N} \tag{3.4}$$

where:

R = range (m)
i = range bin index where $0 \leq k < N/2$
k_0 = chirp slope (Hz/s)
c = speed of light (m/s)
f_s = sampling rate (Hz)
N = number of raw data complex samples

From (3.4) and the information from Table 3-1, the range resolution is 75 cm. Additionally, when $k = 511$, the maximum computed range has a value of 383.73 meters, which is much larger than the radar's designed maximum range of 150 meters. Losing half the sample outputs and computing results outside the ranges of interest of the radar is not an effective use of the 1024 samples that are collected from the SDRs. With the CZT, only the ranges of interest between 0 and 150 meters will be computed, using all 1024 samples. Using (2.4) and the frequency axis from (2.11), the range axis of the CZT is calculated to be:

$$R_{CZT}[k] = \frac{c(f_1 + k\Delta f)}{2k_0 N} \tag{3.5}$$

Since the ranges of interest are between 0 and 150 meters, (3.5) may be simplified so that f_1 and f_2 are set to the beat frequencies which represent 0 and 150 meters, respectively:

$$R_{CZT}[k] = \frac{ckf_2}{2k_0 N} \tag{3.6}$$

Using (2.4), the beat frequency of a target at 150 meters is approximately 48.828 MHz resulting in a range step of 14.65 cm. The CZT range axis is therefore sampled approximately 5 times more often than for the existing FFT implementation, resulting in a finer representation of the waveform's shape and width. Figure 3-9 demonstrates this by showing two point-targets (46 and 52 meters) after range processing using both the FFT and the CZT method. Note that both algorithms used the same data set to generate their outputs. The I and Q samples were generated by the simulated environment, as described in Section 3.1, using the radar parameters listed in Table 3-1.

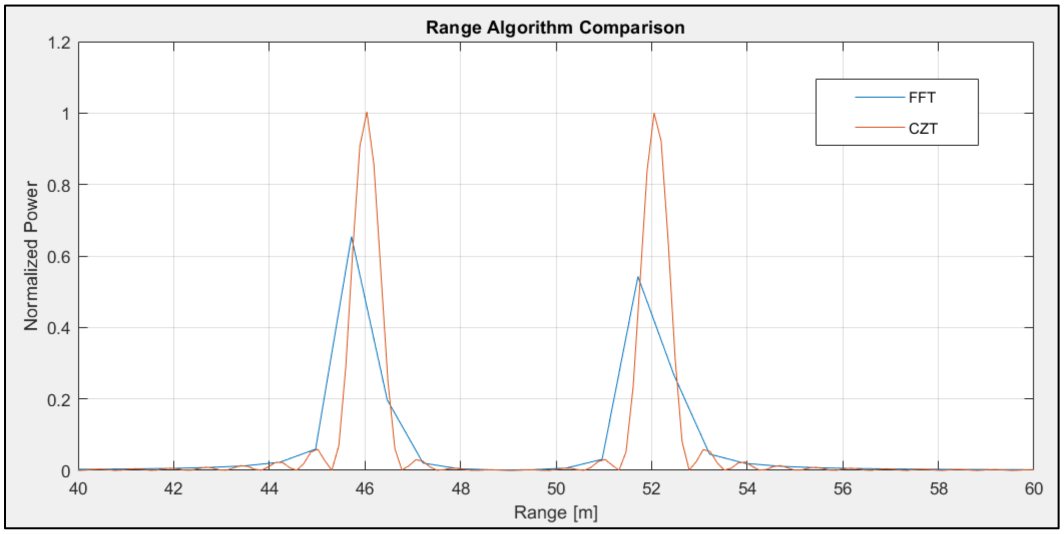

Figure 3-9. Range FFT and CZT Comparison for 2 Targets

As shown in Figure 3-10, the CZT needs to be executed many times in order to cover the entirety of the data cube. However, since the radar parameters remain the same during a CPI, some calculations in (2.11) only need to be performed once. The following arrays are therefore computed in the Initial Setup stage of the program, and are used for every call of the CZT function:

- $W[n] = \exp\left(-\frac{j\Delta\omega n^2}{2}\right)$
- $W_{inv} = \text{FFT}(W^{-1})$
- $B[n] = \exp(-j\omega_1 n)$

As in Figure 2-10, and expressed in (2.11), only the input data $x[n]$ will differ as the CZT gets called throughout the cube. The complexity of the CZT is therefore reduced to the computations operating on $x[n]$, which involve 3 multiplication steps, an FFT, and an IFFT.

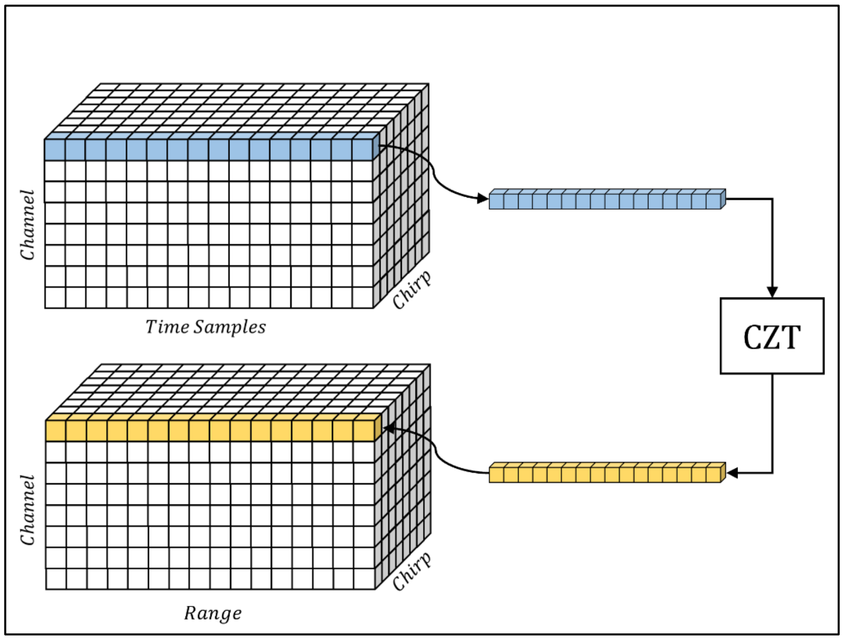

Figure 3-10. CZT, within Context of the Data Cube

Algorithm 1 shows the steps used to compute the CZT where $x[n]$ represents the input data (time samples), and $X[k]$ represents the output data (range).

Algorithm 1: Chirp Z Transform

```
1:  for every channel/chirp set of 1024 time samples{
2:      $y = x \cdot B \cdot W$
3:  }
4:  Perform in-place transform across entire cube $y = \mathrm{FFT}(y)$
5:  for every channel/chirp set of $y$ values{
6:      $y = y \cdot W_{inv}$
7:  }
8:  Perform in-place transform across entire cube $y = \mathrm{IFFT}(y)$
9:  for every channel/chirp set of $y$ values{
10:     $X = y \cdot W$
11: }
12: end of CZT function
```

Unlike the FFT and CZT method, the Bartlett DOA is an iterative calculation. As seen in Chapter 2, the spectral power needs to be calculated for each angle of interest. If the FOV ranges from $-90°$ to $90°$ with incremental steps of $1°$, the algorithm would need to calculate (2.15) 180 times to generate the DOA for a single range/doppler bin. Figure 3-11 illustrates this in the context of the data cube where the Bartlett DOA is being performed on the corner range/doppler bin and only the first two angular bins have been calculated. Note that the 2-way half power beamwidth (HPBW) of the radar system is 1.79 degrees at broadside [7].

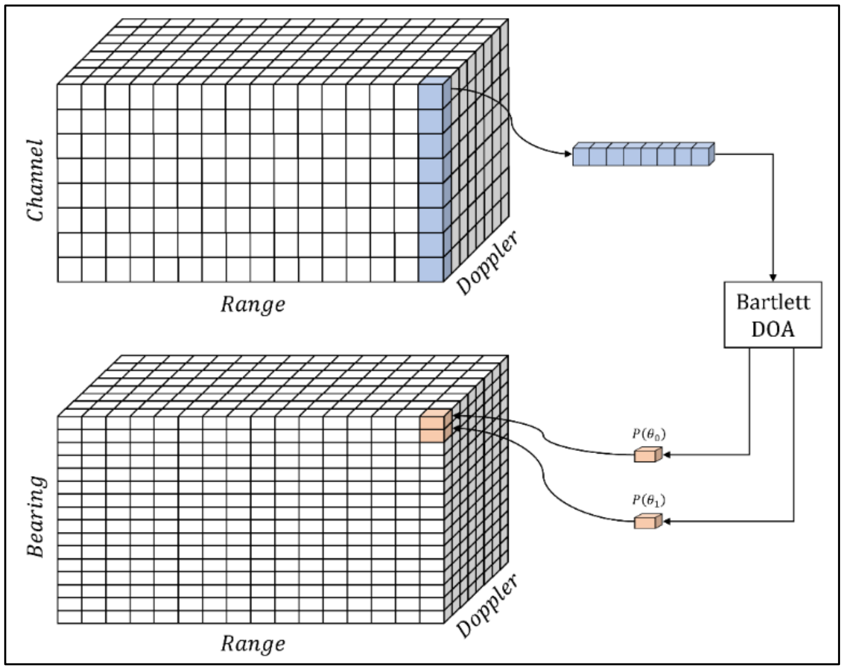

Figure 3-11.Bartlett DOA, in Context of the Data Cube

Since the computations are done on a single blue column (snapshot) for each range/doppler bin, the matrix multiplication of (2.15) can be simplified. For a given range/doppler cell, the samples across the N channels are expressed as:

$$\boldsymbol{x} = [x_0, x_1, \dots, x_{N-1}] \tag{3.7}$$

The covariance matrix $\boldsymbol{R}_x$ has size $N \times N$ and is computed by:

$$\boldsymbol{R}_x = \boldsymbol{x} \times \boldsymbol{x}^H = \begin{bmatrix} x_0 x_0^H & \cdots & x_0 x_{N-1}^H \\ \vdots & \ddots & \vdots \\ x_{N-1}\, x_0^H & \cdots & x_{N-1} x_{N-1}^H \end{bmatrix} \tag{3.8}$$

For a given angle, the steering vector $\boldsymbol{a}$ represents the phase shift across all elements, given a particular wave number β and element spacing d.

$$\boldsymbol{a}_\theta = \left[0,\ e^{-j\beta d \sin(\theta) n}, \dots,\ e^{-j\beta d \sin(\theta)(N-1)}\right] \tag{3.9}$$

By performing the matrix multiplication $\boldsymbol{a}_\theta^H \boldsymbol{R}_x \boldsymbol{a}_\theta$, we get the following expression for the spectral power at the selected angle θ:

$$P[\theta] = \left(\sum_{n=0}^{N-1} a_\theta[n] x[n]^H\right)\left(\sum_{n=0}^{N-1} a_\theta[n]^H x[n]\right) \tag{3.10}$$

$$P[\theta] = \left|\sum_{n=0}^{N-1} a_\theta[n] x[n]\right|^2 \tag{3.11}$$

Algorithm 2 shows the Bartlett DOA algorithm where $x[n]$ represents the input data (one vertical column in range/doppler), $a[n][k]$ represents the steering vector for a given angle θ, and $P[k]$ represents the output data (bearing).

Algorithm 2: Bartlett Beamforming

```
1:  for every range/bearing combination{
2:      for every angle $k$ to be calculated{
3:          $temp = 0$
4:          for all samples $n${
5:              $temp = temp + a[n][k] \cdot x[n]$
6:          }
7:          $P[k] = |temp|^2$
8:      }
9:  }
10: end of Bartlett DOA function
```

3.3.3 Oversampling Factors

Oversampling the DFT, as done via the CZT and the Bartlett DOA, brings distinct signal processing advantages. However, due to the fixed and known bandwidth of the radar waveform, some oversampling factors (OSF) will yield better results than others. The Straddling Loss (in dB) and resolution error (when compared to the analog waveform) is simulated by sliding a point target across two adjacent bins, and then averaged for various OSFs. Figure 3-12 shows the behaviour of the Straddling Loss as a function of OSF. It is clear from the figure that the Straddling Loss is reduced when the waveform is oversampled.

