Parallel Implementation of an Algorithm for High Resolution Range Profiling using a Stepped Frequency Radar

Soma Mukherjee

A dissertation submitted to the Department of Electrical Engineering, University of Cape Town, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Cape Town, January 2004

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author.....

Cape Town
January 2004

Abstract

This thesis describes the implementation and analysis of a frequency domain approach for reconstructing wide-bandwidth high resoultion range profiles using stepped-frequency waveforms. In order to minimize the overall processing time, a parallel algorithm was developed and tested in a homogeneous cluster of computers.

In Ultra-wideband synthetic aperture radar (SAR) systems, stepped-frequency waveforms are preferred for their ability to achieve high range resolution without putting burden to severe instantaneous bandwidth and sampling rate requirements. In a stepped-frequency system, the wide bandwidth is reconstructed by transmitting a group of narrow-bandwidth pulses, which are then combined to obtain the wide bandwidth. Several approaches to stepped-frequency processing exist, namely an IFFT method [9], a time domain method [12] and more recently a frequency domain method [23]. The IFFT method is unsuitable for SAR processing because it produces multiple "ghost" targets in high resolution profiles and the time domain method is computationally inefficient. The inefficiency of these two methods led to the development of the fast, computationally efficient frequency-domain method which does not have those previously mentioned drawbacks. In the frequency-domain method of reconstructing wide bandwith pulses, the narrow-band pulses are Fourier transformed and placed next to each other in the frequency domain with or without (splicing) any spectral overlap. In this method, however, a compensation filter is applied to the reconstructed spectrum to compensate for the amplitude ripples that generate paired echos in the impulse response.

In order to demonstrate the application of a stepped-frequency algorithm,

reconstructions were performed using real and artificially generated data sets. With real data, the splicing has been proved to be more successful in achieving high resolution range profiles as the spectral overlap can sometimes cause distortion in phase.

Real-time SAR processing is both computationally intensive and time consuming. The evolution of low cost, desktop machines at the commercial market together with the availability of 'Open Source' (OS) software has made the distributed parallel computing a viable solution for intensive SAR processing. A parallel version of stepped-frequency algorithm was created to decompose the task into multiple tasks by taking advantage of the inherent parallel nature of SAR data. In this model, the stepped-frequency processing algorithm adopts the master-worker programing paradigm where the worker process performs the same task on diferent sets of data. The parallel virtual machine (PVM) was used as a messaging 'middleware' of the parallel system. After having successfully implemented the parallel algorithm in a 5 node cluster some timing tests were performed. From the performance analysis it can be inferred that though the parallel system is highly scalable it suffers from high communication overhead. In order to reduce the communication and disk I/O opearation, the previously developed algorithm was modified and some timing analysis was done.

This thesis is dedicated to my baby boy.

Saurabh Roychoudhury

Acknowledgements

I wish to express my sincere thanks to my superviser Dr. A. J. Wilkinson for his guidance and for the financial assistance he offered me during the period of this dissertation.

I am grateful to Rakesh Kumar Singh for all his support and encouragement.

My sincere thanks to Thomas Bennett, Dr. Richard Lord and Dr. Daniel Masao for their help and advice.

I am also thankful to all my colleques for providing me a friendly and stimulating environment.

Thanks to my husband Dr. A. N. Roychoudhury for his assistance during the thesis write-up.

To my parents for all their encouragement.

Contents

List of Figures

List of Tables

List of Symbols

A	-	Amplitude of a complex signal [m]
B_{tx}	-	Transmitted RF bandwidth [Hz]
B_t	-	Total radar bandwidth [Hz]
c	-	Speed of light [m/s]
f_0	-	First centre frequency [Hz]
f_s	-	Sampling frequency
δf_i	-	Frequency shift associated with pulse i[Hz]
$\triangle f$	-	Frequency step size [Hz]
$H(f)$	-	Compression filter
H_{mf}	-	Matched filter
n	-	Number of pulses
P_{av}	-	Average transmitted power [W]
r_{max}	-	Maximum unambiguous range [m]
δR	-	Range resolution [m]
S	-	Speed up per node
T_p	-	Pulse length [s]
$v_{bb}(t)$	-	Baseband signal
$v_{rx}(t)$	-	Received waveform
v_{tx}	-	Transmitted waveform
$V'(f)$	-	Reconstructed spectrum
$Z(f)$	-	Target reflectivity spectrum
γ	-	Chirp rate of linear FM waveform [Hz]
$\zeta(t)$	-	Target reflectivity function
τ	-	Time delay [s]
ρ_r	-	Slant range resolution

Nomenclature

A/D—Analogue to Digital (converter)

Azimuth—Angle in a horizontal plane, relative to a fixed reference, usually north or the longitudinal reference axis of the aircraft or satellite.

Beamwidth—The angular width of a slice through the mainlobe of the radiation pattern of an antenna in the horizontal, vertical or other plane.

Buffer—A temporary storage area in memory.

Burst—Set of all frequencies required to produce a synthetic range profile.

Chirp—Linear frequency sweep during a pulse.

Cluster of Workstations—Workstations that are connected together via a hub or switch.

Corner reflector—A radar reflector that reflects nearly all of the radio frequency energy it intercepts back in the direction of the radar which is illuminating it.

CSIR—Council for Scientific and Industrial Research (South Africa).

FFT—Fast Fourier Transform.

FM—Frequency Modulation.

FLOPS—A measure of memory access performance, equal to the rate at which a machine can perform single-precision floating-point calculations.

Message Tag—An integer code bound to a message as it is sent.

MPP—Massively parallel processor.

IFFT—Inverse Fast Fourier Transform.

PRF—Pulse repetition frequency.

Profile—Contour of the target outline which is deduced from reflected signals in a radar system [6].

Range—The radial distance from a radar to a target.

Synthetic Aperture Radar (SAR)—A signal-processing technique for improving the azimuth resolution beyond the beamwidth of the physical antenna actually used in the radar system. This is done by synthesizing the equivalent of a very long sidelooking array antenna.

Swath—The area on earth covered by the antenna signal.

SNR—Signal to Noise Ratio.

SRP—Synthetic Range Profile.

Wideband—Radar systems that transmit and receive waveforms with instantaneous bandwidths between 1 percent and 25 percent of centre frequency [17].

Chapter 1

Introduction

1.1 Background to SAR

"Environmental monitoring, earth-resource mapping, and military systems require broad-area imaging at high resolution" [1]. *Synthetic aperture radar* (SAR) is a technique that provides high resolution radar imagery. Unlike optical and infrared imaging sensors which are inherently passive, meaning they rely on reflected or radiated energy, the active SAR sensor has day/night and all weather capability as it supplies its own illumination in the form of microwave energy. "Synthetic aperture radar technology has provided terrain structural information to geologists for mineral exploration, oil spill boundaries on water to environmentalists, sea state and ice hazard maps to navigators and reconnaissance and targeting information to military operations" [1, 2, 3].

The *South African Synthetic Aperture Radar* (SASAR) is an airborne SAR system developed by *Radar Remote Sensing Group* (RRSG) at UCT (Inggs 1996) and *Council for Scientific and Industrial Research* (CSIR - Defencetek) that operates in the VHF band at 141 MHz and has a 12 MHz bandwidth [4].

Figure 1.1 shows the geometry of the **strip-mapped** SAR. To image terrain, the radar is carried on an aircraft or spacecraft platform moving at uniform speed. The forward motion provides scanning along the track or

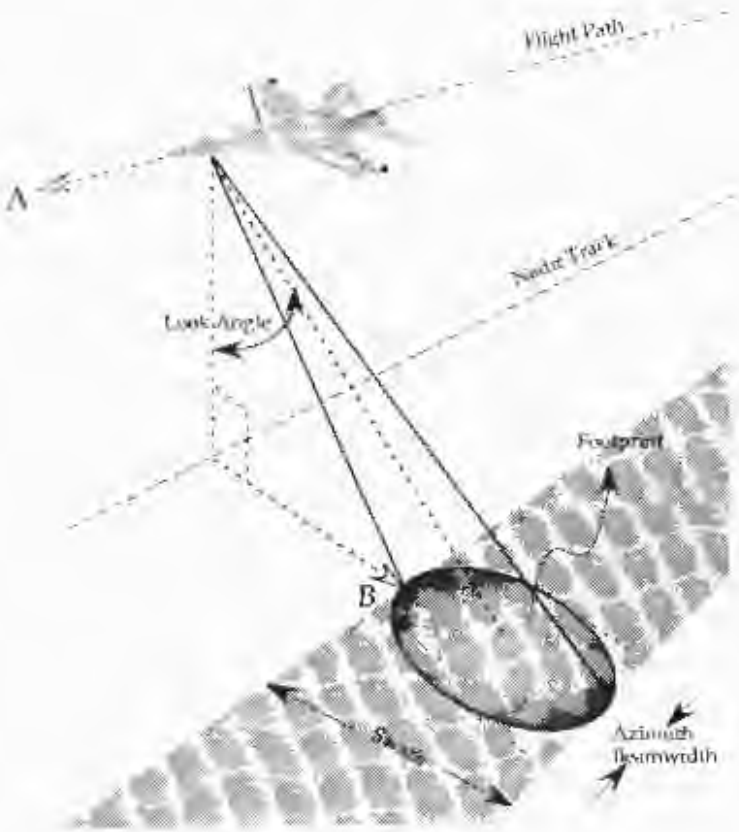

Figure 1.1: *Side-looking stripped-map SAR geometry (modified from [7] and [18])*

azimuth direction (A). The radar beam is directed to the side perpendicular to the track or **range** direction (B) and down to the surface. As the aircraft flies, the radar beam illuminates an area on the ground and a *swath* is mapped out on the ground by the antenna footprint. The ability of SAR to produce relatively fine azimuth resolution differentiates it from other radar systems.

The aircraft with a side-looking antenna as indicated in figure 1.1 moves along a straight flight path, emitting pulses at a regular *pulse repetition interval* (PRI), the backscattered signal corresponding to each pulse is recorded by the receiving antenna. The radar transmits pulses at the *pulse repetition frequency* (PRF), and for each pulse the backscatter return from the ground is sampled in range at the *analogue-to-digital* (A/D) sampling frequency, f_{ad}. Each A/D sample (typically 2048 for a pulse) represents backscatter radiation from adifferent time and therefore range [5, 6].

The return signals are stored as a two-dimensional array in digital memory with a size determined by the range bin size r_{bin} as given in equ. 1.1 and the azimuth bin size az_{bin} as given in equ.1.2 [7].

$$r_{bin} = \frac{c}{2f_{ad}\sin(\phi)} \tag{1.1}$$

$$az_{bin} = \frac{v}{PRF} \tag{1.2}$$

where c is the speed of light, ϕ is the look angle and v is the platform velocity. This two dimensional data set is eventually processed to form an image.

1.2 Background to Stepped frequency processing (SFP)

Range discrimination in radar is generally achieved by timing the returning scattered radiation. Fourier transform theory dictates that a fine timing interval (high range resolution) requires a high bandwidth from the radar system. This creates two potential problem areas. The first problem is that whether or not the microwave hardware will support a high bandwidth, given

that the radars are generally designed to operate at a given carrier frequency. The second is how to undertake the sampling and digitisation necessary for high resolution SAR processing [8].

In order to overcome the above two impediments we have a set of choices available:

1. Use a high instantaneous bandwidth and digitize at the full bandwidth.
2. Stretch waveform processing in which a single frequency-modulated (FM) pulse is used with large time-bandwidth product.
3. Use stepped frequencies

The use of a stretch waveform allows us to digitize at a rate much lower than the full system bandwidth which is desired to maintain a lower sample rate [9]. However, this technique, which requires long highly linear chirped radar pulses, does not take up a sufficiently large bandwidth B of the radio spectrum.

The stepped frequency processing is a preferred choice, mainly for its ability to produce high range resolution without imposing severe instantaneous bandwidth requirements on the radar system [10]. In the stepped-frequency approach, the full bandwidth is divided into number of sub-bands. The received signals from the pulses are processed to produce the equivalent of a full bandwidth pulse return.

Another important advantage of using stepped-frequency waveforms is the capability of skipping certain frequencies that would otherwise be corrupted by external sources such as broadcast FM-radio or TV [6].

There are three methods available to achieve the high resolution range profile or *synthetic range profiles* (SRPs) from a burst of narrowband pulses stepped in frequency. These methods are, namely

1. An IFFT method
2. Time domain method
3. Frequency domain method

The methods are briefly described in the sub-section 1.2.1, 1.2.2 and 1.2.3. The frequency domain method which is the basis of this thesis is described in detail in section 2.3.

1.2.1 IFFT method

To obtain a high resolution synthetic range profile from stepped-frequency data using the IFFT method entails the following steps [9, 11].

1. Transmit a burst of n pulses, in which each pulse is shifted in frequency by a fixed frequency step size, $\triangle f$.

2. For every transmitted pulse, collect one I and Q sample of the target's baseband echo response in each coarse range bin. These samples are the frequency-domain measurements of the target's spectral profile.

3. In each coarse range bin, apply an Inverse Fourier transform (IFFT) on the n complex samples to obtain an n-element SRP of the target in the respective coarse range bin.

The simulation results have shown that high resolution can be obtained by IFFT but target should be only in one coarse range bin. This method is unsuitable for processing SAR images, mainly because the target energy spills over to the next coarse range bin due to matched filtering, which causes multiple "ghost" images to appear in the final range profile. More information regarding this method can be found in [9].

1.2.2 Time domain method

The time domain method involves the reconstruction of a wide-band chirp waveform in time domain from a group of narrow-band chirp waveforms stepped in frequency [12].

The required signal processing steps are as follows:

1. Upsampling of narrow-bandwidth pulses.

2. Frequency-shift of these basebanded narrow-band pulses.

3. Applying a phase correcting term to each pulse so that it's phase coincides with the phase of the wide-bandwidth signal. Otherwise, there will be discontinuities in the phase of the wide-bandwidth signal.

4. Applying an inverse Fourier transform to the narrow-bandwidth spectra, which effectively yields the time-domain return. In order to approximate the radar returns as closely as possible these signals are shifted in time domain before addition.

The reconstructed wide-bandwidth signal is obtained by coherently adding the processed narrow-bandwidth pulses.

For more information about this method please refer [6, 12].

This method does not suffer from ghost-target drawback and is thus more suitable for SAR processing applications. However, this method is inefficient mainly because of the upsampling requirement of the narrow-bandwidth signals.

1.2.3 Frequency domain method

In the frequency domain approach, a broad bandwidth range profile is constructed by piecing together N received echoes in the frequency domain [10, 23].

It is noted that a time domain equivalent of this spectral reconstruction approach is also described in [10], distinct from the time method described in [12].

1.3 Motivation behind implementing the SFP algorithm in a parallel platform

SAR is characterized by high data rates and requires intensive computation to process the raw data into a focused image. The SAR raw data which extends in slant range across swath and in azimuth along the flight path can

not resemble a map of terrain without processing. This results in excessive processing time and demands gigabytes of memory. The relationship between computational requirements and the resolution is as follows [13]:

1) The number of pixels in the image increases as the inverse-square of the resolution.

2) The number of operations per pixel increases with the logarithmic of FFT size which is inversely proportional to the resolution.

In the case of real time applications such as military reconnaissance, disaster prediction etc., supercomputers and massively parallel processors (MPP) are required to realize the requisite processing time.

In the range compression stage, a typical unfocused image may contain as many as 20000 range lines which takes hours or more to process using the fastest sequential algorithm. Concurrently processing these range lines in parallel is the best possible solution to this problem. Also, since the stepped-frequency processing algorithm consists only of single dimensional Fourier transforms and complex multiplications, it lends itself well to parallel computation.

MPPs combine a few hundred to a few thousand CPUs in a single large cabinet connected to hundreds of gigabytes of memory. They offer enormous computational power and are used to solve Computational Grand Challenge problems such as global climate modelling, pollution dispersion etc [32]. But the massively parallel processors such as CrayT3 and Intel Big Blues are limited to projects with big budget as they are extremely expensive machine to purchase and maintain [15].

However, in recent years, the huge consumption of **COTS** (commercial-off-the-self) technologies by the business community has resulted in the availability of low cost, powerful desktop PC's. These PCs, together with Open Source (OS) software, including operating systems such as Linux, have resulted in a major breakthrough in low cost parallel computing. The *Beowulf (pile of PCs)* cluster is one of the most famous fast clusters of workstations and was first constructed using 16DX$ processors connected by Ethernet in 1994 at NASA Goddard Centre [14]. The parallel cluster used by the UCT *Radar & remote sensing group* (RRSG) for testing the parallel implementa-

tions is a Beowulf cluster called *Gollach* .

RRSG's **G2 SAR processor** which is capable of producing focused SASAR images has been mapped to a parallel version also by taking advantage of the gollach cluster [15].

1.4 Objectives

The objectives of this dissertation are as follows:

1. Understanding the effectiveness and suitability of SFP in high resolution radar imaging.
2. Studying the signal processing steps involved in the frequency-domain method of reconstructing a wide-bandwidth chirp pulse.
3. Verifying and testing the SFP algorithm in Matlab and in C using the simulated data set and real data set obtained from CSIR.
4. Implementing the parallel version of the SFP algorithm in a cluster of workstations (COWs) to reduce the overall image processing time.
5. Measuring the performance of the parallel system.

The availability of a stepped-frequency radar in the CSIR was the primary motivation for this project. The conceptual and mathematical modelling of reconstructing a target's reflectivity spectrum was available a priori (Wilkinson 1996). The coding and implementation of this algorithm on simulated and real data comprises the author's work. The parallel version of the stepped frequency processing algorithm was also developed and independently implemented in a cluster by the author.

1.5 Contents of the chapter

The rest of the thesis is structured as follows:

Chapter 2 reviews the relevant theory and concepts of the stepped frequency processing technique. A description is given to show that with little modification a single frequency radar system can be used as a stepped-frequency radar. It includes a block diagram of stepped-frequency radar. In this chapter, the frequency domain approach of reconstructing wide bandwidth pulse is discussed in detail. It also summarizes the methods and steps involved in making the compensation filter which flattens the ripples of the wideband spectrum.

Chapter 3 presents and describes the entire structure of the stepped-frequency processing algorithm applied on artificially simulated data. It also displays and discusses the results obtained using the artificially simulated data.

Chapter 4 presents and describes the program flow of the stepped-frequency processing algorithm applied on real data acquired from CSIR's RoofSAR. It displays and analyses the results obtained using the RoofSAR parameters and artificially generated scene. The results achieved by taking the range compressed real data from CSIR is also presented and discussed.

Chapter 5 covers the some fundamental theory of parallel processing. This chapter provides the taxonomy of a parallel processor, parallel processing theory, network criteria and an overview of parallel virtual machine (PVM) which is used as a messaging middleware in the parallel implementation. A comparison between PVM and message passing interface (MPI) is also given.

Chapter 6 describes the designing procedure of the parallel algorithm from the sequential algorithm. The sequence of communication between the master and the worker process is described with a diagram.

Chapter 7 displays the timing results obtained using simulated and real data sets and analyses the performance of the parallel system in comparion to the sequential one. It also gives a brief description of the parallel cluster used to run the parallel version of the stepped-frequency algorithm.

Chapter 8 gives the conclusions of the results and recommendations for future work.

Chapter 2

Theory and concepts involved in SFP

This chapter covers some of the basic theory needed to understand the stepped frequency algorithm. The concepts of range resolution, matched filtering and sidelobe reduction are reviewed. The basic stepped frequency hardware is described. It also describes the signal processing steps involved in the frequency domain method of processing stepped frequency data.

2.1 Range resolution

The task of a radar is to detect a target and then provide additional information about the target. The targets must be resolved before one can measure the parameters of the individual target. In the context of radar imaging, range resolution is interpreted as the ability to distinguish contributions of isolated scatterers which are closely spaced in range. Two objects can be distinguished if the leading edge of the pulse echo from the more distant object arrives at the antenna later than the trailing object of the pulse echo from the nearer object [16].

In practice, radar resolution is defined as the width of a point target response, measured between the half power or **3dB** points or as the equivalent rectangular width of a pulse [17]. The term 3dB signifies that the amplitude

is reduced to 0.707 and the power is therefore reduced to 0.5.

For example, if a matched filter is used (without windowing), i.e. pass-band is rectangular, the output of a matched filter in the time domain is a sinc function $\mathrm{Sa}(\pi Bt)$. The "3dB" width of a sinc function is measured as the width of the pulse at its half power level and is given by

$$\delta t_{3dB} \approx \frac{0.89}{B} \tag{2.1}$$

and the available resolution δR_{3dB} is inversely proportional to the system bandwidth B and is given by

$$\delta R_{3dB} \approx \frac{c}{2B}(0.89) \tag{2.2}$$

where c is the speed of light.