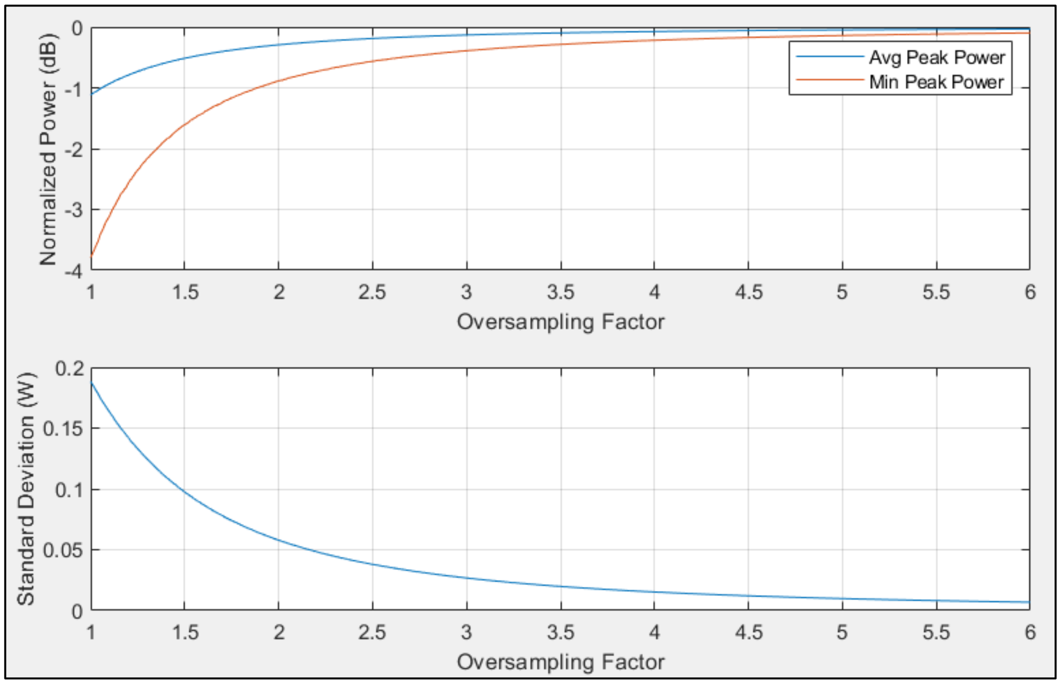

Figure 3-12. Straddling Loss vs Oversampling Factor

Although the average Straddling Loss is less than 1 dB for all OSFs above 1x, there is always the possibility that a target will be located exactly between 2 DFT bins (worst case scenario). The orange line, in the upper plot of Figure 3-12 represents the worst-case Straddling Loss at a given OSF. Choosing an OSF greater than 2.x will prevent the worst-case Straddling Loss from exceeding 1 dB.

Figure 3-13 shows the relationship between the average resolution measurement error (shown as a percentage) and the OSF. Interestingly, the relationship is not linear and exhibits peaks and nulls across the various values of OSFs. The nulls, or zero error regions OSF_0, are located at:

$$OSF_0 \approx N \times \frac{2}{R_s} \approx N \times 2.2576 \tag{3.12}$$

where:

$N = 1, 2, 3, \ldots$
$R_s = 0.88589\ldots = 3$ dB width of $\operatorname{sinc}^2(x)$

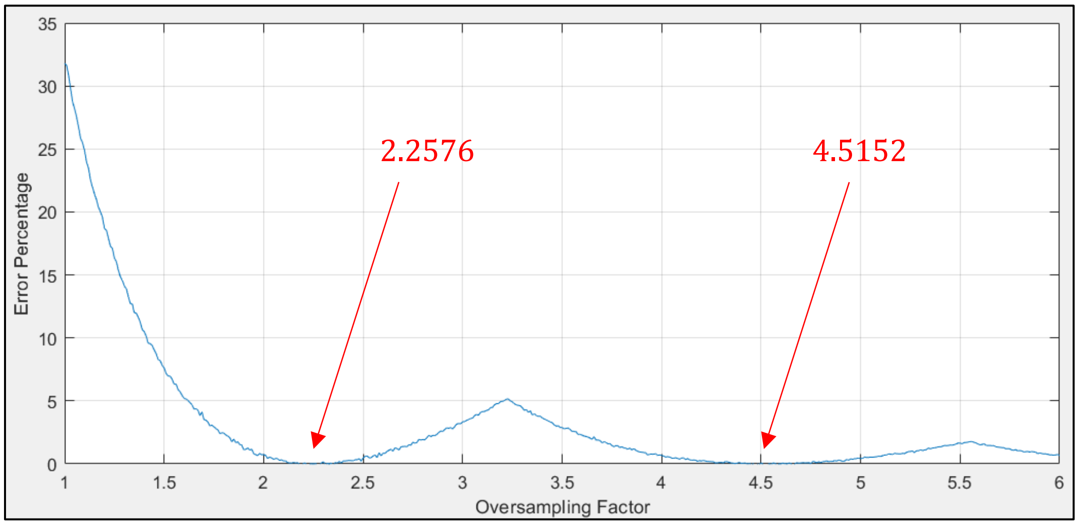

Figure 3-13. Resolution Error vs Oversampling Factor

Using Figures 3-12 and 3-13 as guidance, a better choice can be made when selecting the CZT and Bartlett parameters to enhance the resolution and reduce the Straddling Losses. For the CZT, computing the ranges between 0 m and 170 m will yield an OSF of 4.514x. For the Bartlett DOA, evaluating 144 bearing will yield an OSF of 2.250x.

3.3.4 Multicore CPU Parallelization

As discussed in Chapter 2, parallel tasks can be performed by multiple CPU cores simultaneously to reduce the execution time of an algorithm. The FFTW library is compatible with OpenMP, and FFT plans can be made using multiple threads. The following are additional functions which need to be called when wanting to execute multithreaded FFTs:

Function	Notes
fftw_init_threads()	Performs system initialization to use multiple threads
fftw_plan_with_nthreads(nthreads)	The input of this function will dictate how many threads will be used when creating FFT plans
fftw_cleanup_threads()	Clears up memory, resources, and thread related data allocated by FFTW

Table 3-4. Multithreaded FFTW Functions. Notes from [45]

Parallelization of the FFT is therefore as trivial as identifying how many threads the system should use when planning the FFT. Since a large component of the CZT is the execution of the FFT and the IFFT, executing them in parallel on the CPU will reduce the total execution time. Additionally, the CZT has three multiplication steps where each element of the large arrays gets modified. Instead of performing these multiplications sequentially, OpenMP enables the system to split the task across the multiple cores. To enable these parallel tasks, additional commands (#pragma omp parallel for) are added to lines 1, 5, and 9. With these commands, the entirety of Algorithm 1 is parallelized.

A similar approach is used for the Bartlett DOA. Spreading the work of the **for** loop across multiple cores reduces the total execution time of the algorithm. In Algorithm 2, the additional command (#pragma omp parallel for) is simply added to line 1.

Finally, the Cube Compression function is also parallelized using OpenMP. Note that the dimension of the processed cube is now $1024 \times 256 \times 144$ since the existing range and bearing FFTs have been replaced by the CZT and the Bartlett DOA, respectively. Algorithm 3 shows the Cube Compression function, where RD is the range-doppler matrix and RB is the range-bearing matrix.

When using the CZT and the Bartlett DOA, the number of pixels of the range-doppler and range-bearing displays is increased to 262,144 and 147,456, respectively, for a total of 409,600 compressions, which is an increase of approximately 2.5 times the number of pixels and which adds to the execution time of the Cube Compression function. Even when using OpenMP with 56 threads, each thread still needs to compute the output of over 7300 pixels. Using (2.17), the acceleration achievable by parallelizing the Cube Compression can be predicted. Even if the program is highly parallel (assuming 1% of the algorithm is sequential, for argument's sake), Amdahl's law predicts a speed up of only 36x when using 56 threads.

Algorithm 3: Cube Compression using OpenMP

```
1:  omp parallel for all sample indexes $\boldsymbol{S}${
2:      for all doppler indexes $\boldsymbol{D}${
3:          $temp = 0$
4:          for all angular bins $\boldsymbol{A}${
5:              $temp = temp + |data[S, D, A]|^2$
6:          }
7:          $RD[S, D] = temp$
8:      }
9:      for all angular indexes $\boldsymbol{A}${
10:         $temp = 0$
11:         for all doppler bins $\boldsymbol{D}${
12:             $temp = temp + |data[S, D, A]|^2$
13:         }
14:         $RB[S, A] = temp$
15:     }
16: }
17: end of Parallel Cube Compression function
```

Figure 3-14 shows a visualization of the Cube Compression process when done in parallel, where 3 threads are used to generate the range-doppler image.

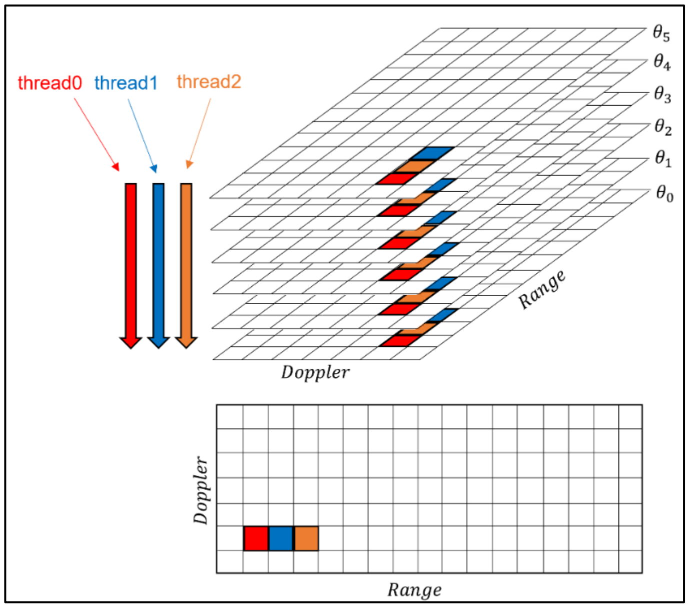

Figure 3-14. Multithreaded Cube Compression

3.3.5 GPU Parallelization

Similar to FFTW, CUDA has a FFT product in its computing toolbox called cuFFT which enables the efficient computation of DFTs on a GPU [46]. Just as with FFTW, cuFFT can compute the DFT for different dimensions (1D, 2D, 3D) and different precisions (i.e., float or double). Note that the baseline and proposed algorithms are computed with **double** precision. The cuFFT API is similar to FFTW where the user must create an FFT plan which can then be executed on the GPU. Prior to executing the transform, the user must first ensure that the data has been transferred onto the device. Additionally, just like the execution of any GPU function, the results must be transferred back onto the CPU for further use or display in the program. Table 3-5 lists useful cuFFT functions:

Function	Notes

cufftPlan1D( )
cufftPlan2D( )
cufftPlan3D( )
	Plans 1D/2D/3D Transforms respectively.

cufftPlanMany( )	Creates a plan which performs many DFTs across a large dataset.

cufftExecC2C( )
cufftExecZ2Z( )
	Executes a complex DFT for single/double precision, respectively. The arguments are pointers to the input and output arrays and the direction of the FFT (FFT/IFFT)

cufftDestroy( )	Clears a cuFFT plan from the program, along with associated memory and resources.

Table 3-5. Useful cuFFT Functions, reproduced from [46]

Some signal processing acceleration may be achieved by parallelizing tasks such as squaring or applying windowing weights to a large data set [36]. Similar techniques were used when implementing the CZT on the GPU. The transforms are performed using cuFFT, but each of the multiplication steps (i.e., multiplying by the chirp weight or performing the fast convolution) is done in parallel on the GPU where each 3D cube bin is mapped to a thread. Algorithm 4 illustrates the parallel CZT implementation on GPU where **Multiply_1** is a kernel which multiplies the input data and the chirp weights, **Multiply_2** is a kernel which performs the fast convolution, and **Multiply_3** is a kernel which multiplies the output of the fast convolution with the chirp weight and writes the final results in an output array.

Algorithm 4: Chirp Z Transform on GPU

```
1:   // Perform $y = x \cdot a \cdot W$ across entire cube
2:   Multiply_1 <<<grid_size, block_size>>>(y, x, a, W)
3:
4:   // Perform in place FFT in the range direction
5:   cufftExecZ2Z(plan_fft, y, y, CUFFT_FORWARD)
6:
7:   // Perform fast convolution
8:   Multiply_2<<<grid_size, block_size>>>(y, $W_{inv}$)
9:
10:  // Perform IFFT
11:  cufftExecZ2Z(plan_fft, y, y, CUFFT_INVERSE)
12:
13:  // Perform final multiplication with chirp weight, $Out = y \cdot W$
14:  Multiply_3<<<grid_size, block_size>>>(Out, y, W)
15:
16:  end of GPU CZT function
```

For the Bartlett DOA, the transformation of (2.15) to (3.11) enables the algorithm to be done in two simple steps on the GPU. Step 1 is to perform an element-by-element multiplication of the input samples $x[n]$ with the steering vector $a[n]$ across the cube for all range-doppler cells. Step 2 is to perform the sum of $x[n] \cdot a[n]$ for each element $n = 0 \rightarrow 63$. This two-step process is then done for all angles of interest, as shown in Algorithm 5. Figure 3-15 shows a visualization of the multiply/sum implementation of the Bartlett DOA for one angle. The process is then repeated sequentially for all required angles. Note that the results are not squared, as in (3.11), because the squaring step will be performed by the Cube Compression function.

Algorithm 5: Bartlett DOA on GPU

```
1:    **for** *all angles of interest* $\boldsymbol{l}${
2:        *Bartlett_Multiply<<<grid_size, block_size>>>(y, x, a, l)*
3:        *Bartlett_Add<<<grid_size, block_size>>>(Out, y, l)*
4:    }
5:    *end of GPU Bartlett DOA function*
```

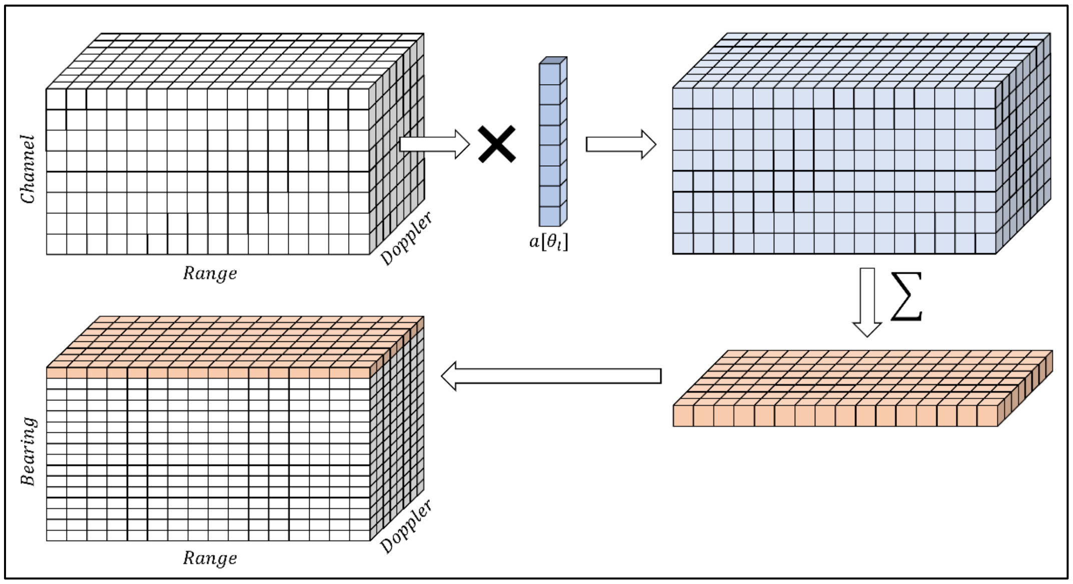

Figure 3-15. GPU Implementation of Bartlett DOA

Finally, the Cube Compression is performed on the GPU by mapping each pixel that needs to be computed (range-doppler and range-bearing displays) to a thread. When the kernel is launched, each thread has a unique ID which means that each pixel can be calculated simultaneously in parallel. Assuming there are A pixels in the range-doppler display and B pixels in the range-bearing display, a total of $A + B$ threads need to be launched. Within the kernel, an initial statement will verify if the thread ID is larger or smaller than A in order to computer either a range-doppler or a range-bearing pixel. Since the data cube has range, doppler, and bearing components, the next step of the kernel is to identify these indices so the power sum can be performed, and so the output can be stored in the correct location. Algorithm 6 represents the Cube Compression kernel implementation. Figure 3-16 illustrates the computation of the radar frame using the CZT and the Bartlett DOA on the GPU.

Algorithm 6: Cube Compression on GPU

```
1:  $index = blockIdx.x * blockDim.x + threadIdx.x$
2:
3:  // Do Range-Doppler Calculation
4:  **if** $(index < A)\{$
5:      *find range and doppler bin indexes* **r** *and* **d**
6:      $temp = 0$
7:      **for** *all bearings* **b**{
8:          $temp = temp + |cube[r,d,b]|^2$
9:      }
10:     $out_{doppler}[r,d] = temp$
11: }
12: **else if** $(index \geq A)\{$
13:     *find range and bearing bin indexes* **r** *and* **b**
14:     $temp = 0$
15:     **for** *all doppler* **d**{
16:         $temp = temp + |cube[r,d,b]|^2$
17:     }
18:     $out_{bearing}[r,b] = temp$
19: }
20:
21: *end of Cube Compression Kernel*
```

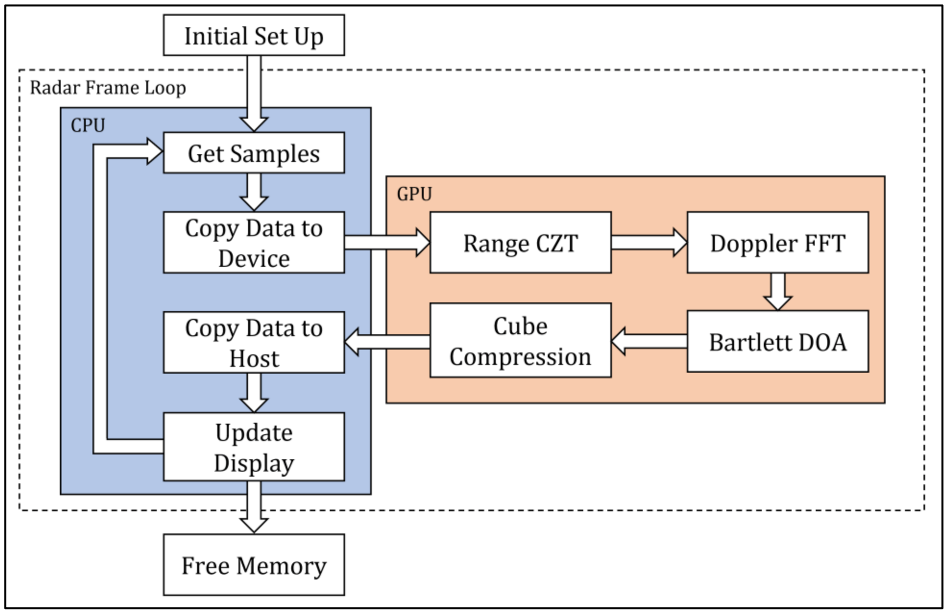

Figure 3-16. Parallel Implementation of Enhanced Resolution Radar Signal Processing

3.4 Short Range Correction

All beamforming methods discussed in Chapter 2 assume that the target is far from the radar and the echoes can be represented by plane waves. When this is the case, the phase shift experienced by subsequent antenna elements can be adequately approximated by (2.13). However, when the target is near the array, the plane wave approximation does not hold [47]. This happens at a range r of:

$$r < \frac{2D^2}{\lambda} \tag{3.13}$$

where:

D = largest dimension of the antenna array (m)
λ = wavelength (m)

3.4.1 Short Range Problem Definition

Although [47] provides a mitigation method for **sonar** systems, the problem of near targets is applicable for antenna arrays in a **radar** environment. In [7], the radar's transmit array measures 1.92 m and its waveform has a wavelength of 6 cm. Using (3.13), the far field of the antenna array is calculated to be 122.88 m. Unfortunately, due to hardware limitations, target ranges were limited to be within approximately 20 m (sometimes as near as 8 m). Therefore, distortion of the beam pattern is expected.

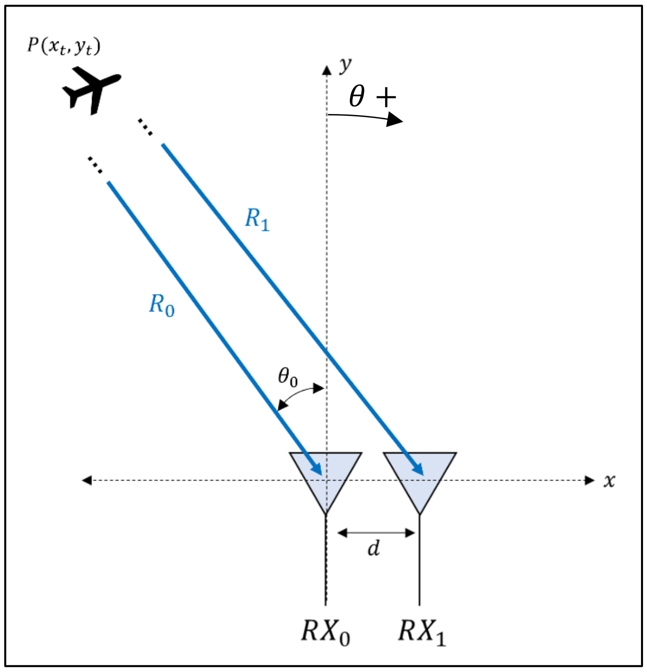

Figure 3-17. Receive Array Geometry, Far Target

Figure 3-17 shows a target at some range R_0 and bearing θ_0 from an antenna element RX_0. Using trigonometry, the range between the target and antenna element RX_1 can be found:

$$R_1 = \sqrt{(x_t - d) + y_t^2} = \sqrt{x_t^2 + y_t^2 - 2x_t d + d^2} \tag{3.14}$$

$$R_1 = R_0 \sqrt{1 - \frac{2x_t d}{R_0^2} + \frac{d^2}{R_0^2}} \tag{3.15}$$

where:

$x_t = R_0 \sin(\theta_0)$
$y_t = R_0 \cos(\theta_0)$
$d =$ element spacing (m)

Equation (3.15) has a familiar form containing $\sqrt{1+k}$. So long as $k \ll 1$, the radical can be approximated by a first order Taylor Series:

$$\sqrt{1+k} \approx 1 + \frac{k}{2} + HOT \tag{3.16}$$

where HOT represents the higher order terms of the sequence, which are usually negligible so long as the $k \ll 1$ criteria is respected. Since the target range R_0 in Figure 3-17 is very large, the approximation (3.16) can be used and (3.15) is reduced to:

$$R_1 = R_0 - d\sin(\theta) + \frac{d^2}{2R_0} \tag{3.17}$$

As R_0 increases, $d^2/(2R_0)$ tends towards zero. Therefore, the range difference between two subsequent antennas $(R_1 - R_0)$ is approximately $-d\sin(\theta)$, as expected by the paraxial approximation. However, when the target is near the array and the $k \ll 1$ condition no longer applies, and more terms of the Taylor Series would be required to adequately approximate $\sqrt{1+k}$.