The slant range resolution ρ_r of a radar which uses short pulse waveforms (e.g. monochrome pulse) is dependent upon the signal pulse length T_p (the distance a pulse travel in time t_p) via system bandwidth and is given by

$$\rho_r = \frac{c}{2B} \approx \frac{cT_p}{2} \tag{2.3}$$

From equ. 2.3 we can see that on decreasing the pulse length, the resolution decreases or gets finer. But in doing so, overall energy in the pulse also decreases, resulting in lower signal to noise ratio (after matched filtering).

2.2 The Linear FM chirp waveform

The requirement of a good range resolution as well as satisfactory SNR, require a pulse of large energy and high bandwidth which is difficult to obtained simultaneously by using short pulse waveforms. The unavailability of higher energy and hence the average peak power simuataneously in short pulse is resolved by pulse-compression technique. Pulse compression makes it possible to transmit longer pulses without sacrificing range resolution, since the bandwidth is increased by modulating either the amplitude, phase or frequency space of the transmitted pulse [16].

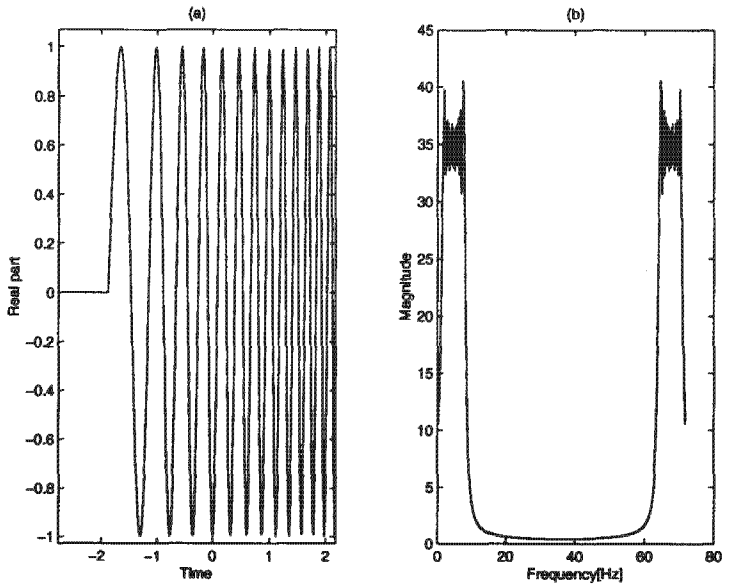

Figure 2.1: *The real-part of a FM chirp pulse in time domain and the magnitude of it's Fourier spectrum*

Radar using such modulation schemes are common, and are known as *chirp systems* where the frequency sweep during a pulse is linear. The real RF signal of linear FM chirp is represented as

$$v_{RF}(t) = rect(\frac{t}{T_p}) \cos(2\pi[f_0 t + \frac{\gamma t^2}{2}]) \tag{2.4}$$

where $f_0$is the carrier frequency which is a linear function of time and ramps up throughout the pulse length at a rate γ known as *chirp rate* in Hz/sec and T_p defines the pulse length. It's phase is a quadratic function of time. The bandwidth or the *sweep range* $\triangle f$ of the pulse is equal to γT. The real part of a linear FM chirp pulse in the time domain (graph(a)) and it's corresponding frequency domain representation (graph(b)) is shown in Figure 2.1.

A factor which defines the spectral properties of a chirp waveform is the *Dispersion factor,* also known as time-bandwidth product and is represented as $D = \triangle f T_p = \gamma T_p^2$ where bandwidth refers to the sweep range $\triangle f$. For $D > 50$ the magnitude of the Fourier transform of Linear FM chirp starts to

look rectangular [16].

2.3 Matched filter

Practical receivers estimate the transmitted signal by using a technique known as matched filtering. A receiver employing such a technique filters the received signal whose shape is matched to the transmitted signal's pulse shape. The matched filter response of the transmitted pulse $f(t)$ in time domain and in frequency domain are as follows

$$h(t) = kf^*(-t + t_m) \tag{2.5}$$

$$H(\omega) = kF^*(\omega)e^{-j\omega t_m} \tag{2.6}$$

From the equ. 2.5 we can see that the matched filter is a mirror image of $f^*(t)$ where $f^*(t)$ is the complex conjugate of $f(t)$, but the zero point is shifted to location $t = t_m$, and scaled by an arbitrary constant k which can be set to unity. In the frequency domain, the matched filter is expressed as shown in equ. 2.6 [19].

The matched filter output in frequency and time domain is given by

$$F_0(\omega) = F(\omega)H(\omega) = k\left|F(\omega)\right|^2 e^{-j\omega t_m} \tag{2.7}$$

$$f_0(t) = \Im^{-1}[F_0(\omega)] \tag{2.8}$$

In this matched filter operation, the dominant frequencies containing the data in the received signal are passed while the remaining weaker frequencies are attenuated. It limits the noise spectrum that is passed on to the subsequent stages in the receiver. It can be shown that the matched filter optimizes the peak output *signal to noise ratio* (SNR) which is proportional to the energy of the pulse.

The maximum SNR for a signal in additive white Gaussian noise is given by signal energy E divided by the noise power spectral density N_0 [17, 18]. If returns from n pulses are processed, the total energy and SNR increase by

the factor n, i.e.

$$SNR \leq \frac{E_{total}}{N_0} = \frac{\sum_n \int |F_n(\omega)|^2 d\omega}{N_0} \tag{2.9}$$

2.4 Windowing function

When a signal being analysed is not periodic with the observation period is processed by a Fourier transform, the discontinuities at the extremities of the observation interval cause spurious spectral responses known as *leakage* [21]. A weighting function may be applied to the data to reduce the effects of spectral leakage. The multiplicative weighting function or *windows* are tapered smoothly to zero at the extreme edges so that continuity of the periodic extension is increased. The design of tapered windows involves a trade-off between suppression of leakage sidelobes and reduced spectral resolution. In the context of radar signal processing, windowing is applied in the frequency domain to reduce the sidelobes in the time domain point target response.

A rectangular system spectrum exhibits the highest spectral resolution and the highest sidelobe level of -13 dB (sidelobes of $\frac{sinx}{x}$). High sidelobes are undesirable as the bright targets are replicated.

In order to reduce those high sidelobes, a commonly used weighting function called the *Hanning window* is applied resulting in very low sidelobe levels and negligible broadening of the main lobe. The Hanning Window is expressed as

$$\omega(n) = (0.5)[1 - \cos(2\pi n/N)] \tag{2.10}$$

for $n = 0, 1, ..., (N-1)$. N represents the number of samples.

2.5 Modifying an existing single-frequency radar system to a stepped frequency radar

2.5.1 Introduction

From the discussion in the previous section it is evident that resolution and sample rate is directly related to the bandwidth of the system. For example, The SASAR1 VHF SAR has a bandwidth of 12 MHz which implies a range resolution of about 12m. A bandwidth of 100 MHz is required in order to increase the resolution from 12m to 1.5m [4]. But for sending and receiving wide-band pulses, we need to design a broad-band antenna and an A/D converter with increased sample rate (*Nyquist criteria*).

A stepped frequency radar system gives us an economically viable path to upgrade an existing single frequency SAR system to a high resolution system by avoiding the requirements for wide instaneous bandwidth and high sample rate.

2.5.2 Stepped-frequency radar model

In a stepped frequency radar system, a series of bursts of narrow-band pulses are transmitted and received. Each burst consist of sequences of stepped frequency pulses. To achieve high resolution range profiles, these stepped-frequency pulses are combined as discrete frequency steps with a kind of signal processing method.

As shown in figure 2.2 the *frequency modulated* (FM) chirp pulses are transmitted via transmitter Tx at the centre frequency f_0, reflected off targets on the ground and are received by the receiver Rx at centre frequency at some time delay t_d which is equal to $2R/c$ where R is the distance from the target to the antenna. The incoming radiation is amplified, mixed down to baseband and the analog signal split into in-phase $I(t)$ and quadrature $Q(t)$ components and sampled with an A/D converter.

A special requirement for step-frequency radar system is a *frequency synthesiser* or a voltage controlled oscillator (VCO) which is programmed to

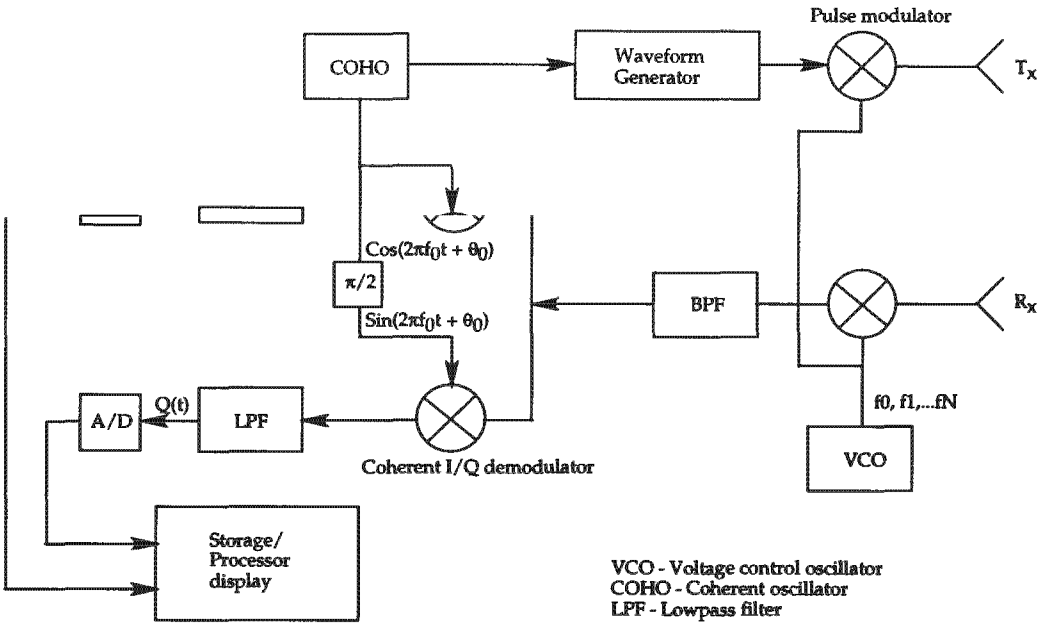

Figure 2.2: *A block diagram of stepped-frequency radar.*

switch rapidly from frequency to frequency while maintaining the phase coherence [22]. Each burst of the transmitted signal contains N number of transmitted pulses whose centre frequency is expressed as

$$f_i = f_0 + i.\triangle f \tag{2.11}$$

where $i = 1, ..., N-1$, f_0 is the centre frequency and $\triangle f$ is the step size. Actually, the requirement for radar receiver bandwidth is only a little greater than $\triangle f$. A step-frequency SAR system is able to improve its range resolution without imposing an extra burden to the radar receiver. Also, there is no need to modify the existing A/D converter since the instantaneous bandwidth received at the radar is a fraction of the total synthesized bandwidth. It is desirable to have a A/D converter which samples at lower rate because sampling at a lower rate or with a larger number of bits increases the receiver dynamic range which in turn reduces the possibility of receiver saturation [6]. The *Radio frequency interference* (RFI) encountered in VHF/UHF-band is

often stronger than the signal. Receiver saturation suppresses the received signal which degrades the image quality and makes the RFI suppression task difficult.

2.5.3 I/Q Demodulation

For high frequency carriers, **coherent I/Q demodulator** (shown in Figure 2.2) or *in-phase* and *quadrature* demodulator is used to recover the necessary information i.e., amplitude A and the phase θ from the received signal.

In the simplest form, the transmitted signal with frequency f_0 and phase θ_0 is expressed by equ. 2.12

$$v_{tx(t)} = \cos(2\pi f_0 t + \theta_0) \tag{2.12}$$

The received signal, for a point target, is shifted by a phase angle θ relative to the transmitted signal and is expressed as follows:

$$v_{rx(t)} = A\cos(2\pi f_0 t + \theta_0 - \theta) \tag{2.13}$$

where constant A account for the various system and propagation effects[16].

As shown in Figure 2.2 the received signal is fed into one of the ports of two multipliers. One port of the multipliers is fed by the reference signal and the other port by the same signal shifted in phase by $\frac{\pi}{2}$ degrees. The output of the upper multiplier after expanding in trigonometric identity is given by equ. 2.14.

$$v(t) = A/2\cos[2(2\pi f_0 t) + 2\theta_0 - \theta] + A/2\cos(\theta) \tag{2.14}$$

The first term represents a signal with frequency twice that of the carrier, and the second term is a dc component proportional to the product of signal amplitude and the cosine of the phase angle. The low-pass filter (LPF) shown in Figure 2.2 rejects the high-frequency term, leaving an output proportional to $A\cos\theta$. The terms *inphase* and *quadrature* refer to the fact that two components of the received signal are recovered, one in phase with the reference ($A\cos\theta$) and the other in-phase quadrature with the reference ($A\sin\theta$) [21].

The received signal processed by the coherent demodulator can be expressed by equ. 2.15

$$v_{rx(t)} = \zeta Re[Ae^{(-j\theta)}.e^{(j2\pi f_0 t+\theta_0)}] \tag{2.15}$$

The quantity $Aexp(-j\theta)$ is called the *complex envelope* of the signal and represents the modulation of the carrier signal which has been expressed by the second exponential. The coherent demodulator provides the translation to baseband while retaining both the real and imaginary components of the complex signal. The reason the signal is mixed down to baseband is to reduce the sampling rate requirements which is determined by the modulation bandwidth. After digitizing, the signal is recorded and stored for processing.

A disadvantage for a stepped-frequency SAR system is that there is a stricter limitation for choosing a PRF to meet the need of ambiguities both in range and in azimuth. In order to avoid *range ambiguities* and *Doppler aliasing effects,* the limits on the PRF are set as follows [7]:

$$\frac{v_d.N}{B_a} \leq PRF < \frac{c}{2 \triangle R_s} \tag{2.16}$$

where v_d is the relative velocity between air-plane and ground, $B_a$represents the Doppler bandwidth and $\triangle R_s$ is the swath width. From equ. 2.16 it can be inferred that to the achieve same azimuth resolution the PRF has to be several times higher than the normal SAR system. Therefore, the reduction of range swath is the only way to avoid range ambiguity in the final image [22].

2.6 Reconstruction of target reflectivity spectrum using stepped frequency waveforms

2.6.1 Introduction

The frequency domain approach is based on reconstructing a broad region of target's impulse response in the frequency domain which is known as target's

reflectivity spectrum.

In order to obtain a greater portion of target reflectivity spectrum, which in turn yields higher range resolution, a series of narrow bandwidth pulses separated by frequency step is transmitted. Each narrow-band pulse maps a portion of the target reflectivity spectrum to base band. These sub-spectrums are coherently added by shifting them appropriately to reconstruct the target reflectivity spectrum.

The sections 2.6.2 to 2.6.4 describes the signal processing steps involved in frequency domain reconstruction of target reflectivity spectrum. These two following sections are paraphrased from [10, 23].

2.6.2 Signal modelling

To model a collection of point scatters, first a base-banded linear FM chirp waveform is sent as a transmitted pulse which is described by

$$p(t) = A rect(t/T_p) e^{j\pi\gamma t^2} \tag{2.17}$$

where A is the amplitude, T_p is the pulse length and is γ the chirp rate. The bandwidth of the chirp pulse is given by $B_{tx} = \gamma T_p$.

The transmitted RF signal is given by

$$v_{tx} = p(t) e^{j2\pi f_c t} \tag{2.18}$$

where f_c is the carrier frequency.

The received signal is the convolution of target reflectivity function or impulse response $\zeta(t)$ with transmitted pulse.

$$v_{rx}(t) = \zeta(t) \otimes v_{tx}(t) = \int \zeta(t-\tau) v_{tx}(\tau) d\tau \tag{2.19}$$

where time delay τ is given by $2R/c$.

The target reflectivity function or impulse response $\zeta(t)$ can be related to backscatter coefficient $\beta(t)$ i.e., a projection of scene's reflectivity charac-

teristic into slant range and is described by

$$\zeta(t) = \frac{\beta(t)}{R^2} \tag{2.20}$$

After coherent demodulation, the signal at baseband is given by equ. 2.22.

$$v_{bb}(t) = v_{rx}(t)e^{-j2\pi f_c t} \tag{2.21}$$

$$v_{bb}(t) = \int \zeta(t-\tau)p(\tau)e^{-j2\pi f_c(t-\tau)}p(\tau)d\tau = [\zeta(t)e^{-j2\pi f_c t}] \otimes p(t) \tag{2.22}$$

A linear base-banded filter $H_{bb}(f)$ is included to model the receiver chain. The signal is sampled at a complex sampling rate of $f_s > B_{tx}$.

In the frequency domain the expression for the baseband signal including noise is given by the equ. 2.23.

$$V_{bb}(f) = Z(f+f_c)P(f)H_{bb}(f) + N(f+f_c)H(f) \tag{2.23}$$

The spectral representation of the received signal is viewed as windowed version of target's reflectivity spectrum $Z(f) = \Im\{\zeta(t)\}$, where the centre frequency determines the position of the window and the shape is the same as that of the transmitted pulse. The receiver noise is modelled by an equivalent referred RF noise signal $n(t)$ with spectrum $N(f)$.

The next step involves matched filtering the individual frequency steps and is known as range compression technique. Matched filter H_{mf} is time-reversed complex conjugate of transmitted pulse $p(t)$. The transfer function of matched filter is given by $H_{mf} = P^*(f)$.

In the frequency domain the range-compression technique is given by

$$V(f) = V_{bb}(f)H_{mf} \tag{2.24}$$

The phase of H_{mf} is always chosen to cancel the phase of $P(f)H_{bb}(f)$. This maximizes signal-to-noise ratio (SNR) and compresses the encoded

signal.

It is noted that if a received signal from a single point target is matched filtered, the time domain impulse response is approximately a sinc function with 3dB resolution and is given by equ. 2.25.

$$v(t) = [\zeta(t)e^{-j2\pi f_0 t} + n(t)e^{-j2\pi f_0 t}] \otimes Sa(\pi B_{tx} t).B_{tx} \tag{2.25}$$

where $Sa(x) = \frac{sin(x)}{x}$.

An appropriate frequency domain window function can be applied in order to reduce the sidelobes at the expense of main lobe broadening.

2.6.3 Coherent addition of sub-spectrum

A wide-band signal spectrum can be constructed by stacking together several adjacent sub-portions of the spectrum of bandwidth B_{tx} incrementing the carrier frequency f_0 by Δf (frequency step size) each time. The frequency step Δf should satisfy the inequality $\Delta f \leq B_{tx}$.

Assuming a sequence of N adjacent windows indexed by $(i = 0, ..., n-1)$, a broad region of the frequency spectrum can be reconstructed symmetrical about zero, by shifting each spectrum at baseband by an amount

$$\delta f_i = (i + \frac{1-n}{2}) \Delta f \tag{2.26}$$

in the positive direction, and adding together the shifted versions. The reconstructed spectrum $V'(f)$ of bandwidth $B_t \approx N \Delta f$ is shown in figure 2.3.

The centre frequency $f_c'$of the entire reconstructed spectrum is

$$f_c' = \frac{f_0 + f_{n-1}}{2} \tag{2.27}$$

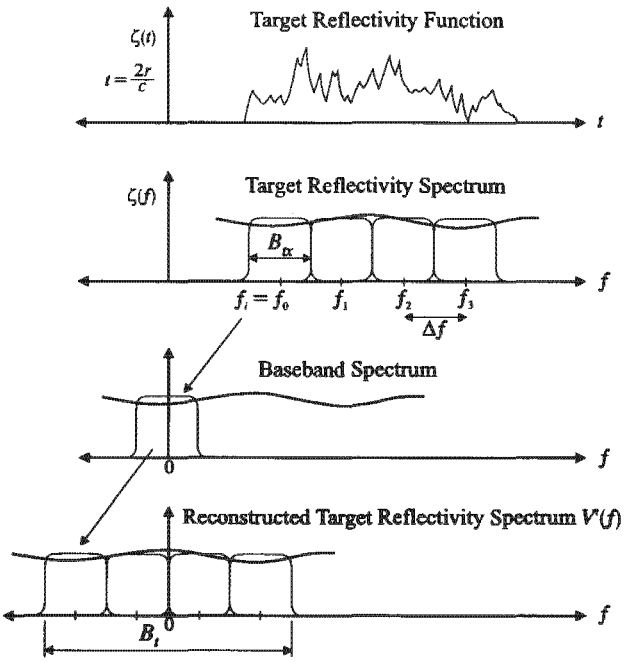

Figure 2.3: *Reconstruction of target reflectivity function or 4 transmitted pulses, each with frequency f_i and bandwidth B_{tx}[23].*

The reconstructed spectrum can be expressed as in equ. 2.29.

$$V'(f) = [Z(f + f_c') + N(f + f_c')] \sum W(f - \delta f_i) \tag{2.28}$$

$$= [Z(f + f_c') + N(f + f_c')]W'(f) \tag{2.29}$$

where $W(f) = p(f)H_{bb}(f)H_{mf}(f)$.