If there is a target located at range of 1 km and a bearing of 20 degrees, the range difference between the first and the last virtual element of the MIMO radar would be:

$$\Delta R = (N-1) \times d \sin(\theta) \tag{3.18}$$

where:

ΔR = difference between RF path length (m)
N = number of virtual antennas
d = spacing between Rx Array elements (m)
θ = target bearing (degrees)

Using the radar in [7] with $d = 0.03\ m$ and $N = 64$, the expected range difference when using (3.18) is $\Delta R = 0.6464$ m. When calculating the true ranges to each element using (3.14), the range difference is $\Delta R = 0.6480$ m. When considering the wavelength of 0.06 m, the phase error across the entire virtual array is $\sim 9.6°$. This low phase error confirms the adequacy of the plane wave approximation method for targets which are in the far field of the array. However, if the example is repeated for a target range of 25 m (well within the far field boundary), the phase error across the virtual array increases to $\sim 186.8°$, which reduces the quality of the beam pattern of the DOA estimators. This problem is visualized in Figure 3-18, where the plane wave approximation is compared to the true ranges of the RF paths.

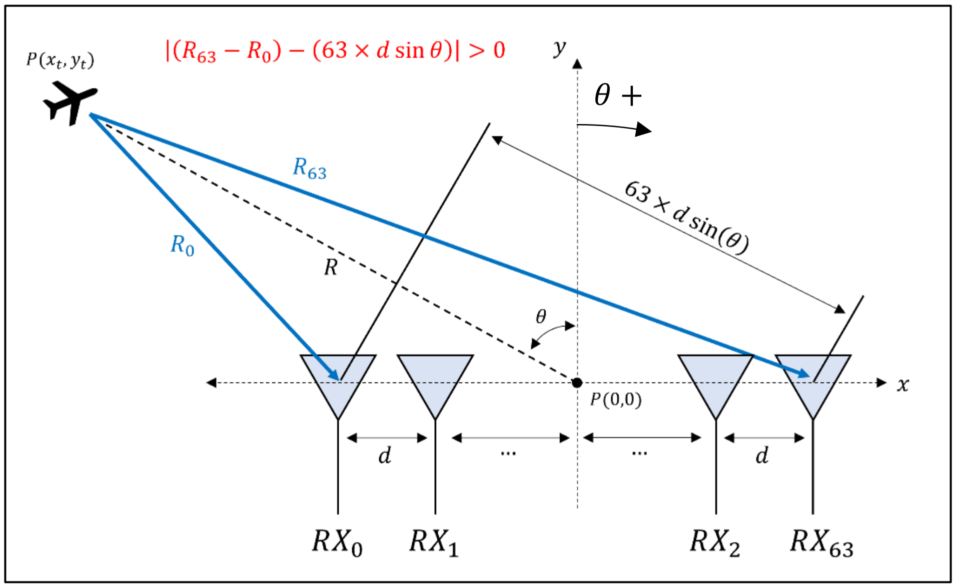

Figure 3-18. Short Range Target Phase Error

3.4.2 Short Range Correction for ULA

One of the issues with compensating for near targets is that the phase error is function of both the **range** and **bearing** of the target. Thankfully, using the enhanced resolution methods discussed in Section 3.3, both the range and the angle of a test cell is known. After range and doppler processing, the Bartlett beamforming method evaluates the power of the signal when "steered" to a given angle. Since this is done for every range cell, the samples across the MIMO virtual element can be directly compensated for. Using (3.15) and (3.17) the error between the true range and the plane wave approximation can be expressed as:

$$\Delta R_n = \sqrt{R^2 - 2d_n R\sin(\theta) + d^2} - (R - d_n\sin(\theta)) \qquad (3.19)$$

where:

ΔR_n = range error (m) at the n^{th} array element
R = range between target and center of the array (m)
θ = bearing to target, measured from the center of the array (degree)
d_n = distance (m) between center of the array and the n^{th} array element

Finally, since the range error will correspond to a phase error, a complex correction coefficient is applied to the discrete input samples for the given range and look angle, which effectively *flattens out* the wavefront as a plane wave.

$$x_c[n] = x[n]\exp(j\beta \times \Delta R[n]) \qquad (3.20)$$

where:

x_c = corrected sample (complex voltage)
x = input signal (complex voltage)
$\beta = 2\pi/\lambda$ = phase constant (rad/m)
λ = wavelength (m)
ΔR = range error (m) calculated from (3.19)
n = virtual array element index

Algorithm 9 illustrates how the phase error is calculated and applied when performing the Bartlett DOA for a given range-doppler bin.

Algorithm 9

```
1:  $N_{rx}$ = number of Rx elements
2:  $\beta$ = phase constant (rad/m)
3:  $R$ = range value of current range bin (m)
4:  $pos_{tx}[\ ]$ = array containing the positions (m) of the transmit elements
5:  $pos_{rx}[\ ]$ = array containing the positions (m) of the receive elements
6:
7:  for all bearings $\boldsymbol{b}${
8:      for all transmit elements $\boldsymbol{i}${
9:          for all receive elements $\boldsymbol{j}${
10:             $d_{tx} = pos_{tx}[\boldsymbol{i}]$
11:             $d_{rx} = pos_{rx}[\boldsymbol{j}]$
12:             $x_t = R * \sin(\boldsymbol{b})$
13:
14:             $L_{tx} = \sqrt{R^2 - 2d_{tx}x_t + d_{tx}^2}$
15:             $L_{rx} = \sqrt{R^2 - 2d_{rx}x^t + d_{rx}^2}$
16:             $R_{exp} = 2R - \sin(\boldsymbol{b})\,(d_{tx} + d_{rx})$
17:             $\Delta R = L_{tx} + L_{rx} - R_{exp}$
18:
19:             $bin = \boldsymbol{j} + \boldsymbol{i} * N_{rx}$
20:             $x_c[bin] = x[bin] * \exp(j\beta * \Delta R)$
21:         }
22:     }
23:     Do Bartlett DOA as per Algorithm 2 with corrected samples $x_c$
24: }
25: end of Corrected DOA Algorithm
```

As seen in Algorithm 9, the errors are calculated for the transmit array **and** the receive array since the total time of flight (TOF) is due to 2-way propagation as shown in (3.1). In order to avoid computing ΔR redundantly every radar frame, a **correction array** is generated during the initialization stage of the program which contains the complex coefficients for all ranges and angles, and are then multiplied with the input samples.

Using a pre-computed correction array translates well when performing the Bartlett DOA. Algorithm 2 is unchanged with the exception that $a[n] \cdot x[n]$ becomes $a[n] \cdot x[n] \cdot c[n]$ when calculating (3.11), where $\boldsymbol{x}$ is the input sample array, $\boldsymbol{a}$ is the steering vector, and $\boldsymbol{c}$ is the correction array for the given range and steering angle.

Figures 3-19 and 3-20 show the beamforming output of the FFT and the Bartlett DOA at different ranges. As the point target gets closer to the array, the beam pattern gets distorted, wider, and exhibits lower power.

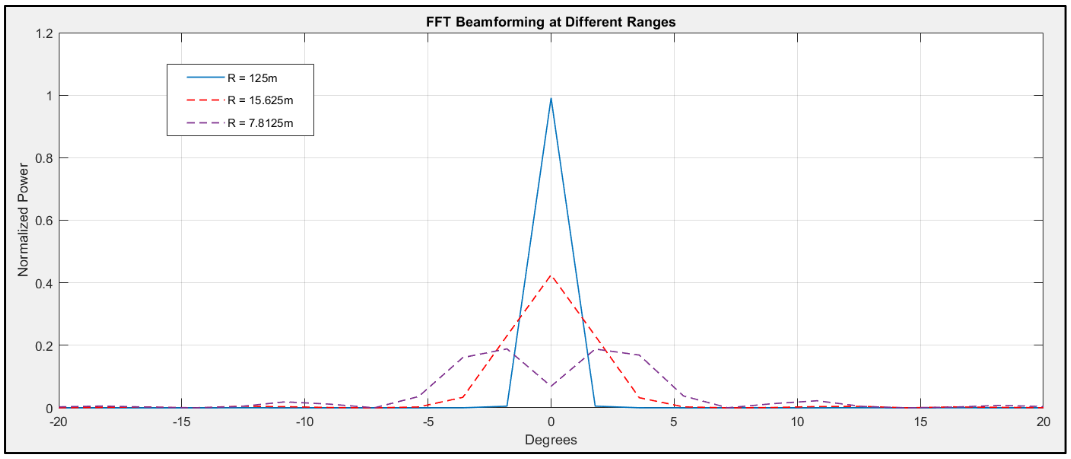

Figure 3-19. FFT (64-point) Beamforming Output at Different Ranges

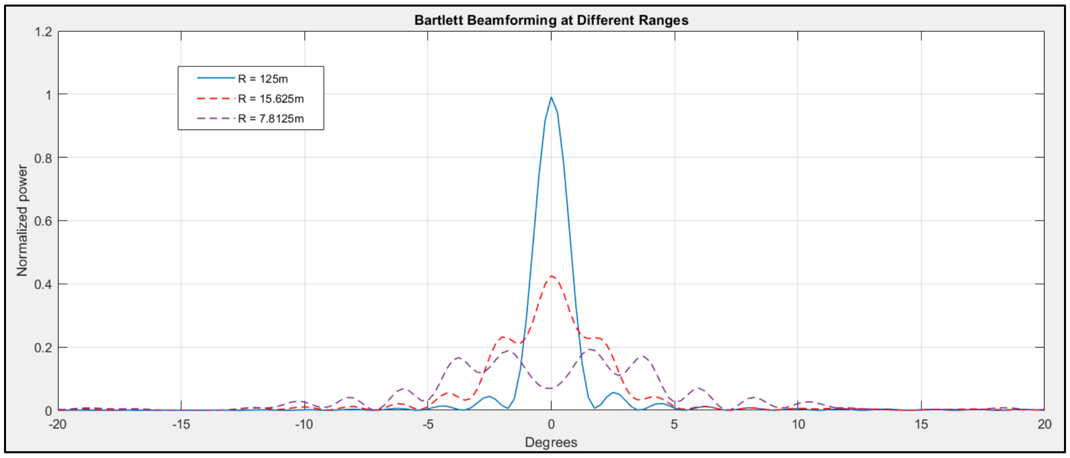

Figure 3-20. Bartlett DOA (720-point) Output at Different Ranges

Figure 3-21 shows a point target located at 8 meters from the array, with a bearing of 0 degrees. The leftmost image shows the distorted beam pattern of the near target. The rightmost image shows the range-bearing plot when using Algorithm 9. Figure 3-21 demonstrates that correcting the phases of the virtual channels enables the use of the Bartlett DOA at short ranges. Note that as discussed in Section 3.4.1, the far field region of the array begins at ~122.88 m.

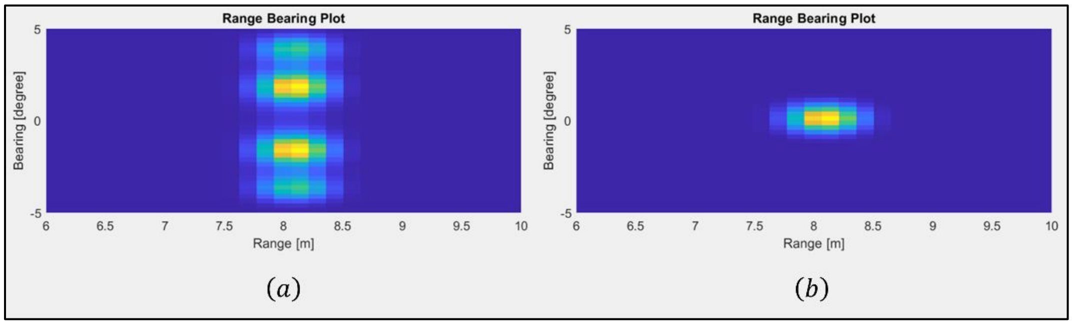

Figure 3-21. Short Range DOA using: (a) Original Samples (b) Phase Corrected Samples

As shown in Figures 3-19 to 3-21, short range angular measurements are degraded when using standard DOA algorithms as these rely on the plane wave approximation. This might explain the poor angular resolution measured in [7], since targets were place well within the far-field boundary. Applying the proposed short range correction method should enable sharp MIMO radar imaging of near targets.