2.6.4 Construction of compensation filter

The reconstructed spectra contain ripples near the edges which causes repeats of the sinc function in time domain [24, 25]. Compensation filter is applied to minimize the ripple followed by normalisation to produce a flat response.

So, if the compensation filter $H'(f)$ is chosen such that $W'(f)H'(f) = rect(\frac{f}{B_t})$, then the time-domain signal is given by equ. 2.30.

$$v'(t) = [\zeta(t)e^{-j2\pi f_c' t} + n(t)e^{-j2\pi f_c' t}] \otimes Sa(\pi B_t t).B_t \qquad (2.30)$$

where B_t is the total bandwidth achieved after combining N frequency steps. The 3dB resolution is $\frac{1}{B_t}$, which is a factor N better than that achieved using a single frequency of bandwidth B_{tx}.

The compensation filter $H(f)$ is reconstructed by two methods i.e.

1. Measuring the return of a physical corner reflector in scene.
2. Simulating a single calibration point target.

The received echoes from corner reflector are then combined as described in the previous section. The corner reflector is most accurate for processing real data because it accounts for all linear system effects. In case of simulating a calibration point target, the reconstruction method of the compensation filter is as follows.

1. It is important to ensure that there is no aliasing and that the individual sub-spectrums are bandlimited.
2. Position the calibration point target at zero or shift it to zero so that the position of the high resolution profile in the scene will be at the same target range as chosen in the simulation.
3. The sub-spectra used in constructing $H'(f)$ should be range compressed with a matched filter.
4. Obtain an estimate of the combined impulse response spectrum

$$U'(f) = \sum P(f - \delta f_i)X(f - \delta f_i) \qquad (2.31)$$

5. Invert the combined spectrum $U'(f)$ to get the compensation filter $H'(f)$. This will effectively smoothen any ripples at the sub-spectra boundaries, yielding the desired time-domain impulse response.

2.6.5 Time domain Output

The final processing steps involved

- Application of a window for side lobe reduction (e.g. Hanning window).
- Inverse Fourier transform of the reconstructed spectrum.

Chapter 3

Practical implementation on artificially simulated data

3.1 Introduction

A stepped-frequency radar system was simulated using the artificially generated data sets in order to verify and illustrate the method described in the previous chapter.

This chapter describes how the exact parameters were generated, the structure of the entire stepped-frequency processing and the simulation results obtained.

3.1.1 Algorithm for non integer and integer shifts

Aforementioned, the subspectras are shifted in frequency before addition. The frequency shift is done on discrete number of samples. Thus, the shift in samples per spectra i.e. $\frac{\Delta f}{\delta f}$ can be a non-integer or an integer number, where Δf is the frequency step size and δf is the sample spacing in frequency domain. In case of a non-integer shift, the steps for reconstructing a wide band spectrum are as follows:

1. Define the final vector N_t in frequency domain.

2. For each sub-spectrum $X(f)$ determine the integer I_i and fractional F_i part. Since we are shifting it by a small amount, we do not need to zero-pad the vector before going to time domain.

3. Transform it into time domain $x(t) = \Im^{-1}(X(f))$ and shift each sub-spectrums by fractional part as shown in equ.3.1.

$$y(t) = x(t)e^{j2\pi F_i t} \tag{3.1}$$

4. Transform into frequency domain using $Y(f) = \Im(y(t))$.

5. Place into correct position by shifting each sub-spectra by an amount I_i into the final resultant vector N_t and add them.

The steps for integer shift are given below:

1. Define the result vector N_t.

2. Ensure that the spectral shift is an integer number of samples.

3. Shift each sub-spectra by an integer number of samples to its true position and add them all in the final vector.

The stepped frequency algorithm implemented here is done by integer shift of the sub-spectrum as the integer shift is fast and easy to process in software and the non-integer shift not very accurate.

3.1.2 The program flow

A stepped-frequency radar was simulated using a collection of $i = 1...M_{point}$ point scatterer and N frequency steps. The returned signal at the receivers is given by,

$$v_{rx}(t) = \sum_{i=1}^{M_{point}} A_i p(t - 2R_i/c) \tag{3.2}$$

where $p(t)$ is the transmitted chirp pulse. Here, we have ignored the additive noise. The summation is performed over the point scatterer, each with

relative amplitude A_i and range R_i. In this present model, start-stop approximation has been adopted, i.e. the range to the point scatterer are assumed constant during N pulses. Otherwise, interpolation is required to resample the data to a common time instant [24].

First a demodulated, basebanded chirp pulse is taken for simulation. Multiplication with a complex sinusoid corresponds to signal demodulation to baseband. At the receiver output, the signal is given by,

$$v_{bb(n)} = e^{(-j2\pi f_n t)} \sum_{i=1}^{n} a_i w_n(t - 2R_i/c) \tag{3.3}$$

where $w_n = v_{rx(n)}(t) * \zeta_n$ and ζ_n is receiver impulse response [25]. The continuous-time signal is then converted to a discrete-time signal by A/D converter and then recorded. This is the radar raw data to be processed. The sampling frequency was set above the Nyquist limit to avoid the aliasing effects. A matched filter is applied to each sub-spectrum prior to addition.

As shown in Figure 3.1 the exact radar parameters and the scene parameters were input to the **simulator**. The **simulator** simulates the artificially generated scene and the calibration point target. After coherent demodulation and matched filtering the complex return signals were stored in time domain in a output binary file. The output binary files which saves the return signals are low resolution profiles (from artificially generated scene) and single low resolution profiles (from calibrated point target). The **reconstruction algorithm** in Figure 3.1 inputs the simulator outputs and the radar and scene parameters to produce high resolution range profiles. The entire processing (i.e. Simulator and reconstruction algorithm) was first done in Matlab (listed Appendix A.1). The **reconstruction algorithm** was later converted into C language.

In the reconstruction algorithm (refer Figure 3.1), first the time domain low resolution profiles are Fourier transformed to frequency domain. The subspectrums obtained were coherently added as described in section 2.6.3, except that the positive and negative component of first sub-spectra were shifted at zero position separately in a long array and then all the other subspectrums were shifted and added in the same array by determining their

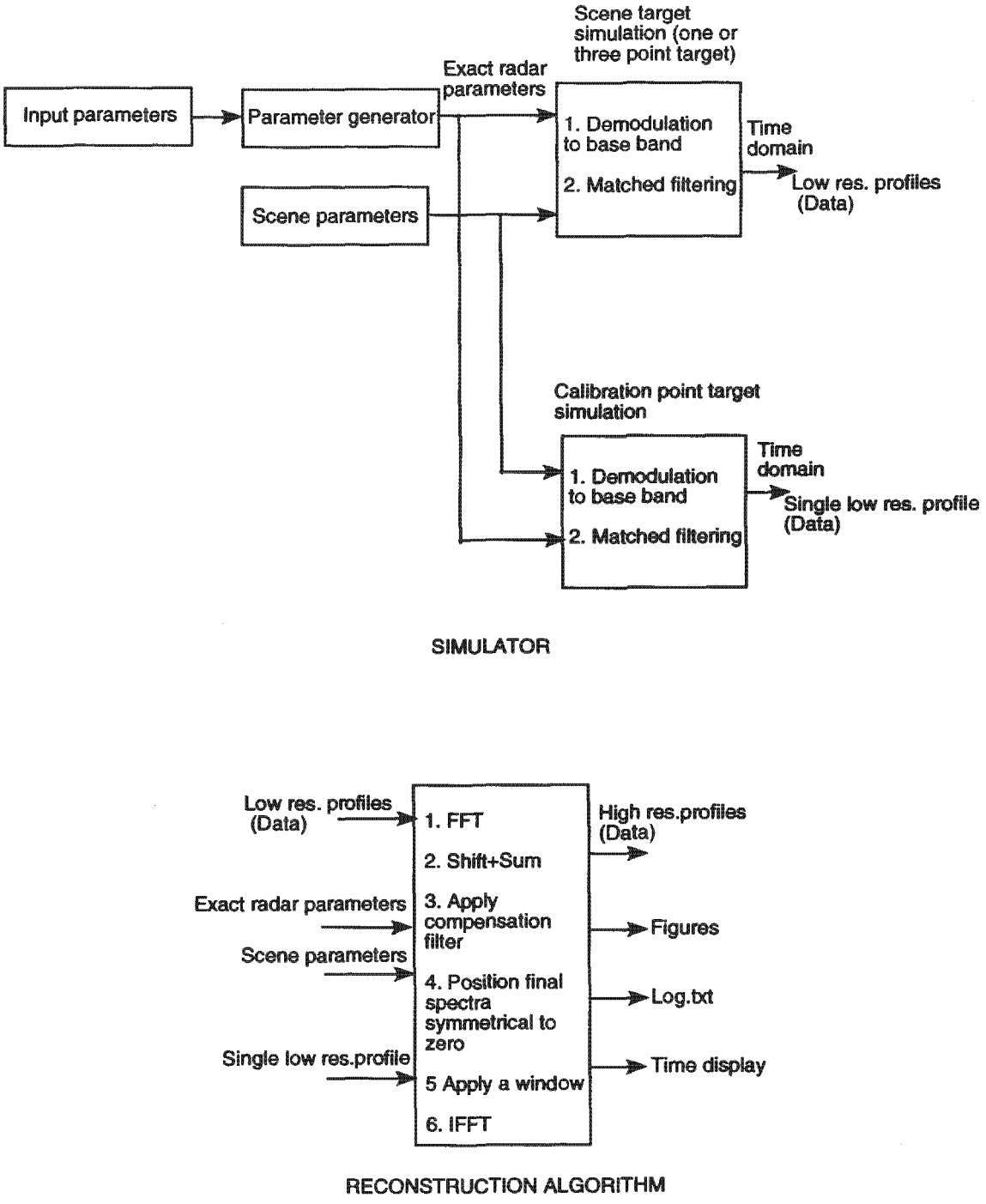

Figure 3.1: *Structure of the SFP algorithm used in the simulation*

true position in a for loop. The result of summing causes spikes in the overlap region of the reconstructed spectrum. It is important to ensure that the phases are added correctly in the overlap region other wise a gap in the final spectrum appears. This is achieved by properly matched filtering each subspectrum before addition. To construct the compensation filter, first the simulated time domain low resolution profiles from calibration point target are Fourier transformed and all subspectrums are shifted and added to make a wide band spectrum. Inverse of that spectrum gives the compensation filter. After having applied the compensation filter, the final target reflectivity spectrum was positioned symmetrical to zero. A Hanning windowing function is applied to the final spectrum which compensates for the sidelobes in the time domain. The high resolution profiles were achieved by inverse Fourier transforming the final target reflectivity spectrum.

3.1.3 Exact parameter generation (for the radar simulator)

To simulate the scene and calibration point target, first a set of sampled radar parameters were taken. Some variables including the number of samples $N_{samples}$ and shift per spectra in samples K were computed from these input parameters. The value obtained for $N_{samples}$ and K can be a non-integer number. As here i am implementing the integer shift method, $N_{samples}$ and the K were made an integer number and the other variables were recalculated accordingly. The following section shows how the exact parameters of the simulations were recalculated from the input parameters.

The input parameters are as follows:

- The first centre frequency f_0.
- The frequency step size $\triangle f$.
- The bandwidth of the narrow-band chirp pulse B_{chirp}.
- The number of steps N_{step}.

- The maximum unambiguous range R_{max}.
- The sample frequency f_s.

The following variables were computed from setup parameters.

- Maximum time delay $t_{max} = \frac{2*R_{max}}{c}$, where c is the speed of light
- Sample spacing in time domain $\delta t = \frac{1}{f_s}$.
- Number of samples $N_{samples} = \frac{t_{max}}{\delta t}$.
- Sample spacing in frequency domain $\delta f = \frac{1}{N_{samples}*\delta t}$.
- Shift in samples per sub-spectra $K = \frac{\Delta f}{\delta f}$.

Based on the values for the $N_{samples}$ and K, another setup file called *parameter generator* was created where these two parameters were forced to have an integer value and accordingly the exact new values for other parameters were computed as follows:

Let

$$N_{samples} = round(\frac{t_{max}}{\delta t}) \Rightarrow \delta f_1 = \frac{1}{N_{samples} * \delta t} \tag{3.4}$$

$$K = round(\frac{\Delta f}{\delta f_1}) \Rightarrow \delta f_2 = \frac{\Delta f}{K} \Rightarrow \delta t_2 = \frac{1}{N * \delta f_2} \tag{3.5}$$

where is δf_2 and δt_2 is the new value for sample spacing frequency and time domain respectively.

The exact setup parameters which were input to the simulator and reconstruction algorithm is summarized in Table 3.1.

Parameters	Symbol	Value
First centre frequency	f_0	10 GHz
Step size	Δf	10 MHz
Number of steps	N	5
Chirp bandwidth	B_{chirp}	16 MHz
Pulse length	T_p	6.25 μs
Sample rate	f_s	64 MHz
Number of samples	$N_{samples}$	1024
Total bandwidth	B_t	56 MHz

Table 3.1: *Artificially generated stepped-frequency radar parameters*

A step-frequency radar was simulated which has a centre frequency $f_0 =$ 10 GHz and transmits linear FM chirp of pulse length $T_p = 6.25\mu s$ and a bandwidth of $B_{chirp} = 16$ MHz sampled at $f_s = 64$ MHz. Five pulses were transmitted in the simulation, spaced at 10 MHz intervals. There was an overlap of 6 MHz between the two pulses. The shift per spectra in terms of sample number is 160.

The total radar bandwidth B_t of the reconstructed spectra is calculated as the frequency range between the two outer edges of the entire spectrum, regardless of any amount of overlap or gap that may have occurred. In my simulation, B_t was computed as follows

$$B_t = ((N-1) \times \Delta f) + B_{tx} \tag{3.6}$$

From the equ. 3.6 B_t is calculated as 56 MHz and therefore, the expected resolution is 2.67.

The artificially generated scene consists of either one or three point targets which were positioned either at 469 m or at 500 m, 1000 m and 1100 m away from origin. The calibration point target was positioned at zero. If we move away the calibration point from the origin to some distance, the target in the scene moves the same distance towards the origin.

3.1.4 Results obtained using artificially simulated data

The stepped-frequency simulation results obtained using the sample parameters summarized in Table 3.1 are shown in Figures 3.2 to 3.9.

Figure 3.2 displays the following graphs relating to the processing of a single pulse:

(a) The magnitude of the range-compressed single target return, first frequency step.

(b) The phase of the single point target return. We can see that the phase of the sinc function is a square wave and jumps from $+\pi$ to $-\pi$. The phase of the target at 469 m is given by $\arg\{\exp(-j2\pi f_0\tau)\}$ where f_0 is the first centre frequency and τ is the time delay. Having $\tau = 2R/c$, we can compute that the wrapped phase near the mainlobe -2.0944 radians, which agrees with the simulation.

(c) The left (positive) and right (negative) portion of the first sub-spectrum has been swapped around and centred in the middle for better viewing.

(d) Phase of one sub-spectrum is shown. Slope of wrapped phase indicates distance to target.

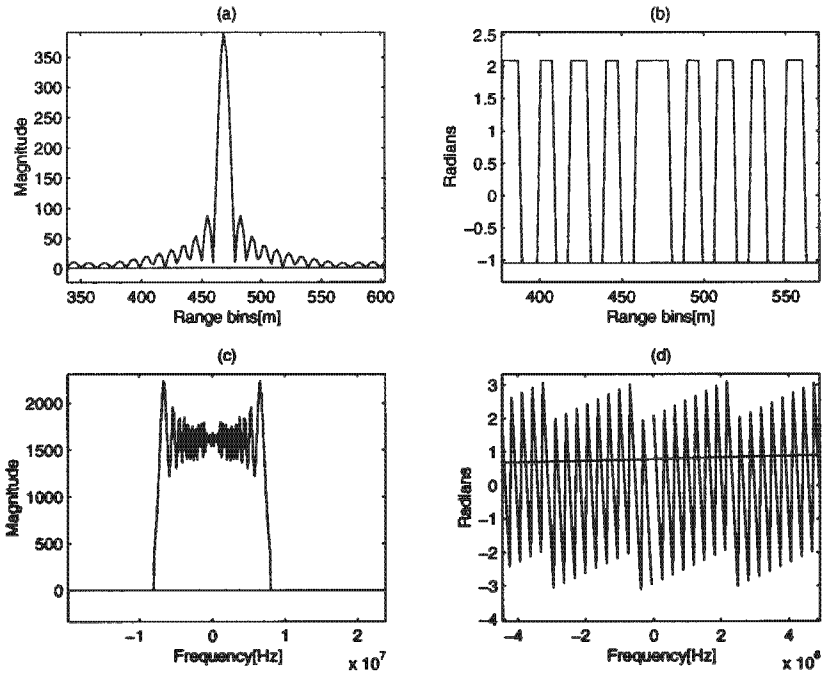

Figure 3.2: *The magnitude and phase of the point target response in time and frequency domain*

In Figure 3.3 graph(a) shows the overlapping of three sub-spectrums before addition. The phase in the overlap region is shown in graph (b). It shows that there is no phase distortion in the overlap region. Graph 3.3 (c) shows the reconstructed spectrum after coherently adding five subspectrums. The spike at every overlap region shows that the result of summing is constructive as expected. Graph 3.3 (d) is the high resolution profile achieved by applying IFFT to the reconstructed spectrum in graph 3.3 (c). In graph 3.3 (d), repeats of sinc function at interval $1/\triangle f$ was observed as the spectrum was not flattened this stage.

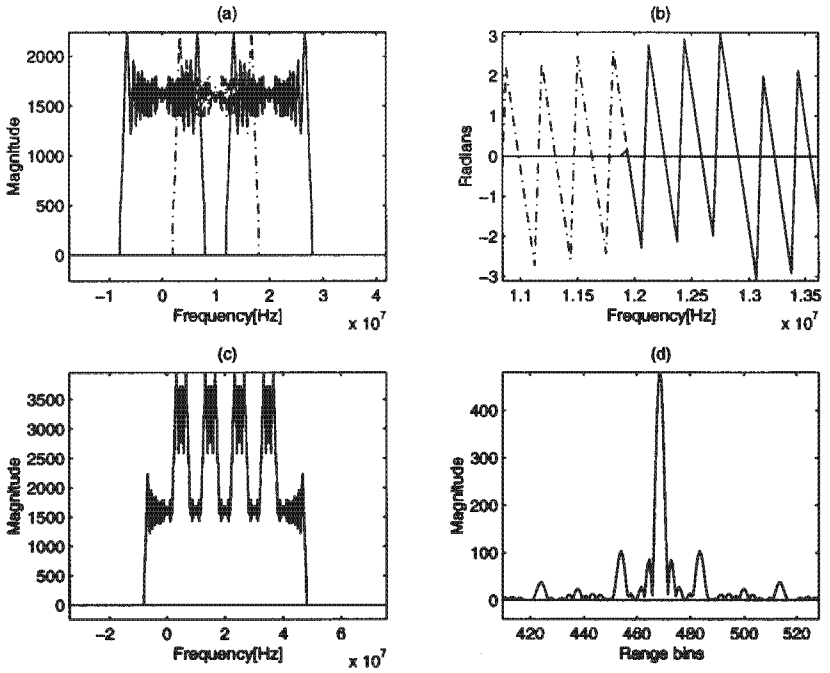

Figure 3.3: *Simulation results after coherently adding five sub-spectrums with overlap.*

The compensation filter was constructed by inverting the reconstructed spectrum from a single point target positioned at zero and is shown in Figure 3.4 (a). The target reflectivity spectrum obtained after applying the compensation filter is shown in Figure 3.4 (b). The compensation filter smoothens the ripples near the sub-spectra boundaries and produce a flat spectrum. The high resolution range profile is obtained after inverse Fourier transforming the spectrum shown in 3.4(b). Graph 3.4 (c) and 3.4 (d) shows the magnitude and phase of the high resolution profile respectively.