3.5 Optimization of CPU Implementation

Memory access, when executing the Bartlett DOA from Algorithm 2, is not optimal when performed on the CPU. Algorithm 10 expands on Algorithm 2 and provides more information on its C++ implementation. In the algorithm, x represents the complex samples from the data cube, a represents the steering vectors, c represents the pre-calculated correction matrix (for short range targets), and X represents the output cube.

Algorithm 10: Bartlett Beamforming – Detailed

```
1:   for all range bins r{
2:       for all doppler bins d{
3:           // Do Bartlett Beamforming
4:           for all angles of interest k{
5:               temp = 0
6:               for all channels i{
7:                   temp += x[i][d][r] * a[k][i] * c[k][i][r]
8:               }
9:               X[k][d][r] = temp
10:          }
11:      }
12:  }
13:  end of Detailed Bartlett DOA function
```

As the innermost loop iterates (lines 6-10), data from the x, a, and c matrices are loaded from memory prior to executing the multiplication. Since the x matrix and the c matrix experience significant reuse throughout the loops, the data can be pre-loaded to smaller temporary matrices which only need to be updated when the required data changes (i.e., the next range bin). Algorithm 11 shows the optimized CPU implementation, with x_temp and c_temp as the temporary arrays. Loading the data from the matrices (lines 5 and 11) is done in a uncoalesced manner and is inefficient. However, the data is stored to the temporary matrices in a way that enables coalesced loads throughout the nested loops of the Bartlett DOA (lines 6-10). The coalesced loads take advantage of CPU caching, which reduces the total execution time of the radar frame.

Pre-loading the data matrices, as detailed in Algorithm 11, while using the radar parameters of Table 3-1 reduces the execution time of the radar frame by ~55% when done sequentially on the CPU, and by ~40% when done in parallel on the CPU.

Algorithm 11: Bartlett Beamforming – Optimized

```
1:   for all range bins r{
2:       // Pre-Load the Correction Matrix
3:       for all angles of interest k{
4:           for all channels i{
5:               c_temp[k][i] = c[k][i][r]
6:           }
7:       }
8:       for all doppler bins d{
9:           // Pre-Load Data Cube Matrix
10:          for all channels i{
11:              x_temp[i] = x[i][d][r]
12:          }
13:          // Do Bartlett Beamforming
14:          for all angles of interest k{
15:              temp = 0
16:              for all channels i{
17:                  temp += x_temp[i] * a[k][i] * c_temp[k][i]
18:              }
19:              X[k][d][r] = temp
20:          }
21:      }
22:  }
23:  end of Optimized Bartlett DOA function
```

4 Results

To verify the success of the proposed algorithm, both the acceleration of the signal processing and the resolution measurements must be quantified. Using the simulated environment described in Chapter 3, raw data cubes are generated and sent to the signal processing chain. Section 4.1 highlights the acceleration results, while Section 4.2 evaluates the resolution measurements of the proposed method.

Note that although super-resolution algorithms were considered (i.e., MUSIC, MVDR, etc.), the CZT and the Bartlett DOA still increases the resolution of the radar images (when compared to the baseline FFT) without having all the associated requirements discussed in Section 2.2. Furthermore, the computational cost associated with the super-resolution methods would have been must larger, making the CZT and the Bartlett DOA a good compromise.

The signal processing was performed on a Dell 7920 workstation which contains 2 Intel® Xeon Gold 5120 processors (14 Cores each), and an NVIDIA GeForce 2080 Ti GPU (4352 CUDA Cores). The base clock rates are 2200 MHz and 1350 MHz for the CPU and the GPU, respectively. Note that the Xeon Gold 5120 processors have Hyper-Threading Technology, which enables each core to run 2 threads at once [48]. Since the workstation has 2 CPUs with 14 cores each, 56 threads in total are available.

4.1 Signal Processing Acceleration

The execution times of different algorithms are measured by calling timing functions before and after they are called from the C++ program. Since the acceleration of all portions of the signal processing is of interest, the timers will measure the execution time of the range algorithm (FFT and CZT), the doppler algorithm (FFT), the bearing algorithm (FFT and Bartlett), as well as the cube compression.

Both FFTW and cuFFT can generate 3D FFT plans, which are executed faster than performing a range FFT, followed by a doppler FFT and a bearing FFT. For completeness, both methods will be timed and included in the results.

For the experiment, simulation runs will be performed using the baseline FFT method (range FFT → doppler FFT → bearing FFT → Cube Compression), the more efficient FFT method (3D FFT → Cube Compression), the proposed method (range CZT → doppler FFT → Bartlett DOA → Cube Compression), and the proposed method with the addition of Short Range Correction. All methods will be tested when run sequentially on the CPU, in parallel on the CPU (56 threads), and in parallel on the GPU. A total of 30 tests per method will be performed, where the average timing measurements are recorded for acceleration calculations. Note that when testing the GPU implementation, data transfer times between the host and the device must be measured and included in the results.

The acceleration A of a task or program is determined by comparing its sequential execution time t_{seq} with its parallel execution time t_{par}:

$$A = \frac{t_{seq}}{t_{par}} \tag{4.1}$$

Table 4-1 lists the average execution time and acceleration results for all methods. Each radar frame computation is performed on 1024 time samples, 256 chirps, and 64 virtual MIMO channels. As mentioned in Section 3.3.5, all samples are complex doubles.

	Process Name	**Execution Times (ms)**			**Acceleration**	
		Sequential CPU	Parallel CPU	GPU	Parallel CPU	GPU
FFT Method	3D FFT	599.82	133.16	7.55	4.5x	79.4x
	Cube Compression	2357.83	102.07	2.74	23.1x	860.5x
	Copy Host → Device			64.93		
	Copy Device → Host			0.84		
	Overall	**2957.65**	**235.24**	**76.05**	**12.6x**	**38.9x**
Baseline FFT Method	Range FFT	102.62	46.47	3.21	2.2x	32.0x
	Doppler FFT	340.27	77.45	4.45	4.4x	76.5x
	Bearing FFT	206.09	60.50	1.52	3.4x	135.6x
	Cube Compression	2322.66	90.97	2.59	25.5x	896.8x
	Copy Host → Device			64.23		
	Copy Device → Host			0.90		
	Overall	**2971.65**	**275.38**	**76.90**	**10.8x**	**38.6x**
Proposed Method	Range CZT	3920.41	646.12	17.98	6.1x	218.0x
	Doppler FFT	316.36	64.48	3.90	4.9x	81.1x
	Bartlett DOA	132472.14	4967.38	228.29	26.7x	580.3x
	Cube Compression	10281.97	349.00	7.15	29.5x	1438.0x
	Copy Host → Device			65.19		
	Copy Device → Host			1.14		
	Overall	**146990.89**	**6026.98**	**323.65**	**24.4x**	**454.2x**
Short Range Correction	Range CZT	3862.72	645.05	17.97	6.0x	215.0x
	Doppler FFT	412.54	80.16	3.83	5.1x	107.7x
	Bartlett DOA	184318.34	7060.80	229.22	26.1x	804.1x
	Cube Compression	10204.25	411.19	7.22	24.8x	1413.3x
	Copy Host → Device			63.87		
	Copy Device → Host			1.12		
	Overall	**198797.85**	**8197.20**	**323.22**	**24.3x**	**615.1x**

Table 4-1. Execution Time and Acceleration Results

As seen from Table 4-1, the multicore CPU and the GPU were able to take advantage of the parallelism within the algorithms. Even when using the baseline FFT method (1 3D FFT plan), reasonable acceleration was achieved: 12.6x on the multicore CPU and 38.9x on the GPU. In general, the execution of the FFTs on the CPU does not seem to benefit much from parallelization. Despite having 56 available threads, the maximum acceleration achieved from the CPU was only 4.5x when using a single 3D FFT plan. On the GPU, the FFTs are performed quickly, yielding an acceleration of 79.4x. The single 3D FFT plan is also observed to be slightly more efficient than 3 individual plans, which suggests that the FFTW and cuFFT planners are well optimized.

When executing the Proposed Method (with 144-point Bartlett), large accelerations are achieved for both the multicore CPU (24.4x) and the GPU (454.2x). The parallelism introduced by the Bartlett algorithm is evident, as this step in the signal processing chain yields speed ups of 26.7x for the CPU, and 580.3x for the GPU. However, only the GPU implementation could be considered practical as it can compute the entire radar frame in ~324 ms. Even with a respectable acceleration, the multicore CPU implementation still takes approximately 6 seconds per frame.

As shown in Section 3.4, targets well within the far-field boundary can properly be imaged by applying a phase correction to the samples, while performing the Bartlett DOA algorithm. Despite pre-calculating the phase correction matrix, the additional tasks of finding the proper indices, performing the memory load, and executing an extra double precision multiplication increases the computation time by ~40% when run on the CPU (sequentially of parallel). The GPU, however, only requires an additional 0.93 ms to perform the corrected Bartlett DOA Algorithm, resulting in the acceleration of 615.1x from Table 4-1. Although this result is important, it may only be pertinent to applications which require the imaging of short range targets. The acceleration achieved with the Proposed Method (not corrected) might be a better indicator of the performance of the GPU, when used for MIMO radar signal processing.

Although the results are not included in Table 4-1, a 289-point Bartlett version of the Proposed Method (without Short Range Correction) would take ~5 minutes to compute a frame when done sequentially on the CPU, ~ 11 seconds when done in parallel on the CPU (27.3x), and ~550 ms when done on the GPU (545.5x).

4.2 Resolution Measurements

To calculate the range and bearing resolutions of the algorithms, the width of the processed waveform must be measured. The spectrum power P_s of the chirp, after matched filtering, is represented by the square of the **sinc** function [12]:

$$P_s(x) = \left|\frac{\sin(x)}{x}\right|^2 \tag{4.2}$$

Figure 4-1 shows the normalized analog spectrum of a target located at 50 m, using the radar parameters from Table 3-1.

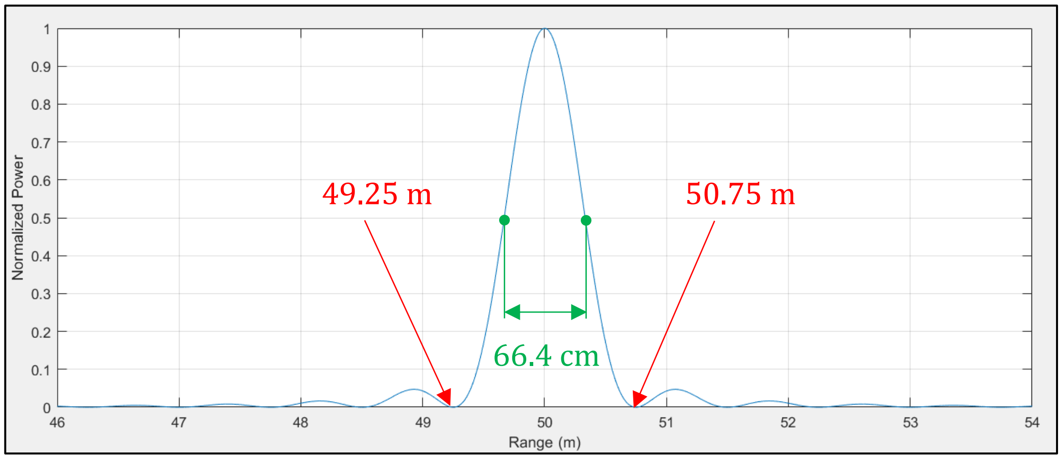

Figure 4-1. Analog Range Spectrum, Target at 50 m

The double sided -3 dB width ΔR_{3dB} of the spectrum commonly defines the range resolution of the radar and is related to the null-to-null width ΔR_{n-n} [12].

$$\Delta R_{n-n} = \frac{c}{B} \tag{4.3}$$

$$\Delta R_{3dB} \cong 0.88 \times \frac{c}{2B} \tag{4.4}$$

where:

c = speed of light (m/s)
B = bandwidth of the waveform (Hz)

From Figure 4-1, the distance between the 2 nulls is 1.5 m. Using (4.4), the range resolution of a 200 MHz waveform is calculated to be 66.4 cm.

Since the radar waveform is discretized, the resulting spectrum will be a "sampled" version of the **sinc** from Figure 4-1. Specifically, the baseline range FFT evaluates an output every 75 cm, while the proposed CZT does so every 16.60 cm. Figures 4-2 and 4-3 show the sampled power spectrum when using the baseline FFT and the proposed CZT, respectively.

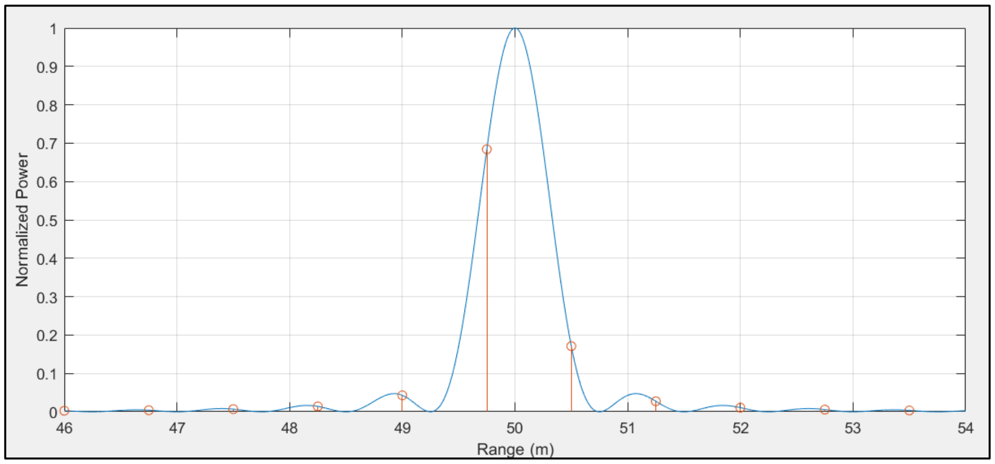

Figure 4-2. Sampled Power Spectrum, baseline FFT

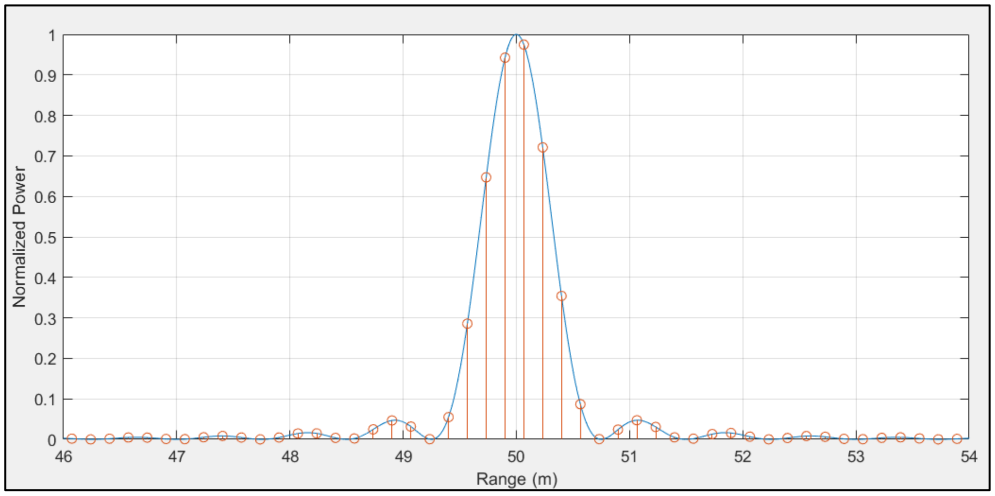

Figure 4-3. Sampled Power Spectrum, proposed CZT

Measuring the spectrum of a DFT comes with difficulties. Stradling Losses occur whenever the target position does not coincide with an FFT bin, which manifests itself by widening the main lobe of the spectrum and by reducing its peak power [12], [49]. Otherwise, the spectrum will have the sharpest response and highest SNR when the target position coincides with an FFT bin, as shown in Figure 4-4.

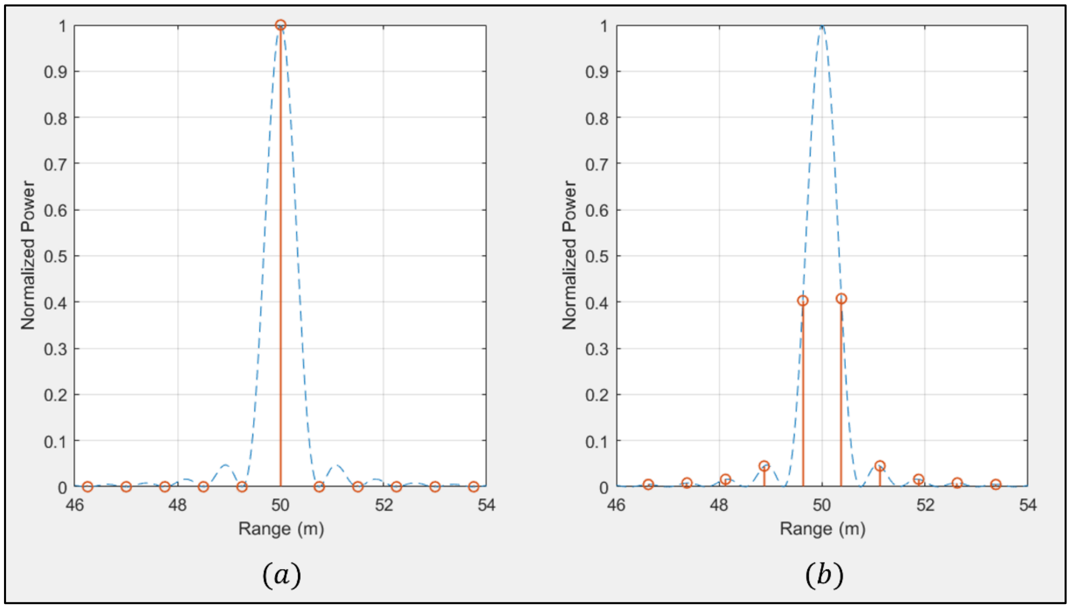

Figure 4-4. Stradling Loss: (a) Target Coinciding with FFT bin (b) Target in between FFT bins

The cases shown in Figure 4-4 (a) and (b) are the extremes, but since targets can be positioned anywhere, there is no single value for the signal strength and resolution of the FFT.

4.2.1 Range Resolution

To obtain a meaningful value for the radar's range resolution (-3 dB), many measurements must be taken and averaged. For the experiment, the target will begin at a range of 50.00 m (stationary target at broadside) where the Stradling Loss and spectral width is measured. The target will then move away from the radar by a small incremental range Δr $(bin\_size/40)$ where the power and width measurements are taken again. This is repeated until the target has traveled a distance equal to the size of the FFT bin.

Table 4-2 shows the results of the range resolution measurements. As discussed in Section 3.3.3, CZT parameters have been chosen as to provide an oversampling factor (OSF) of 4.515x, which has been identified to be one of the optimal OSFs when sampling a squared sinc function.

	Measurement Type	Max	Min	Average	Standard Deviation
Baseline FFT	Straddling Loss	3.87 dB	0.00 dB	1.26 dB	1.17 dB
	3 dB Resolution	149.90 cm	74.95 cm	88.06 cm	28.48 cm
Proposed CZT	Straddling Loss	0.18 dB	0.00 dB	0.06 dB	0.05 dB
	3 dB Resolution	66.41 cm	66.41 cm	66.41 cm	0.00 cm

Table 4-2. Range Resolution Measurements

By using an optimal OSF of 4.515x, the resolution of the Proposed CZT exactly matches the expected 3dB resolution of the waveforms. Additionally, the selected OSF ensures that there are **always** the same number of samples above the -3 dB threshold, bringing the Standard Deviation of the measured resolution to zero. It is found that the Proposed CZT enhances the baseline average resolution of 88.06 cm by ~24.58 precent. The Straddling Loss is also reduced when using the Proposed CZT, which increases the SNR.

4.2.2 Bearing Resolution

Similar to the procedure in Section 4.2.1, the -3 dB resolution and straddling losses are measured in the context of bearing resolution. Due to the issue of short range beamforming, the target is placed at a range of 140 meters, which is outside the far-field boundary of the antenna array. For the experiment, the target's bearing varies from 0° to the next bearing bin in increments of Δb $(bin\_size/80)$. To keep the target within the same range bin, its cartesian coordinates (x, y) are updated so that $\text{atan}(x/y)$ is equal to the desired angle and $\sqrt{x^2 + y^2}$ is equal to the constant range of 140 meters.

To compare the baseline FFT method against the 2 oversampling factors (OSF) identified in Section 3.3.3, the resolutions of both the 144-point and the 289-point Bartlett DOA will be measured. The 144-point DOA results in an OFS of 2.250x while the 289-point DOA results in an OFS of 4.515x, both of which have been identified as being optimal.

The number of MIMO virtual channels from Table 3-1 was chosen to yield a null-to-null beamwidth of 3.581° at broadside. Using (4.4), the theoretical -3 dB resolution is expected to be 1.586° (rounded to 1.59°) at broadside.

Table 4-3 shows the results of the bearing resolution measurements for the 3 methods.

	Measurement Type	Max	Min	Average	Standard Deviation
Baseline FFT	Straddling Loss	3.85 dB	0.00 dB	1.23dB	1.16 dB
	3 dB Resolution	3.58°	1.79°	2.10°	0.68°
Proposed 144-point Bartlett	Straddling Loss	0.71 dB	0.00 dB	0.23 dB	0.22 dB
	3 dB Resolution	1.59°	0.80°	1.58°	0.09°
Proposed 289-point Bartlett	Straddling Loss	0.18 dB	0.00 dB	0.06 dB	0.06 dB
	3 dB Resolution	1.59°	1.59°	1.59°	0.00°

Table 4-3. Bearing Resolution Measurements

Just as with the range resolution, the use of an optimal OFS greatly reduces the variance of the resolution measurement while providing the expected -3dB resolution (rounded to 2 decimal points). In terms of -3 dB resolution, there isn't much of an advantage in choosing the 289-point DOA over the 144-point DOA. Both methods enhance the baseline method resolution of 2.10° by ~24.48 percent.

The advantage of the 289-point DOA lies in the reduction of Straddling Losses. However, doubling the number of points only reduces the maximum Straddling Loss by 0.553 dB and the average Straddling Loss by 0.172 dB. This is consistent with the curves in Figure 3-12 from Section 3.3.3. Since the execution time of the Bartlett algorithm is linearly proportional to the number of evaluated points, one can reasonably conclude that the mild reduction in Straddling Loss is not worth doubling the execution time. The 144-point DOA adequately reduces the Straddling Losses of the baseline FFT method by 3.144 dB (maximum loss) and 0.995 dB (average).