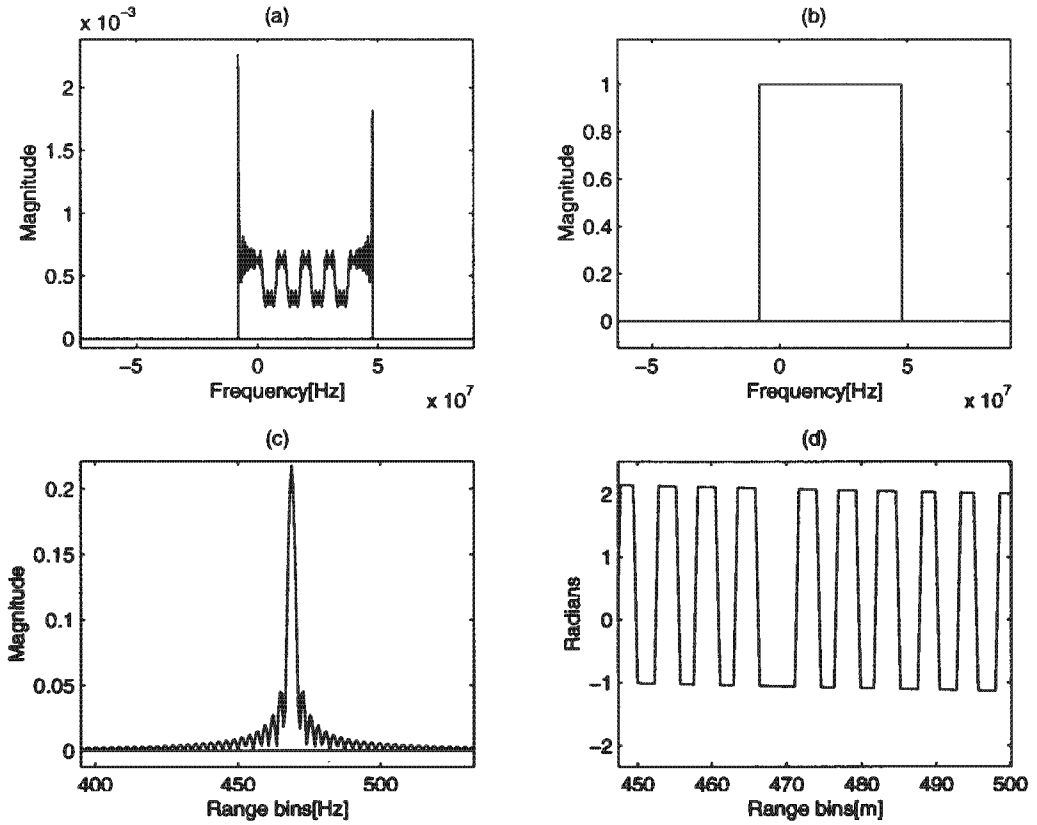

Figure 3.4: *Simulation results after applying the compensation filter.*

Figure 3.5 shows that the repeats of sinc function in graph (a) has disappeared in graph (b) after the compensation filter has been applied.

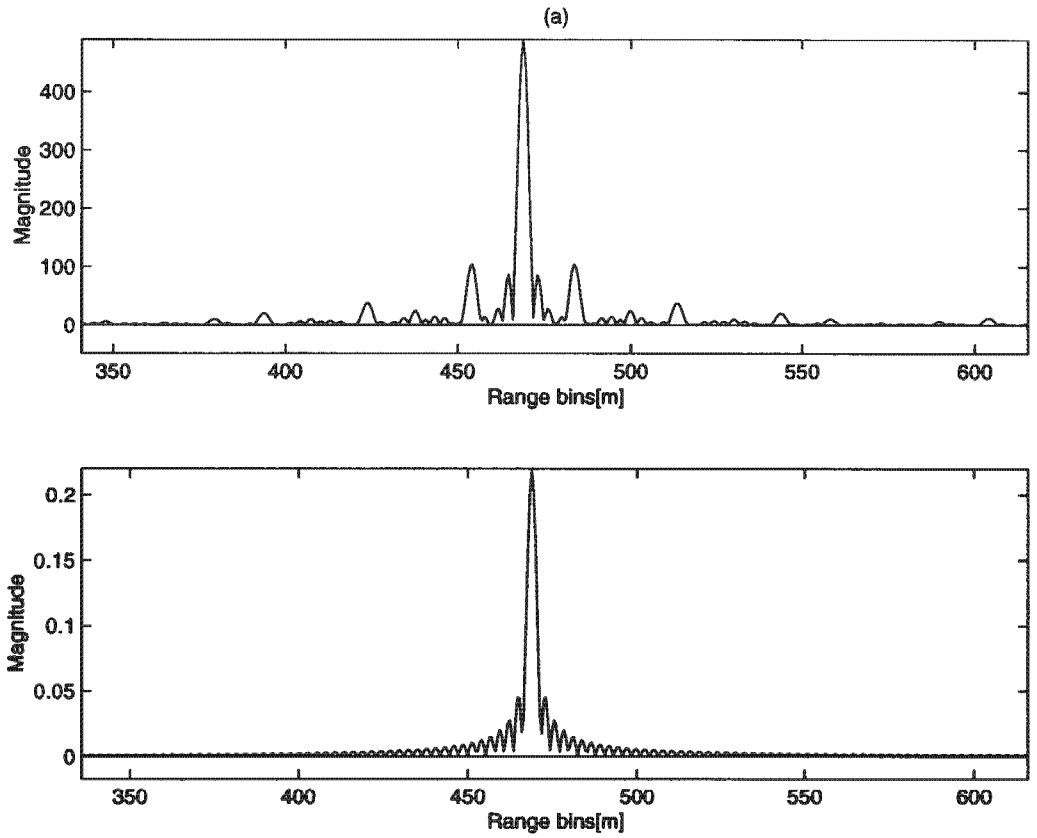

Figure 3.5: *High resolution profile before graph (a) and after graph (b) applying compensation filter*

The resolution of a single point target can be measured either using a dB plot of amplitude versus range samples or using a power plot. In Figure 3.6 graph (a) and (b) shows the power (square of amplitude) plot of the low resolution profile and the high resolution profiles against their range in meters respectively. Recall that 3dB mainlobe width means the resolution at the half power level.

Theoretical low res.	Measured low res.	Theoretical high res.	Measured high res.
8.4 m	8.5 m	2.4 m	2.5 m

Table 3.2: *Theoretical and measured low resolution and high resolution achieved after combining 5 sub-spectrums.*

Table 3.2 shows the theoretical low resolution ($\delta R = \frac{c(0.89)}{2B_{chirp}}$)and high

resolution($\delta R_t = \frac{c(0.89)}{2B_t}$) obtained after adding five sub-spectrums and also summarizes the measured low and high resolution (3 dB mainlobe width).

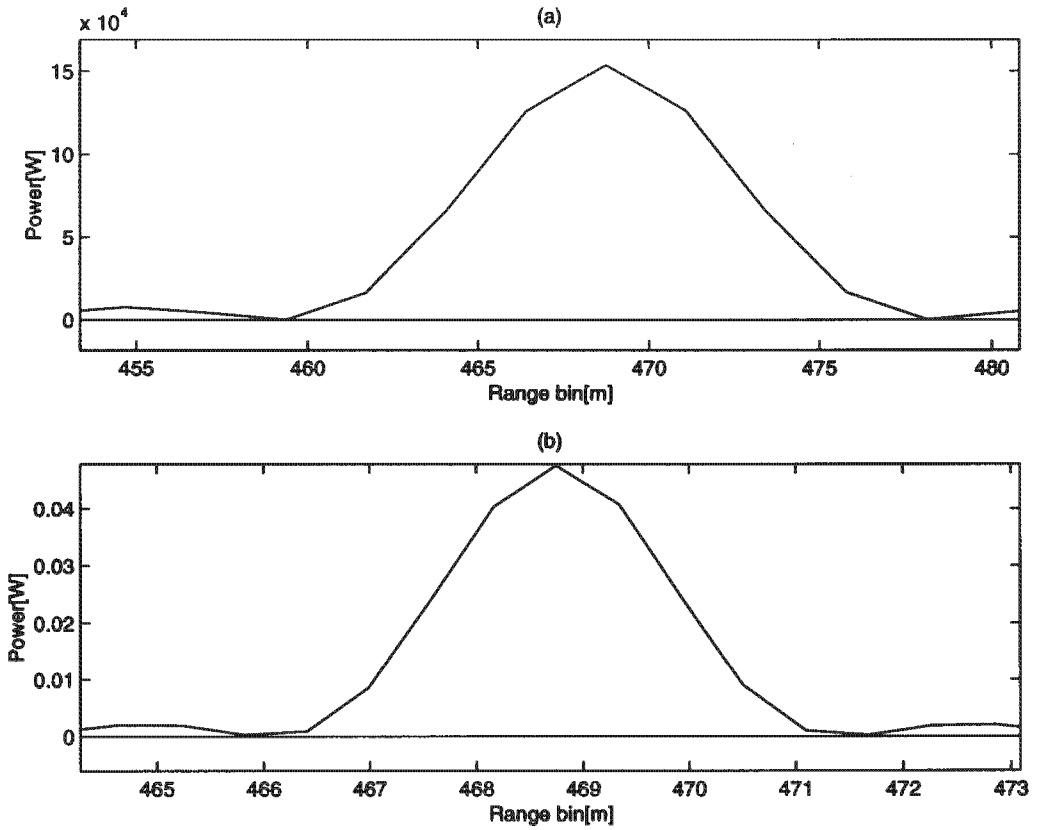

Figure 3.6: *The power plot of low resolution profile (graph (a)) and high resolution range profile (graph(b)) obtained after adding 5 subspectrums*

In Figure 3.7 (b) a Hanning window has been applied to the final flat spectrum in order to reduce the sidelobes of sinc function. Graph 3.7(c) shows the high resolution profile of a single point target obtained after having applied the Hanning window. Side lobes have reduced as compare to Figure 3.4(c) but the mainlobe has broadened. Graph 3.7 (d) shows the phase of the high res. profile in graph (c) which is similar to the graph 3.4(d).

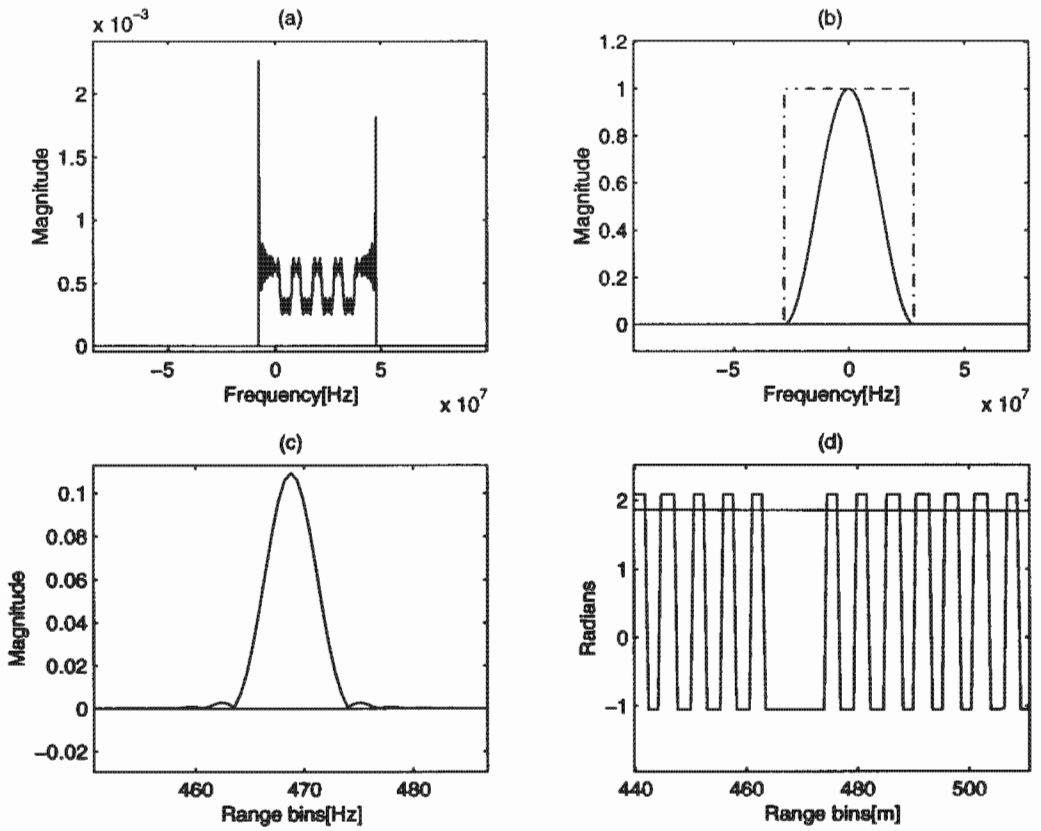

Figure 3.7: *Demonstrating side lobe reduction using hanning window*

Figure 3.8 shows that the high res. profile after both the compensation filter and the window has been applied. It can be seen that in graph 3.8 (b) the sidelobes are reduced in compare to graph 3.5 (b) but the width of the mainlobe has increased.

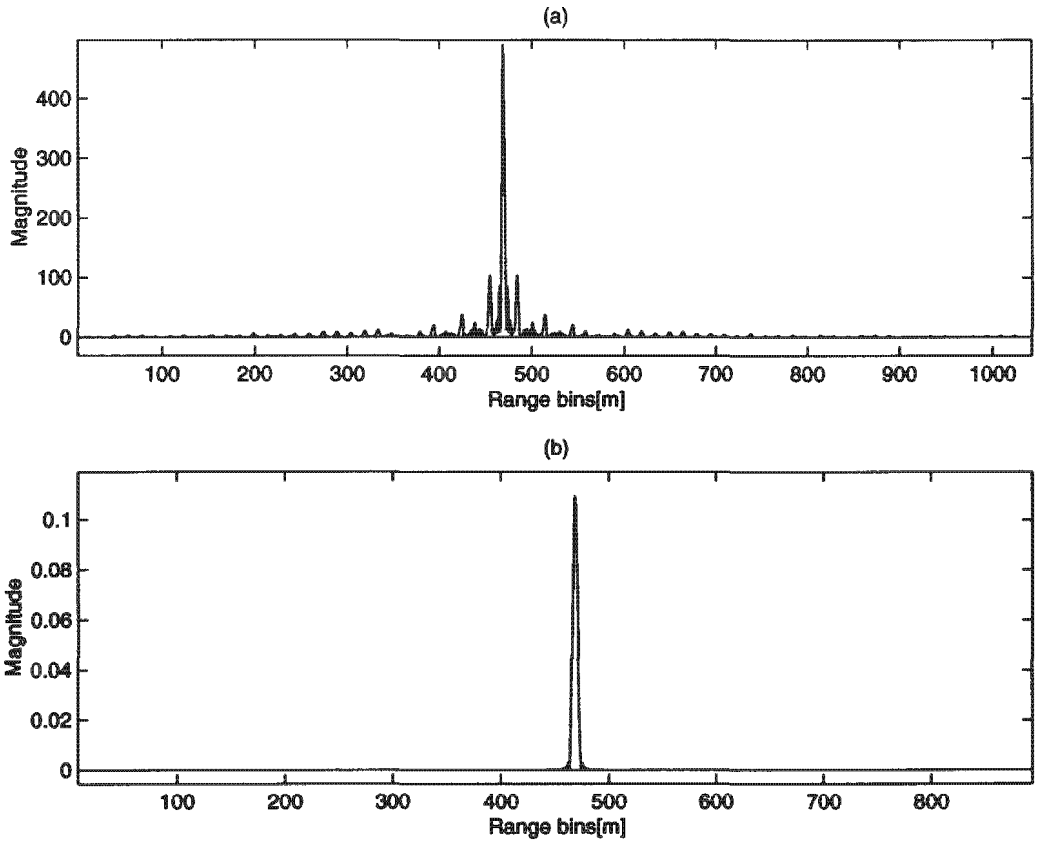

Figure 3.8: *High resolution profile a) before and after b) applying the compensation filter and hanning window*

Figure 3.9 shows the simulation results of three point targets each having backscatter coefficient (ζ) 1, 2 and 1 respectively. Graph 3.9(a) shows the first sub-spectra obtained and graph 3.9(b) is the reconstructed combined spectrum with 8 MHz of spectral overlap. Graph 3.9(c) is the target reflectivity spectrum obtained after applying the compensation filter and graph 3.9 (d) shows the reflectivity spectrum after Hanning window was applied to it.

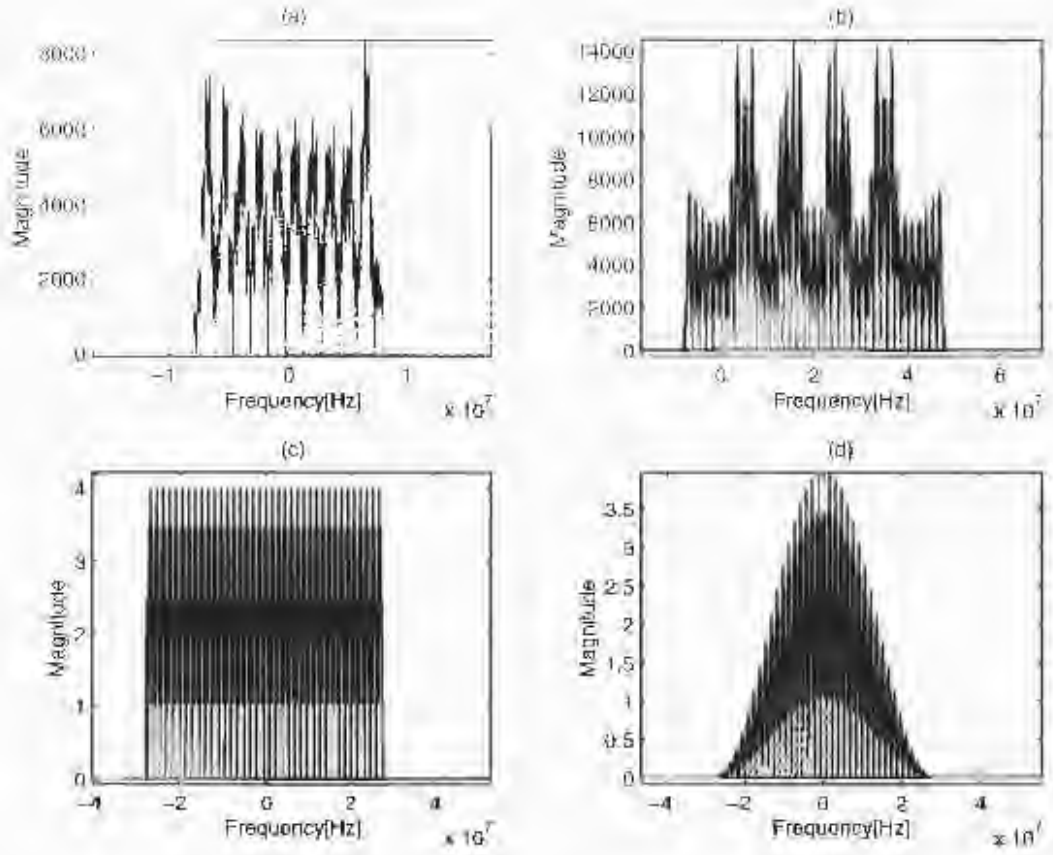

Figure 3.9: *Simulation results for three point targets.*

Figure 3.10 graph (a), (b) and (c) shows the high resolution profile obtained before flattening the spectrum, after multiplying the spectrum with the compensation filter and after windowing the spectrum respectively. We can see that repeats of graph 3.10 (a) has disappeared in graph 3.10 (b) and the sidelobes of graph 3.10 (b) have reduced in graph 3.10 (c).

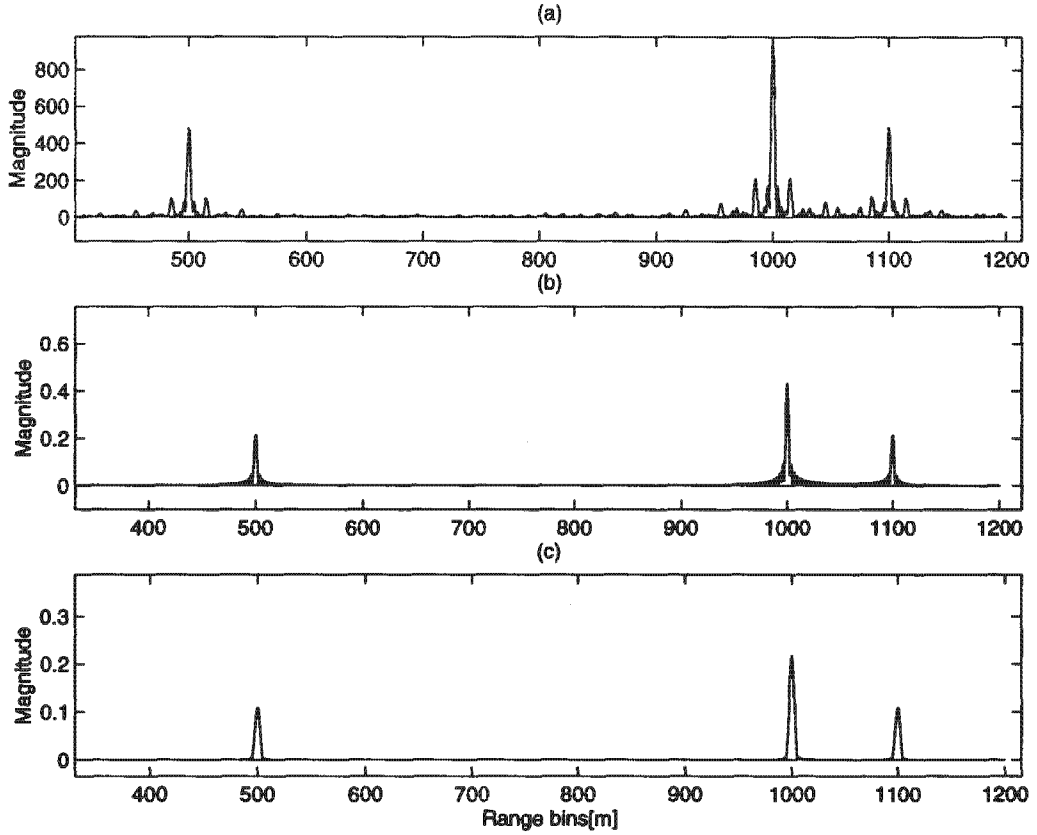

Figure 3.10: *Highresolution profiles of artificially simulated data after applying compensation filter and hanning window*

The above simulation results illustrates that the high resolution range profiles can be practically obtained by stepped-frequency processing. It is important that the compensation filter be suitably matched to the data in order to successfully flatten the reconstructed spectrum.

Chapter 4

Practical implementation on real data set

4.1 Introduction

In this chapter, two stepped-frequency processing implementation results are presented in order to validate the method. Section 4.5 shows the results of the simulation that was carried out using CSIR's X-band Roof-SAR parameters and the artificially generated scene used in the previous simulation. Section 4.6 presents the results obtained by processing the real radar data acquired by the CSIR's Roof SAR. Section 4.3 & 4.4 describe issues relating to the practical implementation of the code.

4.2 The Roof SAR radar parameters

Table 4.1 summarizes the CSIR's X-band Roof-SAR radar parameters which has centre frequency $f_0 = 141$MHz and transmits FM pulse with a pulse length of $T_p = 0.31\mu s$. It has a bandwidth of 15MHz sampled at 20 MHz. 161 pulses are transmitted each with 15 MHz bandwidth and spaced at 7.5 MHz intervals. The total bandwidth is given by $B_t = N\frac{B_{tx}}{2} = 1.2$GHz and from equ. 2.2 the range resolution is expected to improve from 10 m to 0.11 m.

Parameters	Symbol	Value
Start frequency	f_0	8.9 GHz
Step size	Δf	7.5 MHz
Pulse length	T_p	$0.31\mu s$
Chirp bandwidth	B_{chirp}	15 MHz
Number of steps	N	161
Sampling frequency	f_s	20 MHz
Number of samples	$N_{samples}$	200
Total chirp bandwidth	B_t	1.20 GHz

Table 4.1: *Stepped-frequency radar parameters obtained from CSIR's X-band Roof-SAR*

The range compressed data which was input to the reconstruction algorithm was simulated using these setup parameters.

The artificial scene consists of three point target and their position is stated in section 3.1.3.

The raw data were acquired from a grass land near the CSIR building. The antenna moved 100 m from the origin and collected the raw data by sending and receiving 1238 bursts of chirp pulses each having 161 frequency steps. The calibration data were obtained by receiving echos from a corner reflector placed at a distance 202.5 meters from the antenna. But in practical implementation, the calibrated point target was shifted to zero so that the target range in the scene remains the same.

4.3 The program flow

Range compression was achieved by convolving the received signal with a matched filter. The range compressed data file thus comprises of a two-dimensional array of complex values (with I as real part and Q as the imaginary part), with range as one dimension (fast-time) and the transmitted pulse number as the other dimension (slow-time).

The algorithm was tested for both the overlapping and splicing case. For splicing case, first the correct location of each sub-spectra was determined

and then the positive half of one spectrum and the negative half of the consequetive spectra was positioned in such a way that after addition they sit next to each other without any overlap or gap. This is done by chopping off the exact portion of the bandwidth required. It is important to ensure that there is not a single sample point overlap between the two sub-spectra because the phase at that point may not add properly leading to repeats in the high resolution profiles. Splicing is the simplest way to combine the sub-spectra, with some loss in SNR. Figure 6.1 in section 6.2 shows the flow-chart of the reconstruction algorithm used on real data. The flowchart is described in section 6.2.

4.4 Complex FFT routines

As FFT and IFFT routines are used in stepped-frequency processing it is important to understand how to use them. The Fast Fourier Transform is computationally efficient way to calculate the Discrete Fourier Transform (DFT). The DFT usually arises as an approximation to the *continuous Fourier transform* when functions are sampled at discrete intervals of space or time. By decomposing the set of data to be transformed into a series of smaller data sets, the FFT greatly reduces the amount of calculation required [16].

In Matlab, the real and imaginary part is stored in the same element. To apply the FFT operation to a complex matrix in Matlab, the row-column matrix is transposed and the Matlab FFT routine is applied to each column. Many good FFT implementations are available in C and microprocessor manufacturers generally provide free optimized FFT implementation in their processor's assembly code.

SuiTable FFT routines available in C are as follows:

1. FFTW-The "fastest Fourier Transform in the west"(www.FFTw.org).
2. MixFFT-Mixed-radix FFT by Jens Joergen Neilsen (http: //hjem.get2net.dk/jjn).
3. Gsl_complex_FFT (http: // www.gnu.org/software/gsl/).

The GSL subroutines are used in my C code to perform FFT. The GNU scientific library (GSL) is a collection of routines for numerical computing. The routines are written in C to presents a modern Application Programming interface (API) for C programmers and is a 'freeware.

The radix-2 complex FFT and mixed radix complex FFT are available in GNU scientific library. The inputs and outputs for complex FFT routines are **packed arrays** of floating point numbers. In a packed array the real and imaginary parts of each complex number are placed in alternating neighbouring elements.

The gsl_FFT_complex_radix2_forward(complex_packed_array_data,1,size) and gsl_FFT_complex_radix2_backward(complex_packed_array_data,1,size) routine were used to perform the one dimensional FFT and IFFT in the C code. These two require the length of the array to be a radix 2 number and also of the form *complex double*. All these routines are defined in the header gsl_FFT_complex.h and gsl_errno.h

Thus, prior to Fourier transforming each range line of the range compressed data was zero-padded to the nearest radix power of two in the time domain near the edges, and the data was cast as complex double.

4.5 Results obtained using the RoofSAR radar parameters and artificially generated scene

The simulation results are shown in Figure 4.1 to 4.4. All the graphs are plotted against their number of samples. First, the reconstruction algorithm was run for 3 sub-spectrum.

In Figure 4.1 graph(a) shows the three point targets return in time domain after range compression technique has been applied. Note that the $1/r^2$ factor which is proportional to the impulse response of the target has been excluded from the signal model. Graph 4.1 (b) shows the first sub-spectra. Note that, as the dispersion ($D = \triangle f T_p$) here is less than 60 the shape of the spectrum is not rectangular. In graph 4.1 (c) the calibration point target return is shown. The calibration pont target was positioned at zero. In the Figure

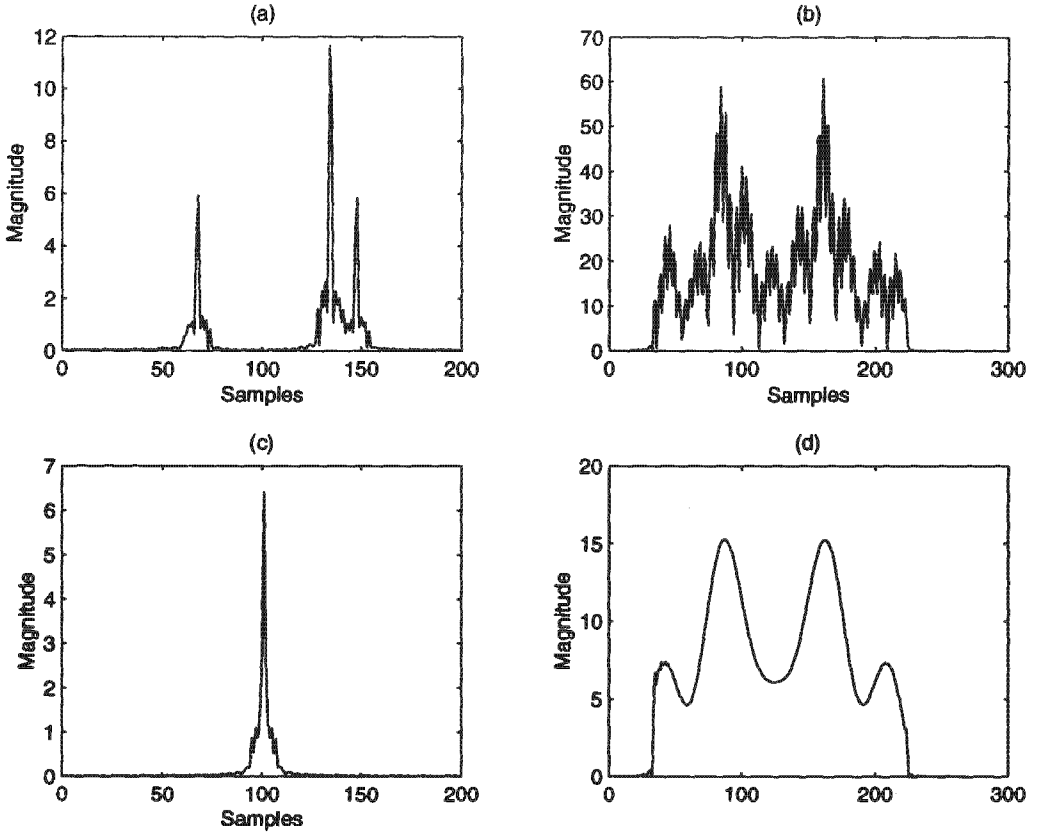

Figure 4.1: *The magnitude of time and frequency response of the artificially generated scene and calibration point target using Roof SAR parameters*

4.1 (c) the left and right portion of the impulse response has been swapped around and placed in the middle. Graph(d) shows a single spectrum of the compression filter.

Graph 4.2 (a) shows the reconstructed spectrum of three point targets obtained after 3 subspectrums were shifted in frequency and coherently added with 50% spectral overlap. The graph (b) in Figure 4.2 shows the combined spectrum obtained from the single point target returns. Graph 4.2 (c) and graph 4.2 (d) shows the compensation filter $H'(f)$ and the final spectrum after applying compensation filter and Hanning window respectively. The high resolution profile before applying the compensation filter and Hanning window is shown in graph 4.2 (e). Graph 4.2 (f) shows the final high resolu-

tion. Note that with the application of compensation filter and the Hanning window repeats appeared in Figure 4.2 (e) has disappeared in Figure 4.2 (f) and the sidelobes are also reduced.

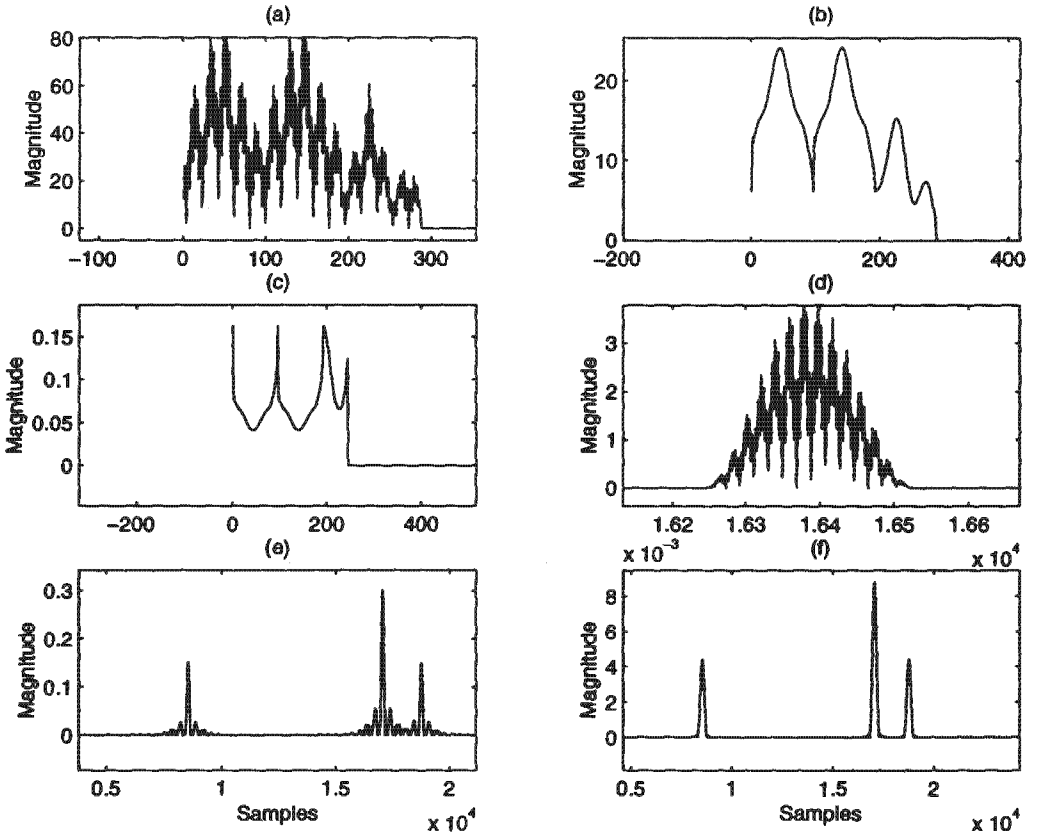

Figure 4.2: *The simulation results obtained for three point targets using the Roof SAR radar parameters (overlapping case).*

Figure 4.3 shows the simulation results when there is no overlap or gap between the two adjacent sub-spectra (splicing). The coherent addition of three sub-spectra can be seen in graph 4.3 (a). The sub-spectrums obtained from calibration point target was also added in graph 4.3 (b) using splicing method. Graph 4.2 (b) and 4.3 (b) shows the difference between adding and splicing. Note that in splicing there is no upwards or downwards spike in the combined spectrum. Graph 4.3 (c) shows the compensation filter and graph 4.3 (d) shows the final target reflectivity spectrum after compensation filter

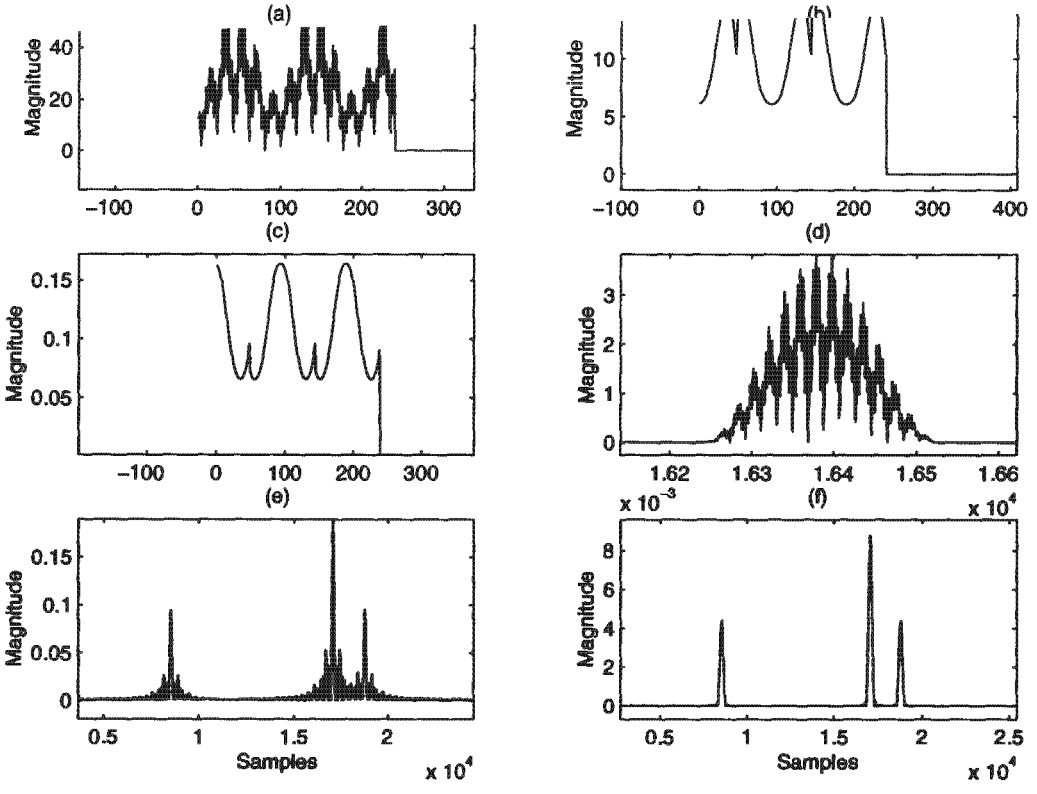

Figure 4.3: *Simulation results obtained for three point targets using the Roof SAR radar parameters (splicing case).*

and the Hanning window was applied. The high res. profile before and after applying the compensation filter and Hanning window is shown in Figure 4.3 (e) and 4.3 (f), respectively.

Figure 4.4 and 4.5 demonstrates the difference in highresolution profiles achieved before and after compensation filter and windowing function has been applied.

In Figure 4.4 graph(a) shows the high resolution profile obtained adding 3 subspectrums. No compensation has been applied here. We can see the repeating artifacts has appeared. Graph 4.4 (b) shows the high resolution after the compensation filter was applied. Note that in graph 4.4 (b) the

sidelobes have decayed faster as compared to graph 4.4 (a). Finally, graph 4.4 (c) shows the high resolution profile after being applied the Hanning window to the final spectrum. It can seen that in graph 4.4 (c) the sidelobes has been reduced with some increase in the mainlobe width.

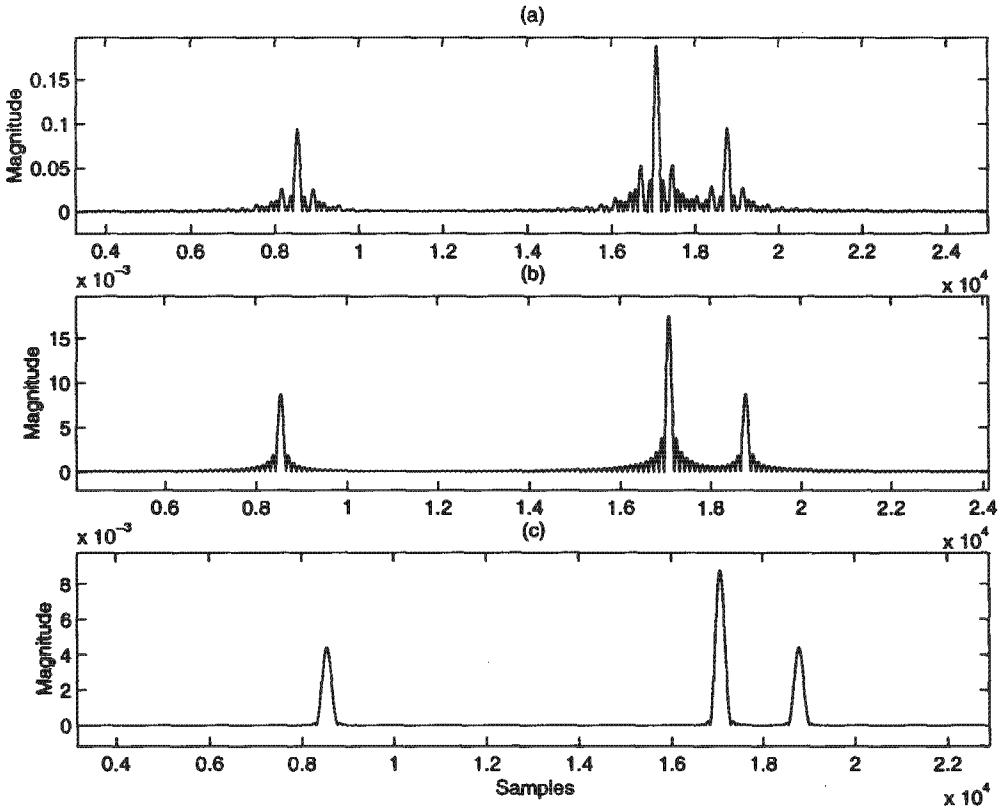

Figure 4.4: *The highresolution obtained using 3 sub-spectrums a) before applying compensation filter and hanning window b)after applying compensation c) after applying hanning window (splicing case).*

Figure graph 4.5 (a) shows the high resolution profile obtained after adding 30 subspectrums but without applying the compensation filter. Graph 4.5 (b) and 4.5 (c) shows the high resolution profile after applying the compensation and the Hanning window respectively.