4.2.3 Short Range Angular Resolution

As described in Section 3.4, beamforming algorithms suffer when targets are near the antenna array. A loss in amplitude was identified, along with the widening of the bearing response. To demonstrate the phenomenon of near target beamforming and to evaluate the performance of the proposed Short Range Correction algorithm, the procedure from Section 4.2.2 is repeated for a target located at 13.5 meters from the radar.

Table 4-4 shows the results of the bearing resolution measurements for the baseline FFT method, the Proposed Bartlett DOA method (144-point), and the Proposed Bartlett method with Short Range Correction.

	Measurement Type	Max	Min	Average	Standard Deviation
Baseline FFT	Combined Losses	6.38 dB	5.32 dB	5.78 dB	0.36 dB
	3 dB Resolution	7.16°	5.37°	5.60°	0.60°
144 - point Bartlett	Combined Losses	5.57 dB	5.22 dB	5.34 dB	0.11 dB
	3 dB Resolution	5.57°	4.78°	5.34°	0.36°
144 - point Bartlett + Short Range Correction	Combined Losses	0.71 dB	0.00 dB	0.23 dB	0.21 dB
	3 dB Resolution	1.59°	0.80°	1.58°	0.09°

Table 4-4. Bearing Resolution Measurement: Target at 13.5 m

As listed in Table 4-4, both the baseline FFT method and the 144-point Bartlett method suffer from considerable losses (~ 6 dB). These losses are a combination of beamforming distortion and Straddling Losses, which cannot be remedied by simply oversampling in bearing. Additionally, the average -3dB resolution is much larger than when the target is in the far-field. For the baseline FFT, the resolution is, on average, 2.67x wider than for far-field targets. For the 144-point Bartlett, the resolution is, on average, 3.37x wider than for far-field targets.

When the phase corrections are applied, as derived in Section 3.4.2, the average resolution is re-sharpened to its expected value of ~1.58°, and the losses are reduced to 0.71 dB (maximum) and 0.23 dB (average).

4.3 Result Summary

As shown in Sections 4.1 and 4.2, the proposed MIMO radar signal processing provides enhanced range and bearing resolution while enabling real-time operation. Both the range and bearing resolution measurements are reduced by ~24.5%, which now match the theoretical values. Additionally, when executing the algorithm (144-point DOA) on the GPU, an acceleration of 454.2x was achieved.

The Short Range Correction, proposed in Section 3.4.2, was also verified to correctly *flatten* the incoming wavefront and enable the correct operation of the DOA algorithm. Without correction, targets at 13.5 meters from the radar suffered significant losses in bearing resolution (~ 3x wider than expected) and peak power (~ 6 dB). When the correction was applied, the bearing resolution and peak power measurements matched those from targets in the far field. With the correction enabled, the CPU required ~ 40% more time to execute the Bartlett DOA. The GPU, however, only needed an additional ~ 1 ms, which explains the large acceleration of ~ 615x from Table 4-1.

The added complexity of the proposed algorithms is evident when comparing the execution times. On the CPU (sequential), performing the proposed algorithm with Short Range Correction takes ~67x more time than the baseline FFT method. On the GPU, however, performing the proposed algorithm only takes ~ 4.25x more time that the baseline, highlighting the suitability of GPUs for highly parallel applications.

5 Conclusion

To conclude, the contents of the previous chapters will be summarized, the principal contributions yielded by this thesis will be highlighted, the thesis hypothesis and its success will be discussed, and future work topics will be proposed.

5.1 Summary

In Chapter 1, the topic of MIMO radars and their computational demands were introduced. The problem statement, motivation, and thesis statement were also presented. Finally, the methodology and thesis organization were outlined.

In Chapter 2, relevant theory such as multiplexing techniques, radar fundamentals, signal processing techniques, and an introduction to MIMO radars was presented. Additionally, state of the art in MIMO radar prototypes and case studies on GPU accelerated radar signal processing were reviewed. Existing solutions to real-time operation were identified such as choosing fast algorithms (at the cost of resolution), reducing the quantity of data to process (i.e., the dimension of the radar or the number of range and bearing bins to compute), or by offloading the work on FPGA, DPS, or ASIC. A flexible GPU based solution, capable of quickly executing the signal processing without sacrificing resolution, was proposed for a 3D (range, bearing, and velocity) MIMO radar.

In Chapter 3, the simulation environment was introduced, which enables target echo generation for MIMO radar geometries. The simulated radar, modeled from [7], was presented along with its requirements, parameters, and hardware. The workstation used for processing the radar data was also described, including its installed hardware (CPU and GPU) and required software. Following this, the baseline and proposed algorithms, along with Short Range Correction, were developed. Finally, parallelization techniques for the baseline 3D FFT method and the proposed algorithm were described.

In Chapter 4, timing and acceleration measurements were taken for the different algorithms when run sequentially on the CPU, in parallel on the CPU, and in parallel on the GPU. Additionally, range and bearing resolution measurements were performed for all methods to quantify the improvement provided by the proposed method. Finally, bearing resolution was evaluated for short range targets with the aim of validating the Short Range Correction, as proposed in Section 3.4.2.

5.2 Contributions

The work achieved in this thesis yielded solutions to the limitations presented in Chapters 1 and 2. The key contributions of this work are listed below:

1. A simulated environment, which provides a virtual proving ground for MIMO radar signal processing, was developed. Multiple target echoes can be simulated within an environment where individual transmit/receive paths between the antennas and the targets are important. The simulation environment has configurable radar parameters, can accommodate different antenna array geometries, and all signal processing can be replaced, making the simulated environment a useful tool in future MIMO radar research.

2. Methods to accelerate the baseline 3D FFT and the Cube Compression algorithms were proposed. Despite the additional transfer time required to copy ~17 million complex doubles from the Host to the Device, the baseline FFT method saw significant acceleration when executed on the GPU (~39x) and can now compute the entire radar frame within 76 ms. Considering that the SDRs take approximately 308 ms to collect the chirp to form the CPI, the GPU accelerated method truly enables real-time operation of the radar by making the refresh rate acquisition-time-limited.

3. For a radar waveform with a known bandwidth, optimal oversampling factors (OSF) have been identified. These optimal OSFs guarantee that the sampled digital resolution is equal to the expected analog resoltuion, with little to no variance. In this work, OSFs of 2.257x and 4.515x were used to guide the parameter selection of the CZT and the Bartlett DOA. The use of these optimal OSFs have been validated in Chapter 4, which implies that very high oversampling factors (i.e., 10x or 20x) are not required to guarantee good resolution measurements and low straddling losses.

4. The CZT and the Bartlett DOA were proposed as replacements for the range and bearing FFTs, respectively. Using optimal OSFs, the measured resolutions were enhanced by ~24.5%. In addition to being configurable with regards to oversampling factors, the CZT and the Bartlett DOA can provide extra flexibility to the radar system. The computed ranges and bearings can be modified and even provide a zoom around a target of interest. Although the Bartlett DOA is much more computationally expensive than the FFT, the CZT maintains a complexity of $n\log_2(n)$.

5. A Short Range Correction technique was proposed in Section 3.4 which addresses the difficulty in performing beamforming on targets near the antenna array. By applying a phase correction to the samples, the spherical wavefront is effectively *flattened* into a plane wave which can be correctly beamformed. This phase correction is made possible by the MIMO mode, since individual transmit and receive paths can be separated and compensated for prior to the beamforming algorithm. The Short Range Correction, however, cannot be used with the FFT, since the angle information from the FOV is required during computation. The Short Range Correction technique has been validated in Chapter 4, and can be used for any beamforming technique which scans the FOV (i.e., Bartlett, MVDR, ESPRIT, IAA, etc.).

6. Methods to parallelize the CZT and the Bartlett DOA on the CPU and on the GPU were proposed. By executing each step of the proposed algorithms at once on the entire data cube, accelerations of ~24.4x for the CPU and ~454x for the GPU were achieved. When the short range correction is applied, the speed up is increased to ~615x.

The development of the proposed method with GPU acceleration, was presented at the 4th International Conference on Computing and Wireless Communication Systems (ICCWCS) [50].

5.3 Discussion

In Chapter 1, the following thesis statement was identified:

> *"Multicore CPU and GPU parallel processing techniques will be investigated to accelerate the high computational demands of MIMO radar signal processing. Additionally. The Chirp Z Transform and the Bartlett Beamformer will be evaluated to improve the range and angular resolutions. Resolution measurements will then be compared against the current version of RMC's MIMO radar."*

To verify the hypothesis that hardware acceleration would benefit MIMO radar processing, all algorithms described in Chapter 3 were developed to be executed either sequentially in the CPU, in parallel on the CPU, and in parallel on the GPU. In Chapter 4, all methods were tested and timed in order to calculate the achieved acceleration. Although any acceleration could be deemed a success, the total execution time of the radar frame will also be compared against the current version of RMC's MIMO radar. To be considered a success, the radar frame must be performed within 1.25 seconds [7].

To verify the hypothesis that replacing the range and bearing FFT by the CZT and the Bartlett DOA would enhance the resolution of radar, both the baseline FFT method and the proposed method have been programmed. In Chapter 4, resolution measurements have been conducted to quantify the improvement in both the range and bearing resolutions. Both far and near targets were used, to validate the Short Range Correction. Comparing the measured resolution against the current RMC MIMO radar is not meaningful, as the resolution is not well defined in [7]. Therefore, any enhancement in range or bearing resolution when using the proposed method is considered a success.

In the case of processing speed, acceleration enabled execution times well within 1.25 seconds. When using the baseline 3D FFT method, both multicore CPU and GPU acceleration met the requirement, with processing times of ~235 ms and ~76 ms, respectively. However, when using the proposed method (CZT and Bartlett DOA), only the GPU was able to compute the radar frame within the requirement, with an execution time of ~324 ms. If a signal processing time of 1.25 seconds is considered acceptable, the proposed algorithm on GPU still allows for another ~926 ms of computation. Therefore, this solution enables more radar processing to be performed (i.e., detection algorithms, tracking, etc.) within the allotted time.

In the case of resolution, the measurements from Chapter 4 show enhancements of ~24.5% in both range and bearing. With the CZT and Bartlett DOA configured to yield optimal OSFs (2.257x and 4.515x), the double-sided half-power resolutions of the radar now match the expected values exactly. For short range targets, the advantage of using the Bartlett DOA with Short Range Correction is apparent. At 13.5 meters, the average resolution of the baseline 3D FFT is ~5.6° whereas the proposed method yields the expected resolution of ~1.58°. This represents an enhancement in bearing resolution of approximately 3.5x. The proposed algorithms therefore provide enhanced resolutions at all ranges.

Given the results listed above, the thesis statement has been validated and it can be concluded that the proposed algorithm enhances the resolution of the MIMO radar, and that GPU acceleration enables faster refresh rates.

5.4 Future Work

Before concluding this thesis, 3 research opportunities are proposed for future work.

Firstly, the implementation of super-resolution beamforming techniques on the MIMO radar could be investigated. It has been shown in this thesis that the GPU can quickly perform the proposed algorithm, meaning that it might also be able to perform more complex algorithms in a reasonable amount of time. As discussed in Section 2.2, there are additional difficulties in implementing super-resolution techniques due to multiple correlated signals and clutter. Additionally, subspace-based algorithms (such as MUSIC and ESPRIT) require that the quantity of targets be known a priori, which is generally not the case in a surveillance radar. However, all of these methods (including MVDR and IAA) depend on complex linear algebra, such as the inversion of matrices. Researching the effectiveness of the GPU in the execution of these tasks would be of interest.

Secondly, orthogonal waveforms should be studied. This thesis focuses on the signal processing after demultiplexing and does not consider the effects of non-orthogonality of the chosen multiplexing technique (TDM, CDM, FDM). For a given waveform, target illumination and orthogonality could be evaluated. Additionally, the performance of the selected waveform could be validated in clutter and against moving targets. Research in this field could potentially yield a novel multiplexing scheme optimized for MIMO radar.