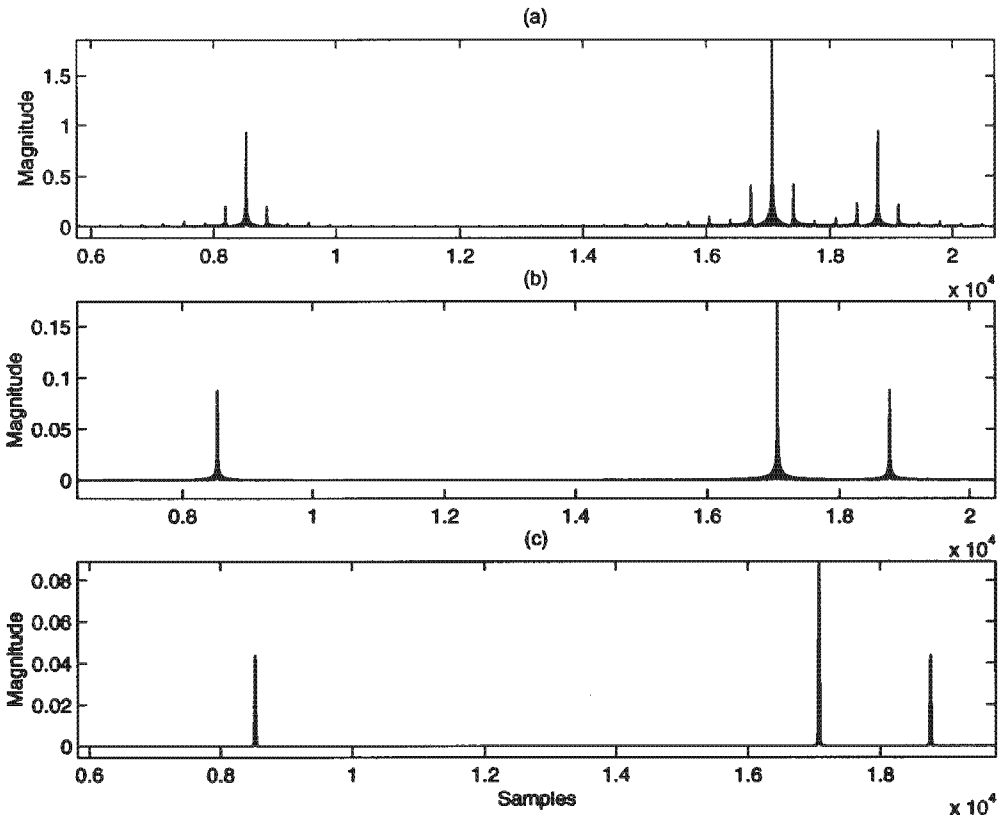

Figure 4.5: *The highresolution obtained using 30 sub-spectrums a) before applying compensation filter and hanning window b) after applying compensation c) after applying hanning window (splicing case).*

4.6 Results obtained using CSIR's RoofSAR range compressed data set

The stepped-frequency processing results of the real Roof SAR data are shown in figures 4.6 to 4.9 on the following pages. The Roof SAR parameters are discussed in section 4.2.

Figure 4.6 displays the following graphs:

(a) The range compressed target return after zero-padding. The spikes in the graph shows the bright targets in the scene.

(b) Here we can see a single spike i.e. the return from calibration point.

(c) First sub-spectrum obtained by Fourier transforming the target return in graph (a).

(d) First sub-spectrum obtained from the calibration point. Initially, the final spectrum was created by adding the subspectrums with overlaps and then compensation filter was applied to it. Figure 4.7 shows the simulation results for the overlapping case.

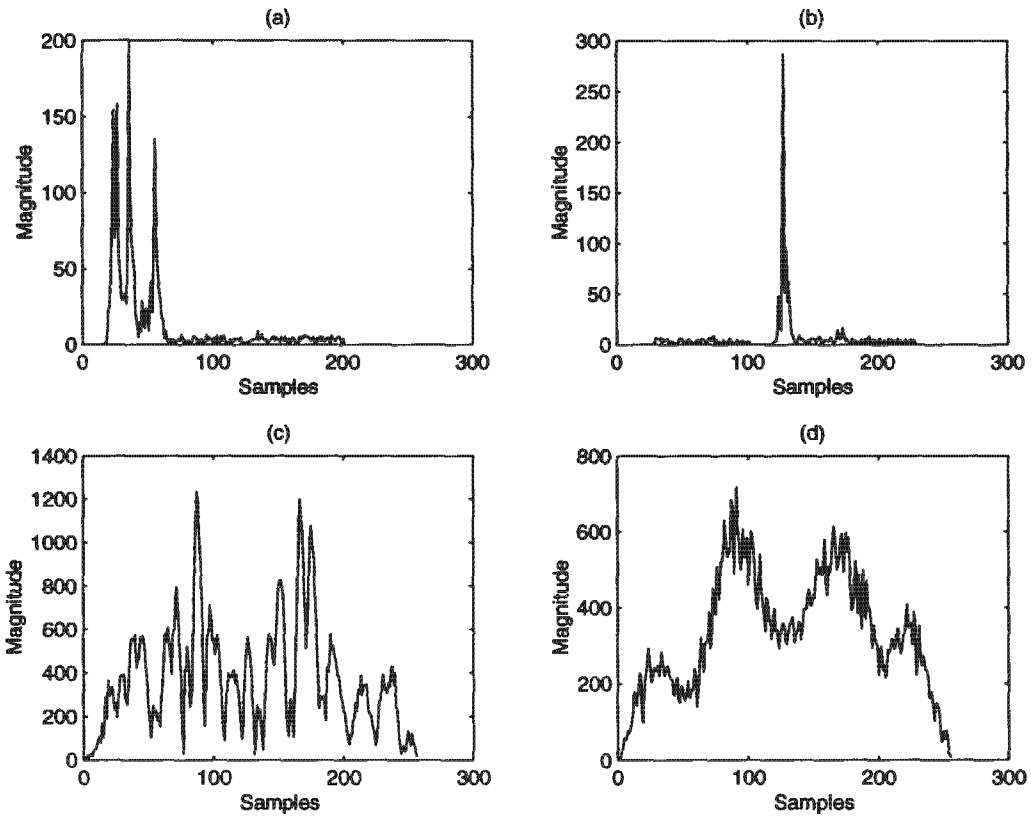

Figure 4.6: *The Magnitude of the range compressed radar return of single transmitted pulse in time and frequency domain as obtained from CSIR data sets.*

In Figure 4.7 (a) 30 subspectrums were added with overlaps. Figure 4.7(b) shows the combined spectrum from calibration point target and Figure (c) shows the compensation filter created by inverting the spectra in 4.7 (b).

The final target reflectivity spectrum after applying the compensation filter and a Hanning window is shown in Figure 4.7 (d). Figure 4.7(e) and 4.7 (f) shows the high resolution profile achieved before and after applying the compensation filter and window. From Figure 4.7 (f) it can seen that that high resolution profile has not been successfully achieved in overlapping case.

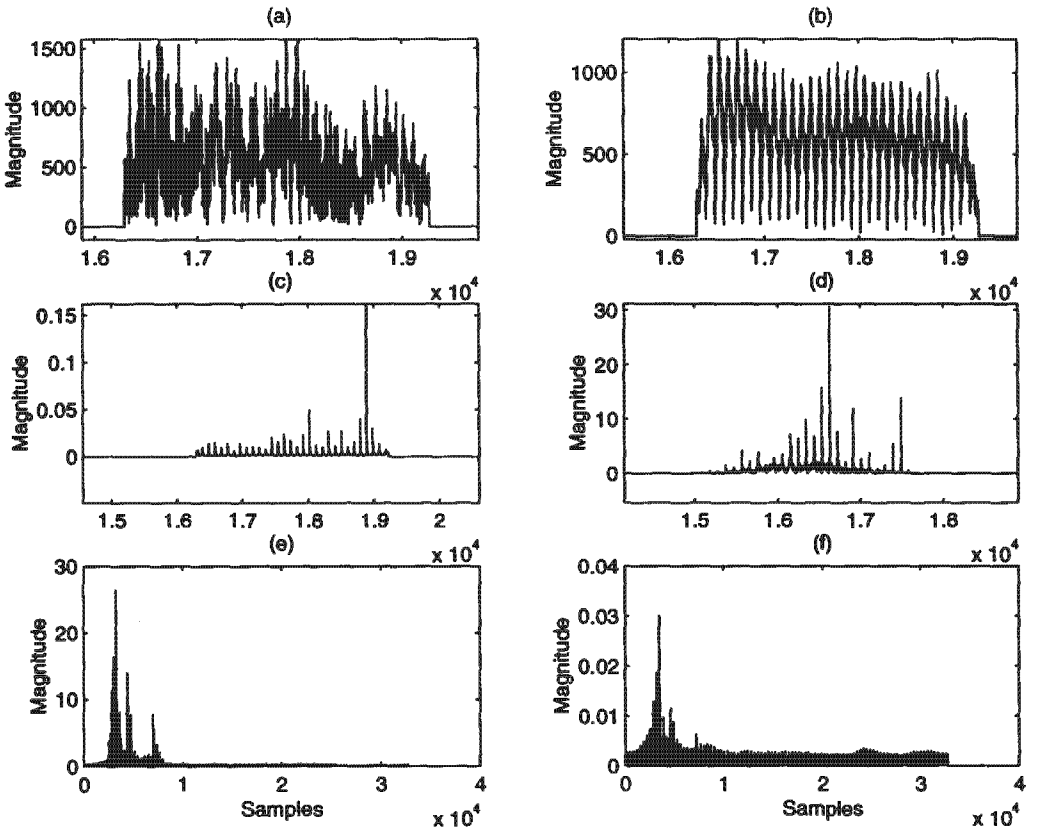

Figure 4.7: *Stepped-frequency processing results obtained after adding 30 sub-spectrums with overlap (using CSIR's Roof SAR data sets)*

As the overlapping of sub-spectrum didn't give us optimum results, I have carried out another reconstruction using the splicing method.

Figure 4.8 represents the following graphs:

(a) Three sub-spectrums are being added using splicing method after shifting them in frequency. The right half of the first spectra is not shown.

Care has been taken to ensure that there is no overlap or gap between two sub-spectra.

(b) The shifted and added sub-spectrums from calibration point.

(c) The compression filter obtained after inverting the reconstructed spectrum in graph (b).

(d) The target reflectivity spectrum after applying the compensation filter and the Hanning window.

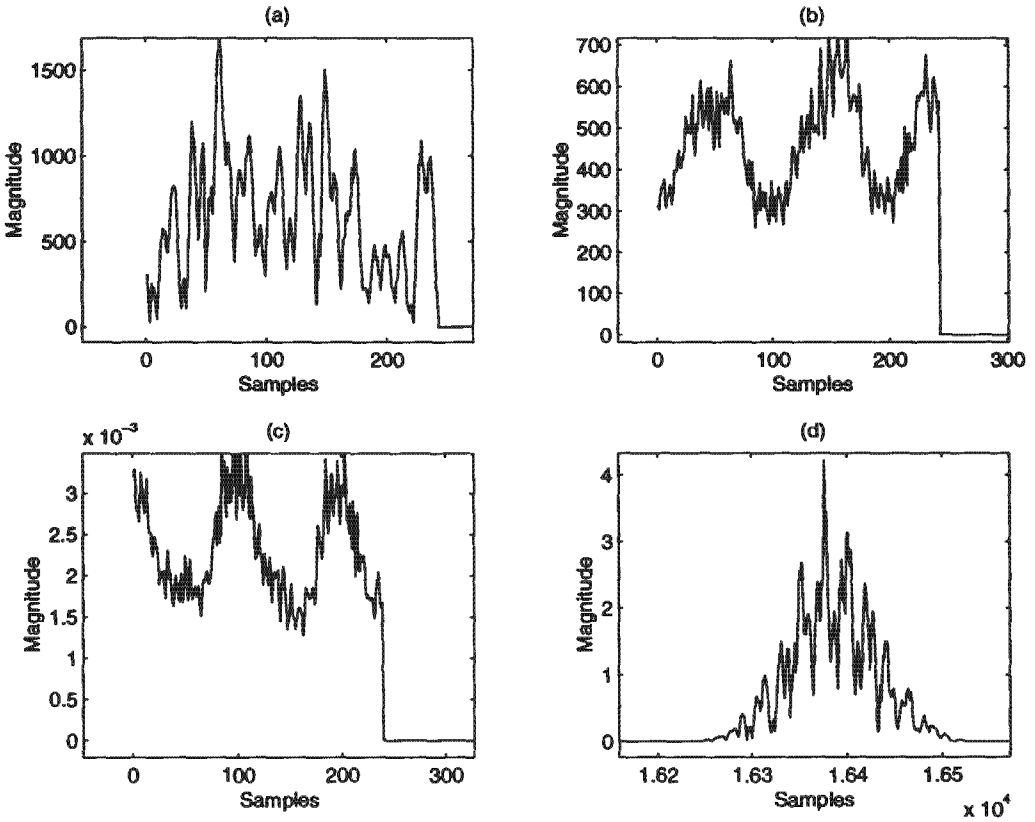

Figure 4.8: *The reconstructed spectrum after coherently adding 3 sub-spectrums (splicing case)*

Figure 4.9 (a) shows the high resolution profile that was obtained after 3 subspectrums were added. No compensation has been applied here. Graph 4.9 (b) shows the final high res. profile after applying the compensation filter and Hanning window. The side lobes are reduced here in compare to graph

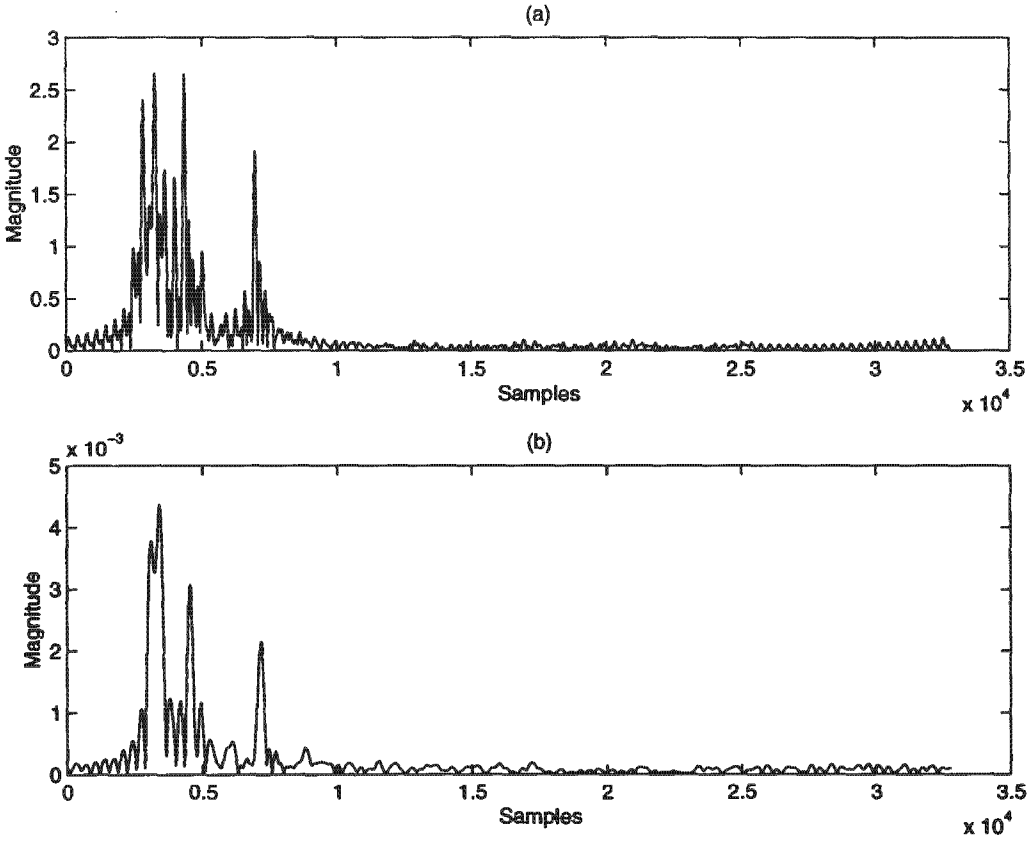

Figure 4.9: *The high resolution range profiles obtained (adding 9 subspectrums with splicing) a) before and b) after applying compensation filter and hanning window*

4.9 (a) and the broadening of mainlobe is visible.

Figure 4.10 (a) shows the high resolution profile achieved after adding 161 subspectrums but before applying the compensation filter. The final high resolution profile after applying compensation and Hanning window is shown in graph 4.10 (b). In graph 4.10 (b) it can be seen that there are few repeating artifacts has appeared in the final high resolution profile.

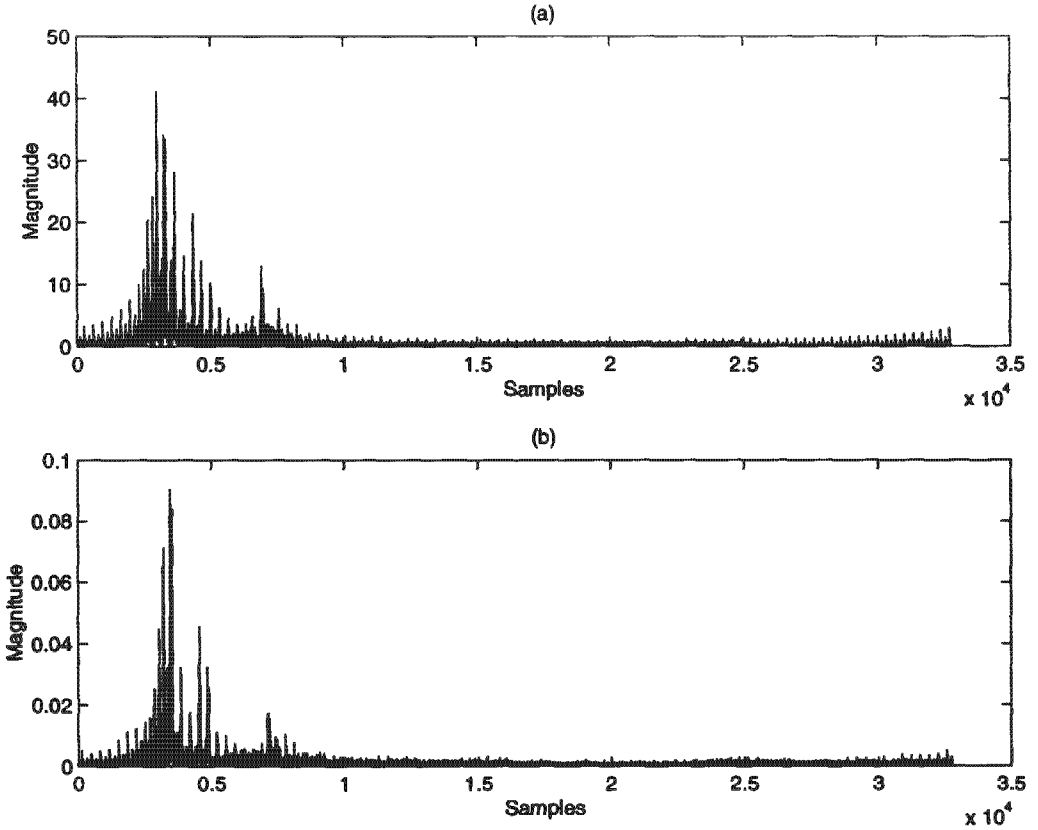

Figure 4.10: *The high resolution range profiles obtained (adding 161 sub-spectrums with splicing) a) before and b) after applying compensation filter and hanning window*

The graph 4.11 (a) at the bottom zooms the left portion of the entire profile in order to expose those repeating sidelobes.

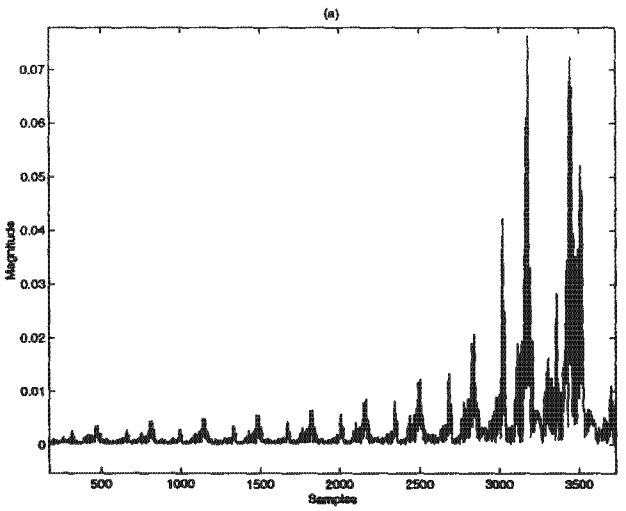

Figure 4.11: *Left portion of the high resolution profile obtained after processing 161 sub-spectrum using splicing method*

4.7 Discussions

The simulation results in section 4.5 and the application to the real data in section 4.6 demonstrate that the stepped-frequency processing can be practically applied on real data sets. The Roof SAR scene was "illuminated" with 161 chirp pulses stepped in frequency, each pulse having a bandwidth of 15 MHz, which corresponds to range resolution of 10 m. The use of stepped-frequency processing yielded a final range resolution of 0.11 m. In the simulation, where CSIR's Roof-SAR radar parameter was used, the sub-spectrums were added with or without any overlap to show that the high resolution range profile can be achieved in both without any repeating side-lobes (refer Figure 4.2 and 4.8). But the overlapping method was not quite successful in case of real data simulation.The splicing method, however successfully produces high resolution range profiles in real data except that few repeats appeared in the high resolution profile. One should be careful while positioning the narrow band pulses next to each other as overlapping of even one sample can cause repeats in final high resolution profiles.

Chapter 5

Overview of Parallel Programming

5.1 Introduction

A system based on single processor executes only one instruction at a time, and thus are limited by the performances of its hardware. The fastest sequential supercomputers presently operate within an order of magnitude of their theoretical maximum speed, which is on the order of 3 Giga flops (3×10^9floating point operations per second). Real time SAR processing systems often require tens or hundreds of Giga flops of processing power and Gigabytes (GB) of memory for storing successive frames of data to be processed [26, 27].

Distributed computing is a process where a set of computers connected through a network are used to solve a single large problem. As more and more organizations have high-speed local area networks interconnecting many general purpose workstations, the combined computation resources may exceed the power of a single high-performance computer.

Parallel computers are classified by the relationship of the memory to the processors and the number of instruction streams available to the system. This chapter provides a brief description of models of parallel computers, their memory architecture and the interconnection networks in the cluster of

computers. Furthermore, it explains the parallel processing theory, decomposition strategy and the parallel processing software tools based on which the parallel algorithm has been developed and implemented.

5.2 Architecture Taxonomy

Fynn in 1966 classified all the parallel computers according to the number of instruction streams and the data streams they can have. According to his taxonomy there are four categories namely *Single Instruction -Single Data stream* (SISD), *Single Instruction stream, Multiple Data stream* (SIMD), *Multiple Instruction Multiple Data stream* (MIMD) and *Multiple Instruction Single Data stream* (MISD) [28]. One of the problems in Flynn's taxonomy is that there is no classification found for MISD.

5.2.1 SISD

SISD -This model is a sequential computer model where by a set of instructions is executed sequentially on a data stream. It is limited by the number of instruction that can be issued in a given unit of time.

5.2.2 SIMD

SIMD - refers to a parallel execution model in which all processors execute the same operation at the same time, but operate upon their own data. This model naturally fits the concept of performing the same operation on every element of an array, and is thus often associated with vector or array manipulation. Because all operations are inherently synchronized, interaction among SIMD processors tend to be easily and efficiently implemented. Only distributed memory is used in this type of machine. Examples of this types of computers are Cray1, MPP, CM-2, MasPar MP-1 etc [29].

5.2.3 MIMD

MIMD - a category of Flynn's taxonomy in which many instruction streams are concurrently applied to multiple data sets. Each processor can perform any operation regardless of what other processors are doing. For example, one processor might update a data base file while another processor generates a graphic display of the new entry. This is more flexible than SIMD execution, but can be difficult to program.

MIMD systems require synchronization to be programmed in. Examples of MIMD are nCUBE, Intel iPSC,FX-8, TC-2000 [28]. MIMD systems have been split according to their memory configuration, namely shared and distributed memory.

Shared memory

Systems like multiprocessor Pentium machines running Linux physically share a single memory among their processors. Parallel computers of this type generally use a common system bus or a switching network to gain direct access to memory. While frequently good for multiple users whose individual programs are not large or complex, the bus-based system can be a bottleneck that limits the speed at which processors can access memory [30].

The shared memory systems can be divided into following categories [26].

1. **SMP** (*Symmetrical multiprocessing*) - In this system, a group of processors access the memory equally at equal speed.

2. **NUMA** (*Non-uniform memory access*) - The memory is physically shared but access to different portions of memory may require significantly different amount of time.

3. **DSM** (*Distributed shared memory access*) - The memory is distributed among the processor but it gives an illusion that the memory is shared.

Distributed memory

Systems of this kind are made up of set of nodes, each of which consists of a main processors, memory and interface to network. Data is shared across

communications network as a message using *message passing.*

Message passing is a model for interactions between processors within a parallel system. In general, a message is constructed by software on one processor and is sent through another processor, which must accept and act upon the message contents. Message passing can yield high *bandwidth* (number of bits/sec) making it very effective way to transmit a large block of data from one node to another [27].

The following are the advantages of using a distributed memory system.

- It is flexible. Nodes with different types of processor can be used to adapt the system to specialized problems.
- Memory is scalable to number of processors. It is possible to start with a smaller system and expand the system as the computing requirements grow without the addition of different or complex hardware to support the expansion [27].
- It is cost-effective. Some parallel systems, such as the iPSC systems, use readily-available processors to reduce the cost of the systems without putting the enormous development costs of designing a processor for that system only [28].

Examples of these types of architecture are Intel Paragon, IBM SP1, nCUBE Hypercube etc.

Distributed memory systems can be divided into following three categories[26]:

Fixed: The number of connections is fixed as more processors are added(e. g. Ethernet connected workstations).

Linear: The number of connections grows linearly with the number of nodes (e.g. Intel Paragon).

Scalable: number of connections grows as P log P or greater (e.g.hypercubes such as Intel iPSC/860), where P is the number of processor [30].

The single program multiple data **(SPMD)** structure can be viewed as an extension of SIMD or as a restriction of MIMD in which all processors

execute the same code independently using many instances of a single type of process.

5.3 Network criteria

The following network related terminology plays significant role in determining the performance of a parallel system [29].

- Bandwidth - the number of bits that can be transmitted in unit time, measured as baud or bits/second.
- Communication latency - the total time taken to transmit one object, including any send and receive software overhead. Latency determines the minimum run time for a segment of code to yield speed-up through parallel execution [32].
- Diameter - the minimum number of links between the two farthest nodes in the network.
- Hardware cost - Indicated by the number of links in the network. It is important that the cost to install communication networks is less compared to the connection of the whole system.

5.4 Parallel processing software

Parallel processing software or the messaging middleware is the software layer that acts as a common communications interface between software applications in a parallel system. The message passing models allows processes to communicate either directly with one another or through a coordinating daemon on each node. The software technologies that have been used in RRSG's Gollach cluster are

- *Parallel Virtual Machine* (PVM)[31]
- *Massage Passing Interface* (MPI) [34]

- MOSIX (http:// www.mosix.org)

Since PVM was used as a message passing middle-ware in the parallel algorithm used in this project, the next sub-section gives an overview of PVM. MOSIX is an addition to the Linux kernel for transperant process migration and resource sharing in clusters.

The comparative features of PVM and MPI are summarized in sub-section 5.4.2

5.4.1 PVM

PVM is public domain software package developed at Oakridge National Laboratory [33]. PVM operates on a collection of heterogeneous Unix computers connected by one or more networks, which build the *virtual machine.* It is comprised of two components: the PVM daemon, called *PVMd3* that runs on each machine and a set of PVM library routines that are needed for cooperation between tasks.

PVM daemon controls the task spawning, performs inter-task communication and data conversion. The library contains the user-callable routines for message passing, processes spawning and tasks coordination. [33].

Figure 5.1 shows the message passing model of PVM where the copy of information is passed through the network from node A to node B using send and receive call.

PVM message passing routines are listed below:

PVM_initsend() clears the send buffer and is called before packing a new message. PVM send routines are nonblocking **PVM_send()**, while receive routines can either be blocking **PVM_recv()** or non-blocking **PVM_nrecv()**. Multicasting is done by **PVM_mcast()** call.

The message should be unpacked according to the format in which it was packed. **PVM_pk*()** is used to pack data and **PVM_upk*()** is used to unpack data in order [31].

PVM is dynamic in nature. Computing resources, or "hosts," can be added or deleted either from a system "console" or from the user's application.

5.4.2 Comparison between PVM and MPI

The widespread use of MPI in industries has caused programmers to think that if they should change their code developed by PVM to MPI standard. This section summarizes the pros and cons of each application.

If an application is going to be developed and executed on a single MPP, the MPI has the advantage of expected higher communication performance. Because of having a much higher set of communication functions, MPI is favoured when an application is structured for special communication modes not available in PVM [33]. The two major drawbacks of MPI are 1) lack of inter-portability between any of the MPI implementation, i.e., one vendor's MPI cannot send a message to another vendor's MPI and 2) lack of ability to write fault tolerant applications [34]. If a task fails in MPI, the result remains undefined and usually the system shutdowns. The only thing guaranteed is the ability to exit the program after a MPI error.

PVM is preferred when the application is going to run over networked collection of heterogeneous hosts. PVM contains resource management and process control functions that are important for creating portable applications that run on clusters of workstations and MPP [34].

PVM's fault tolerance feature is quite significant. The ability to write long running PVM applications that can continue even when hosts or tasks fail, is quite important to heterogeneous distributed computing [33]. Some comparative results can be found at http://rrsg.ee.uct.ac.za/mti/results.html.

5.5 Performance of a parallel system

The most commonly used terms to describe the performance of multiprocessor systems are speed-up and efficiency.

5.5.1 Speedup

Speedup factor is a measure that compares a parallel solution with a sequential one. The speedup is given by the equ. 5.1

$$S = \frac{T_s}{T_n} \tag{5.1}$$

where is T_s execution time of the sequential code and T_n is the time the parallel code takes to run on n nodes.

If the fraction of the computation involved in the serial section is f, and no overhead incurs when the computation is divided into concurrent parts, then according to ***Amdahl's law*** the theoretically achievable speedup S for a parallel version of an algorithm is given by equ. 5.2 [29].

$$S \leq \frac{n}{1 + (n-1)f} \tag{5.2}$$

According to Amdahl's law the speedup is limited by the fraction of the algorithm that cannot be parallelized irrespective of the number of processors used in the parallel system. i.e.

$$\lim_{n \to \infty} S(n) = \frac{1}{f} \tag{5.3}$$

5.5.2 Efficiency

Efficiency is a measure of hardware utilization, equal to the ratio of speedup achieved on nth processor to total number of processors (n) used in the parallel system [27]. It is defined as

$$E = \frac{S}{n} \tag{5.4}$$

5.5.3 Granularity

The ratio between computation and network communication is known as task granularity. Scale of granularity ranges from fine-grained (very little computation per communication-byte) to coarse-grained (extensive computation per communication-byte).

The finer the granularity, the greater the limitation on speedup, due to the amount of synchronization needed [14].

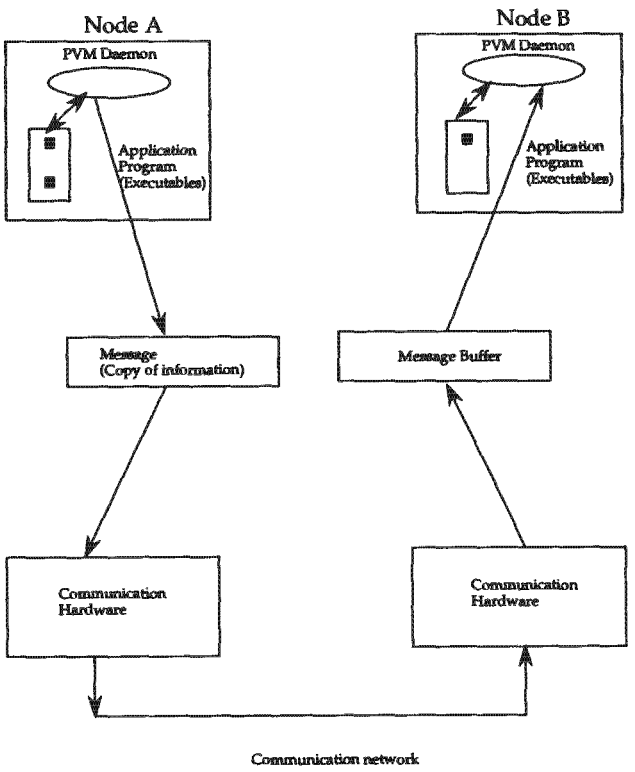

Figure 5.1: *Message passing model in PVM*

Chapter 6

Parallel Implementation applied to simulated and real data sets

6.1 Introduction

The master-worker model [31] is a standard approach that can be used to implement parallel programs. This chapter describes how the parallel algorithm was developed from the serial code. It also describes in detail the control of data flow between the master and worker process under PVM environment. The results and timing analysis are included in the next chapter.

6.2 The sequential algorithm

The serial C program that produces N number of high resolution profiles was run using the artificially simulated data (small data set) and the RoofSAR data (larger data set).

Figure 6.1 shows the flow-chart of the serial version of the stepped-frequency processing algorithm.

The max_range_lines refers to the maximum number of high resolution profiles to process. In the flow-chart in Figure 6.1, there are two loops. The first loop is used to make the compensation filter. First, the range compressed calibrated data is loaded once in complex float form as were

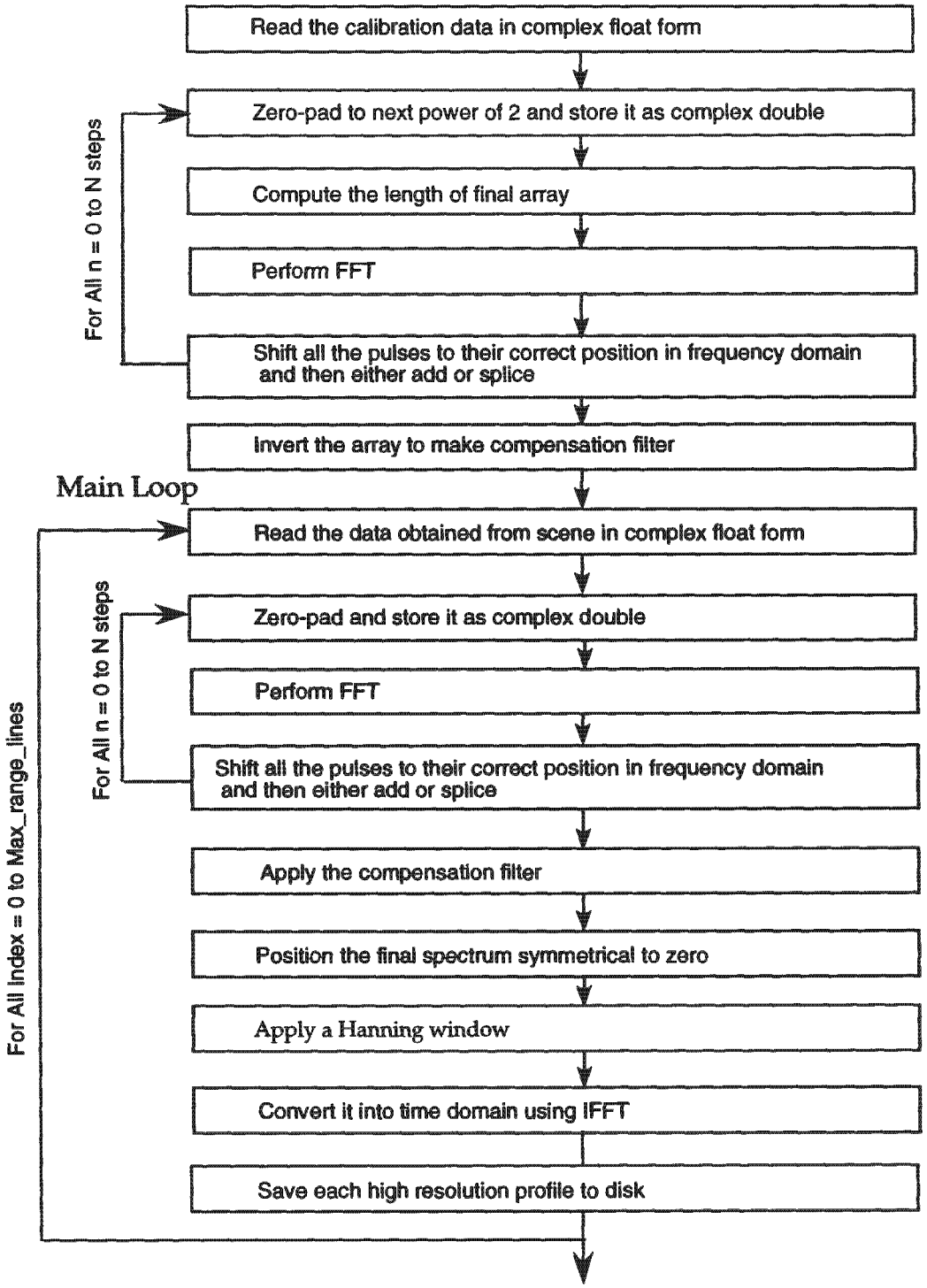

Figure 6.1: *Flow-chart of serial version of stepped-frequency algorithm*

acquired from CSIR. Then, it was zeropadded and was saved in the form complex double. The reconstructed wide band spectrum is inverted to make the compensation filter. In the main loop, first the data from the scene were read in the same way as for the calibrated data. After performing FFT on that data, the subspectrums were shifted to their correct position and were added in a long array. The reconstructed wide band spectra is multiplied with the compensation filter to flatten the amplitude ripples. Then, the final spectrum is positioned symmetrical to zero and a Hanning window is applied to it. Finally, the high resolution profile is achieved by performing IFFT to the final spectrum. The main loop continues after saving each high resolution profile to disk.

Due to processors memory constraints, the data were read from the hard disk by N_steps (maximum N number of steps) to an input array each time it were required. Recall from section 4.4 that the gsl complex FFT routine used requires an input array of type double and of radix 2 number. As I am zero padding the array and storing the data in complex double format, it is important to free the memory of some arrays subsequently which were allocated within the main loop and are not required for further computation. Otherwise, it gives a memory allocation error as data gets accumulated after few iteration. Also, while running the serial code it is better to halt or delete other application programs which are taking significant amount of memory.

The serial code applied on artificially simulated data was run to generate 120 range lines or high resolution profiles each with 100 steps. The range compressed real data from the CSIR required production of 1238 range lines each with 161 steps.

6.3 Designing the Parallel Program

"Parallelism is a processing task that must be first explored before the task can be parallelized. It is a property of the algorithms that allows different parts of the task to be done concurrently and the correctness of the result is

maintained"[28]. The most indispensable and neccesary condition for parallelizeable operation is that the operations are local. A local operation does not require the result of other operation.

Below listed are few basic steps based on which the parallel code was developed [29]:

1. **Debugging the serial code completely** - It is sequential important to ensure that there isn't any error in computation.

2. **Identifying the parts of the program that can be executed concurrently.**

3. **Decomposing the program** - There are two ways to split up a large problem to smaller multiple tasks, namely 1) *Functional Parallelism* (Algorithmic partitioning): A node or set of nodes is assigned to each step in a process, and tasks are put through one after another. In purely algorithmic partitioning, the number of processor that one can add is the maximum stages that the application has. 2) *Data Parallelism:* Here all the processors run the same application, but on different sets of data elements. The total processing time brought down by T/N where T is the time taken on single processor and N is total number of processors deployed in parallel to do the same task [27]. This kind of parallism is very much used in almost all image processing application. In some applications, combination of both the parallelism is required to solve a problem.

4. **Code development**: Code may be influenced by machine architecture. It is vitally important that sufficient RAM memory is available for a given application. If the the memory is not enough for a specific algorithm, disk swapping is required for virtual memory access [14].