Finally, GPU acceleration of an Electronic Warfare system should be researched. As an example, extracting waveform information in an Electronic Support system can be computationally demanding. Whether the system is performing pulse sorting, frequency and time estimations, DOA estimations, or executing the Wigner-Ville and Radon Transforms, there could be parallelism exploitable by the GPU. Since there is value in quickly identifying signals in EW systems, accelerating the signal processing on a GPU should be investigated.

Bibliography

[1] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, “Massive MIMO for next generation wireless systems,” *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 186–195, Feb. 2014, doi: 10.1109/MCOM.2014.6736761.

[2] K. Li, B. Yin, M. Wu, J. R. Cavallaro, and C. Studer, “Accelerating massive MIMO uplink detection on GPU for SDR systems,” in *2015 IEEE Dallas Circuits and Systems Conference (DCAS)*, Oct. 2015, pp. 1–4. doi: 10.1109/DCAS.2015.7356600.

[3] J. Bergin and J. R. Guerci, *MIMO Radar Theory and Application*. Boston, MA, USA: Artech House, 2018.

[4] J. Li and P. Stoica, “MIMO Radar with Colocated Antennas,” *IEEE Signal Process. Mag.*, vol. 24, no. 5, pp. 106–114, Sep. 2007, doi: 10.1109/MSP.2007.904812.

[5] A. M. Haimovich, R. S. Blum, and L. J. Cimini, “MIMO Radar with Widely Separated Antennas,” *IEEE Signal Process. Mag.*, vol. 25, no. 1, pp. 116–129, 2008, doi: 10.1109/MSP.2008.4408448.

[6] S. Sun, A. P. Petropulu, and H. V. Poor, “MIMO Radar for Advanced Driver-Assistance Systems and Autonomous Driving: Advantages and Challenges,” *IEEE Signal Process. Mag.*, vol. 37, no. 4, pp. 98–117, Jul. 2020, doi: 10.1109/MSP.2020.2978507.

[7] R. T. Gilpin, “Real-Time Multiple Input Multiple Output (MIMO) Radar Using Sotware Defined Radio,” Royal Military College of Canada, Kingston Ontario, 2021.

[8] A. Ganis, “Architectures and Algorithms for the Signal Processing of Advanced MIMO Radar Systems - CORE,” Universita’ Degli Studi Di Udine, 2018. Accessed: Mar. 09, 2022. [Online]. Available: https://core.ac.uk/display/195748925?recSetID=

[9] Y. L. Sit, B. Nuss, S. Basak, M. Orzol, W. Wiesbeck, and T. Zwick, “Real-time 2D+velocity localization measurement of a simultaneous-transmit OFDM MIMO Radar using Software Defined Radios,” in *2016 European Radar Conference (EuRAD)*, London, Oct. 2016, pp. 21–24.

[10] T. Al-Nuaim, M. Alam, and A. Aldowesh, “Low-Cost Implementation of a Multiple-Input Multiple-Output Radar Prototype for Drone Detection,” in *2019 International Symposium ELMAR*, Sep. 2019, pp. 183–186. doi: 10.1109/ELMAR.2019.8918664.

[11] M. Skolnik, *Radar Handbook*, Third. New York, NY, USA: McGraw-Hill, 2008.

[12] M. Budge C. and S. German R., *Basic Radar Analysis*, Second. Norwood, MA, USA: Artech House, 2020.

[13] “IEEE Standard Letter Designations for Radar-Frequency Bands - Redline,” *IEEE Std 521-2019 Revis. IEEE Std 521-2002 - Redline*, pp. 1–22, Feb. 2020.

[14] F. Neri, *Introduction to Electronic Defense Systems*, Second. Edison, NJ, USA: SciTech Publishing, 2006.

[15] D. L. Adamy, *EW 102 A second Course in Electronic Warfare*. Norwood, MA, USA: Artech House, 2004.

[16] M. Skolnik, *Introduction to Radar Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2001.

[17] Y. Zhang, Y. Guo, and Z. Chen, "Range-Doppler domain signal processing for medium PRF ubiquitous radar," in *2016 CIE International Conference on Radar (RADAR)*, Oct. 2016, pp. 1–4. doi: 10.1109/RADAR.2016.8059428.

[18] A. Antoniou, *Digital Signal Processing Signals, Systems, and Filters*. New York, NY, USA: McGraw-Hill, 2006.

[19] P. Rajmic, Z. Prusa, and C. Wiesmeyr, "Computational cost of Chirp Z-transform and Generalized Goertzel algorithm," in *2014 22nd European Signal Processing Conference (EUSIPCO)*, Sep. 2014, pp. 1004–1008.

[20] C. A. Balanis, *Antenna Theory Analysis and Design*, Fourth. Hoboken, NJ, USA: Wiley, 2016.

[21] R. Sturdivant, C. Quan, and E. Chang, *Systems Engineering of Phased Arrays*. Norwood, MA, USA: Artech House, 2019.

[22] F. Meinl, M. Kunert, and H. Blume, "Hardware acceleration of Maximum-Likelihood angle estimation for automotive MIMO radars," in *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Oct. 2016, pp. 168–175. doi: 10.1109/DASIP.2016.7853815.

[23] "Smart-S Mk2," *Thales Group*. https://www.thalesgroup.com/en/worldwide/defence/smart-s-mk2-3d-medium-long-range-surveillance-radar (accessed Mar. 16, 2022).

[24] Y. Shao, G. Zheng, F. Liu, and F. Jiang, "A Coherent Weak Target DOA Estimation Method Based on Target Features," in *2021 OES China Ocean Acoustics (COA)*, Jul. 2021, pp. 5–8. doi: 10.1109/COA50123.2021.9519912.

[25] I. A. H. Adam and M. D. R. Islam, "Perfomance Study of Direction of Arrival (DOA) Estimation Algorithms for Linear Array Antenna," in *2009 International Conference on Signal Processing Systems*, May 2009, pp. 268–271. doi: 10.1109/ICSPS.2009.47.

[26] H. Liu, X. Wang, B. Jiu, J. Yan, M. Wu, and Z. Bao, "Wideband MIMO Radar Waveform Design for Multiple Target Imaging," *IEEE Sens. J.*, vol. 16, no. 23, pp. 8545–8556, Dec. 2016, doi: 10.1109/JSEN.2016.2604844.

[27] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors*, Third. Cambridge, MA, USA: Elsevier, 2017.

[28] T. Baji, "Evolution of the GPU Device widely used in AI and Massive Parallel Processing," in *2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM)*, Mar. 2018, pp. 7–9. doi: 10.1109/EDTM.2018.8421507.

[29] S. T. Ataullah and M. Siddique, "Optimisation Techniques for Multicore Architectures and Parallel Processing using OpenMP," in *2021 International Conference on Decision Aid Sciences and Application (DASA)*, Dec. 2021, pp. 187–191. doi: 10.1109/DASA53625.2021.9682392.

[30] K. Li, "GPU Accelerated Reconfigurable Detector and Precoder for Massive MIMO SDR Systems," Thesis, Rice University, 2015. Accessed: Mar. 18, 2022. [Online]. Available: https://scholarship.rice.edu/handle/1911/88088

[31] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming," in *2008 IEEE Hot Chips 20 Symposium (HCS)*, Aug. 2008, pp. 40–53. doi: 10.1109/HOTCHIPS.2008.7476525.

[32] G.-J. van den Braak, B. Mesman, and H. Corporaal, "Compile-time GPU memory access optimizations," in *Modeling and Simulation 2010 International Conference on Embedded Computer Systems: Architectures*, Jul. 2010, pp. 200–207. doi: 10.1109/ICSAMOS.2010.5642066.

[33] W. Wang, D. Liang, Z. Wang, H. Yu, and Q. Liu, "Design and Implementation of a FPGA and DSP Based MIMO Radar Imaging System," *Radioengineering*, vol. 24, no. 2, Art. no. 2, 2015.

[34] A. Figueroa, N. Joram, and F. Ellinger, "A fully modular, distributed FMCW MIMO radar system with a flexible baseband frequency," in *2021 IEEE Radar Conference (RadarConf21)*, May 2021, pp. 1–6. doi: 10.1109/RadarConf2147009.2021.9455174.

[35] L. Jun, L. Yang, and Q. Hu, "Airborne SAR motion compensation and imaging based on GPU architecture," in *IET International Radar Conference 2013*, Apr. 2013, pp. 1–6. doi: 10.1049/cp.2013.0146.

[36] R. S. Perdana, B. Sitohang, and A. B. Suksmono, "Radar Signal Processing in Parallel on GPU: Case Study Dual Polarization FMCW Weather Radar," in *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*, Jul. 2019, pp. 657–661. doi: 10.1109/ICEEI47359.2019.8988845.

[37] Y. Gao, H. Gao, and X. Zhang, "MIMO Radar Algorithm Parallel Implementation Based on TMS320C6678," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, Aug. 2014, pp. 231–236. doi: 10.1109/DASC.2014.49.

[38] F. Meinl, M. Kunert, and H. Blume, “Massively parallel signal processing challenges within a driver assistant prototype framework first case study results with a novel MIMO-radar,” in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, Jul. 2014, pp. 351–357. doi: 10.1109/SAMOS.2014.6893232.

[39] G. Liu *et al.*, “MIMO Radar Parallel Simulation System Based on CPU/GPU Architecture,” *Sensors,* vol. 22, no. 1, Art. no. 1, Jan. 2022, doi: 10.3390/s22010396.

[40] A. Hassanien, M. G. Amin, Y. D. Zhang, and F. Ahmad, “High-resolution single-snapshot DOA estimation in MIMO radar with colocated antennas,” in *2015 IEEE Radar Conference (RadarCon),* May 2015, pp. 1134–1138. doi: 10.1109/RADAR.2015.7131164.

[41] G. F. Rideout, “GPU Parallelization of the MVDR Beamforming ALgorithm for Multiple Input Multiple Output Radar,” Royal Military College of Canada (Canada), Kingston Ontario, 2019.

[42] “Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux - Ettus Knowledge Base.” https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux (accessed Mar. 24, 2022).

[43] “Pseudocolor plot - MATLAB pcolor.” https://www.mathworks.com/help/matlab/ref/pcolor.html (accessed Mar. 24, 2022).

[44] “Call MATLAB from C++ - MATLAB & Simulink.” https://www.mathworks.com/help/matlab/calling-matlab-engine-from-cpp-programs.html (accessed May 09, 2022).

[45] M. Frigo and S. G. Johnson, “FFTW Manual” https://www.fftw.org/fftw3.pdf (accessed Mar. 24, 2022).

[46] NVIDIA, “cuFFT Library User’s Guide” https://www.docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf (accessed Mar. 25, 2022).

[47] W. Yuan, T. Zhou, S. Jiajun, W. Du, B. Wei, and T. Wang, “Correction Method for Magnitude and Phase Variations in Acoustic Arrays Based on Focused Beamforming,” *IEEE Trans. Instrum. Meas.*, vol. 69, no. 9, pp. 6058–6069, Sep. 2020, doi: 10.1109/TIM.2020.2972657.

[48] “Product Specifications.” https://www.intel.com/content/www/us/en/products/sku/120474/intel-xeon-gold-5120-processor-19-25m-cache-2-20-ghz.html (accessed Jun. 06, 2022).

[49] J. P. Keradec and X. Margueron, “Improved Frequency Resolution DFT Eases Teaching FFT Analysis and Provides Better Amplitude Accuracy,” in *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*, May 2005, vol. 2, pp. 1149–1154. doi: 10.1109/IMTC.2005.1604324.

[50] E. Pitre, V. Roberge, J. Bray, and M. Hefnawi, “MIMO Radar Hardware Acceleration with Enhanced Resolution,” in *4th International Conference in Computing and Wireless Communication Systems (ICCWCS)*, Morocco, Jun. 2022, pp. 1–6.