5. **Choosing a programming paradigm:** Two most common programming paradigms in PVM application are *master-worker* and *node-only* (tree computation) model. In the *master-worker* model, a control process termed as master is responsible for process spawning, initialization, collection and display of results. The worker program performs

the actual computation involved by either allocating their workloads by master or by themselves. In *node-only* model multiple instances of single program execute, with one process taking over the non computational responsibilities in addition to contributing to the computation itself [31, 33].

6. **Establishing the communication** - There are several communication software tools such as PVM, MPI, MOSIX that are available to effectively exchange data between processors. These communication software has their own library function which are used to establish synchronization between the nodes.

7. **Debugging** - While debugging, careful attention should be given in both communication and computation details. The gcc compiler allows us to compile the task under gdb debugger. In PVM, PVMTaskDebug option is set to spawn routine in order to start the task in a debugger [32].

8. **Measuring performance** - The last step is to compare the timing results of the parallel code with the serial code and optimize the parallel code to increase the performance.

6.4 The Parallel algorithm

The parallel version of SFP algorithm can be divided into three parts. Reading the calibrated data and the data obtained from scene has been done in sequential way, processing N high resolution profiles using stepped-frequency method has been done in parallel and writing each processed high resolution profile to disk is again a sequential process. This section describes in detail exactly how the data were exchanged between the master and worker using the message passing software PVM. The description of the cluster which was used to run the parallel application is given in the next chapter. 5 nodes or PCs were used to run the parallel code in which the master node had dual CPU.

6.4.1 Implementation of Master process

Initialization phase:

Master process at first reads the range compressed calibrated data which is needed to make the compensation filter. It then computes maximum length of the final array and performs mallocs or callocs for all the arrays to be used subsequently. This phase doesn't require any interaction with PVM environment. Master initializes itself by an *integer task identifier* (TID) supplied by local PVM daemon. The routine used here is **PVM_mytid.**

PVM_setopt(PVMRoute, PVMRouteDirect) function is set at the beginning of each task to enable direct route communication between PVM tasks without interacting with PVM daemon. The communication *bandwidth* increases over a network by using this option. The master process then spawns all the workers using **rc = PVM_spawn("worker", NULL, PVMTaskDefault, " ", nproc, tids)** routine. The first variable is the execuTable filename of the PVM process to be started. The second argument in PVM_spawn is set to NULL. PVMTaskDefault flag has been setup so that PVM can choose any machine to start. In order to start the task through a debugger script, **PVMTaskDebug** flag is sometimes added. The next argument specifies the host name or a PVM architecture class and PVM selects it by itself. The last two arguments are the number of copies of the executable to start up a task and TIDS of the spawned process. In case of a system error, rc returns a negative number.

If there are 5 nodes available and 5 workers need to be spawned the master process will spawn the 4 tasks on less loaded nodes and the last task on the same node where master process is running.

Once the workers have been spawned the master process decides how to distribute the workload evenly among the workers. A program executes most quickly when every processor has an equal share of the total amount of work to perform. This is called perfectly *load balancing.* As the system where I have implemented my parallel code is a homogeneous cluster of computers, data partitioning approach has been suitably applied here. Data partitioning can be done either *statically* i.e., self scheduling where each worker process

knows a priori of its share of workload or *dynamically* where a master process allocates subunits of the workload to worker process when they become free [31, 32].

After spawning the workers, the master transfers an initial message to each worker process which contains the information about the number of processor cycles. For example, in order to process 1238 range lines using 5 nodes each node should go through a loop $(1238 - 3)/5 = 247$ times and the remainder 3 profiles should be sent to 3 worker process after the master process has received all (1238-3) processed high resolution profiles. PVM multicast option can transfer a message stored in the active send buffer to n number of tasks specified by the **tids** in the array. The code is flexible enough to process any number of range lines using any number of nodes.

6.4.2 Scattering the range compressed data to workers

At this stage the master process has dynamically allocated all the memory required. The next phase is to pass the calibration data that needs to be processed by the worker process. The calibrated data is transfered only once and each worker should have copy of it each time it spawns. The routine PVM_initsend clears the send buffer and prepares it for packing new message. The data is packed using **PVM_pk*** routine as integer and float format and is multicasted using **PVM_mcast(TIDs, nproc, FROMMASTER_MSG)**.

Table 6.1 gives an example of the parameters that are passed in the multicast message.

Parameter	Explanation
TIDs	Number of worker processor
nproc	Worker Task ID's
FROMMASTER_MSG	Message identifier
Array Size	Size of the input buffer to store the elements of an array
Array	The elements of the array in float format

Table 6.1: *Example of parameters sent in multicast message.*

After sending the calibrated data, Master process sends the data obtained from scene evenly one after another to the 5 worker processes in a loop using PVM_send(TIDs, FROMMASTER_MSG) function. The data is packed into PVM send buffer prior to sending. This message is Asynchronous or *nonblocking* i.e., computation of the sending process resumes right after the message is safely on it's way to the receiving process [31].

Receiving processed high resolution profiles from workers: Master process receives the processed high resolution profiles in the same sequence as it was sent using a synchronous blocking routine **bufid = PVM_recv(TIDs, FROMWORKER_MSG)** and saves it in a file on disk. A synchronous or *blocking* process waits until a message matching the user specified task identifier (tid) and message tag (msgtag) value arrives at the local PVMd [31]. If PVM_recv is successful, bufid will be the value of new active receive buffer. In case of any error, it gives a negative value. The master process saves each high resolution profile to disk.

The remaining profiles are sent to worker process after master process has received all previous high resolution profiles. The worker process receives the next profile and continues processing only if there are more profiles left to process otherwise it receives a stop status with a value zero.

6.4.3 Implementation of worker process

In order to receive and send data to master process the worker process needs to assign its unique TID value and needs to find the master process TID value. Each node keeps a local copy of the executable for the worker process. PVM_setopt (PVMRoute, PVMRouteDirect) routine is used again to reduce the message overhead by passing data directly to the tasks. The worker process is responsible for doing all the computation required to process the high resoluion profiles. Each worker process has a copy of setup parameters. It receives and unpacks the receive message buffer containing calibrated data exactly as the same format as it was packed using **PVM_unpk*** function. Then it zeropadd the data and casts it as double and makes the compensation filter.

In the same way it receives the data obtained from scene in a loop, processes the high resolution profiles and sends them back to master. After that the worker process either receives the remaining profiles to process or receives a stop status. Lastly, it hand overs the remaining processed high res. profiles and exits from the PVM environment. Note that here the worker processors does not communicate or transfer data between each other. This kind of application where there is no communication between the worker nodes falls into the category of *perfectly parallel.* The sequence of communication between the master and worker process has been summarized in Figure 6.2.

Assumptions:

There are Sn number of worker process available
Master and worker processor communicate via Index variable

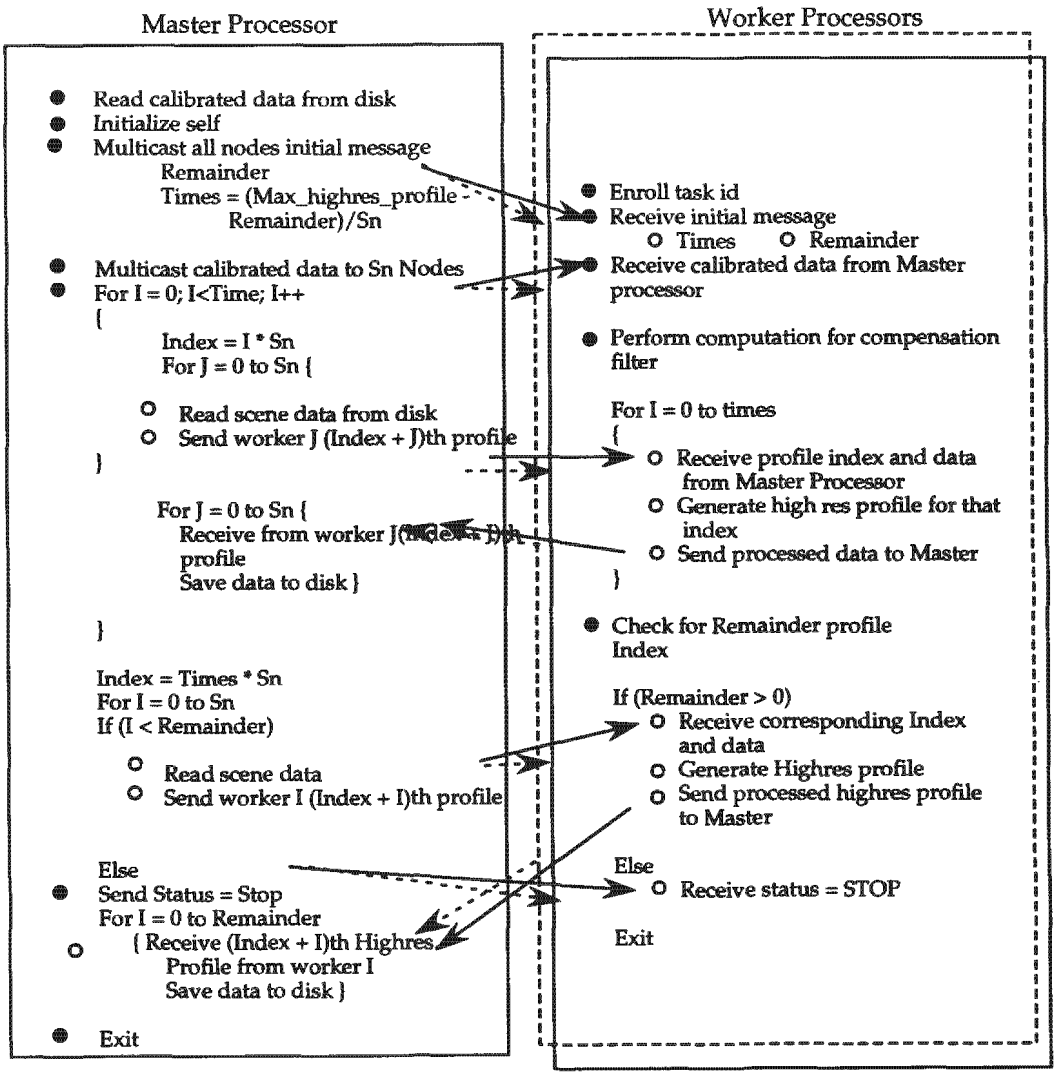

Figure 6.2: *The sequence of communication between the master and worker process*

Chapter 7

Results and Timing Analysis

The parallel algorithm that has been adopted here is exactly the same for the simulated and real data set. In this chapter, the timing results and the performance analysis of the parallel code compared to the serial code are given. Two data sets are analysed:

1) A small data set (obtained by simulation) is described in section 7.1.

2) A larger data set (obtained from the CSIR's RoofSAR) is described section 7.2.

More closer analysis has been done on the real (RoofSAR) data set.

7.1 Timing results for artificially simulated data set

The serial and the parallel code were first tested and run on a cluster using the artificially simulated data set. This section presents the description of that cluster and the timing results obtained by the serial and parallel code that uses the simulated data set. The input data for the serial and the parallel code was of size 98 MB (120 range lines×100 low resolution profiles×4 byte×2×1024 samples per low resolution profile) and the output data size was 7.8 MB (120 range lines×8 byte×2×4096 samples per high resolution profile) .

The parallel cluster *gollach* which has been used to verify and test the

parallel algorithm can run code developed using PVM, MPI and has Linux kernel module. The 8 node cluster has 1 Dual-CPU Intel Pentium II, 350 MHz machine with 512 M bytes of RAM where the Master process runs and 7 single-CPU Intel Pentium II 350 MHz machines having 256 MBytes of RAM which are used to spawn the worker process. All Nodes are inter-connected with each other by a 100Mb/s full duplex CISCO/CS21912 12 port fast Ethernet switch. The PVM message is transmitted over the underlying network hardware using Unix Datagram Protocol (UDP) [14]. Figure 7.1 shows the picture of the RRSG's *gollach* cluster. Only 5 nodes were operational at the time of this work.

Figure 7.1: *The RRSG's gollach cluster (front view)*

7.1.1 Timing result of the serial code

The Table 7.1 shows the performance of the parallel version of the stepped-frequency algorithm (listed Appendix A.2) in comparison to the serial version. The time was measured using the GNU scientific Library function **sys/time.h** which takes time from a system clock. The time was recorded 4 times and the average of those readings are displayed in Table 7.1 and 7.2. To process all 120 range lines, the serial code took 122 seconds (i.e. approx. 1 second per line) on a 350 MHz Pentium node in gollach cluster.

7.1.2 Timing results of the parallel code

The parallel code was tested by spawning the master and one worker process on the dual-CPU Pentium in gollach cluster. The other 4 workers were spawn separately on 4 single-CPU Pentium. From Table 7.1 it can be seen that the maximum speedup achieved with 5 nodes is $S = 2.4$ which corresponds to an 5-node efficiency of $E = 2.4/5 = 0.48$. The speedup of $S = 3$ is regarded as the standard measure of speedup for 5 nodes which corresponds to 60% of efficiency.

Processor	Time(sec)	Speedup
Serial code (timed)	122	
Theoretical performance for 5 node (100% efficiency)	24.4	5
Theoretical performance for 5 node (60% efficiency)	40.66	3
Parallel code-5 node (timed)	50	2.4

Table 7.1: *comparison of serial to parallel code (for simulated data set)*

As one of the node among 5 nodes has dual CPU where both the master and one worker were spawn, the total number of CPU becomes 6. Therefore, in this particular case, the efficiency can either be defined as node efficiency as the total number of nodes in the parallel system is 5 or CPU efficiency as the total number of CPU used here is 6. Figure 7.2 shows a diagram of the gollach cluster where 5 nodes are connected by switched Ethernet.

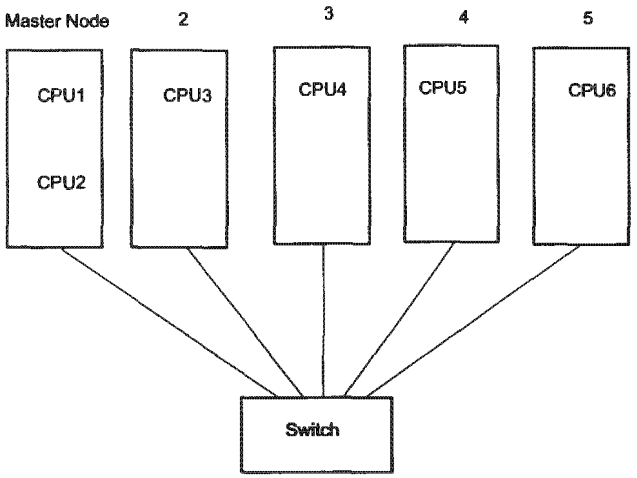

Figure 7.2: *The diagram of gollach cluster (showing the nodes connected via communication network)*

The recorded execution time for 1 to 5 nodes to process 120 range lines and the corresponding speedup and efficiency (Node and CPU) has been shown in Table 7.2. The speedup and efficiency has been calculated using equ. 5.1 and 5.4. Note that the time to write each processed high resolution profile of size 0.6 MB (8 byte×2 ×4096 sample points) to disk which is a sequential process was also included within the timing results of the parallel code.

Nodes(n)	Time spent(T_p)	Speedup(S)	Node eficiency(E_1)	CPU efficiency(E_2)
1	175 sec	0.69	0.69	0.34
2	145 sec	0.84	0.42	0.28
3	101 sec	1.20	0.40	0.30
4	78 sec	1.5	0.37	0.30
5	50 sec	2.4	0.48	0.40

Table 7.2: *Changing the number of nodes from 1 to 5 with constant number of range lines (for simulated data)*

Figure 7.3 shows three graphs. The Speedup and node efficiency graph is plotted against the left Y axis and CPU efficiency graph is plotted against right Y axis. The speedup graph shows almost linear scaling. Figure 7.3 shows that Node Efficiency achieved is higher than CPU efficiency.

speedup graph shows almost linear scaling. Figure 7.3 shows that efficiency achieved with 5 node cluster is higher than 6-CPU efficiency.

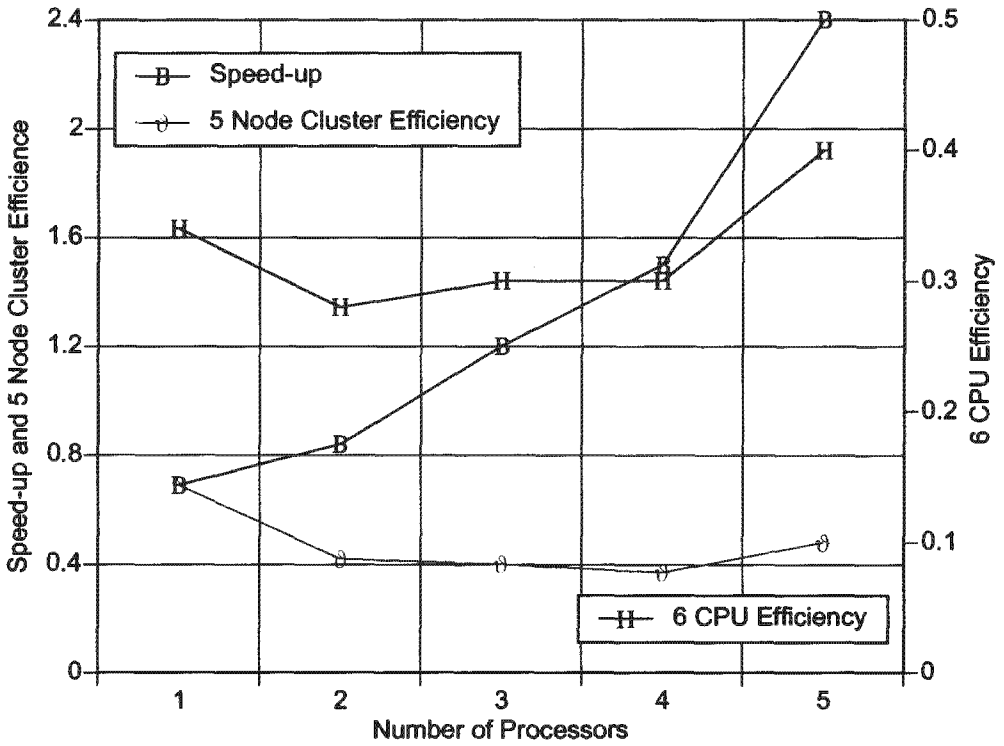

Figure 7.3: *The Speed-up and efficiency graph for simulated data set*

7.2 Timing analysis for real data set

After having implemented and tested the parallel algorithm on the simulated data set, it was tested and run in Gollach cluster using the real data set. The problem size of real data i.e. number of range lines, FFT size etc. was different from the simulated data. This section describes the performance of the parallel system applied to the real data set which contains 1238 range lines, 161 low resolution profile and 200 samples points per low resolution profile. Input data was of size 319 MB(1238×4 byte×2×161×200) and the output data size was 649 MB(1238×8 byte×2×32768).

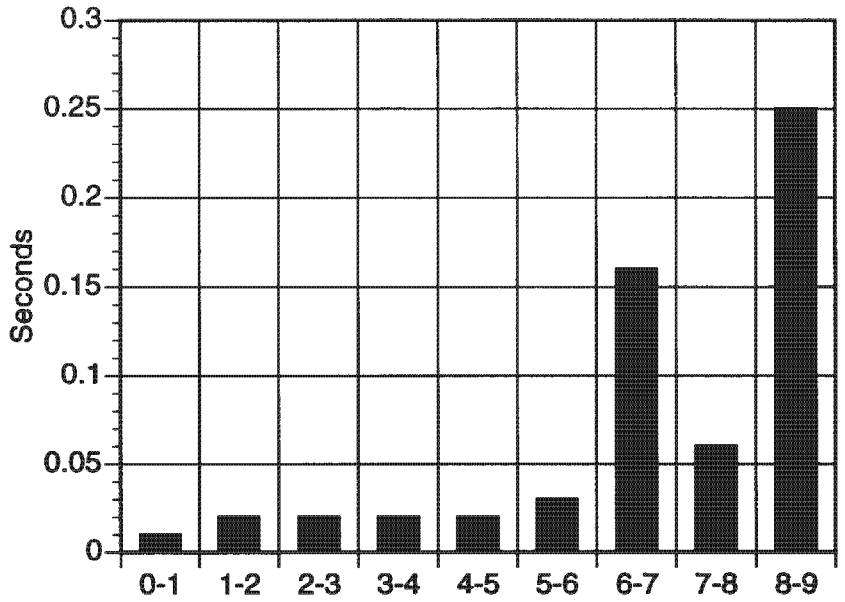

Figure 7.4: *Time Profile of the serial code which produces 1 high resolution profile*

7.2.1 Time profile of the serial code

The following Figure 7.4. shows the amount of time the serial code spends on the different parts of the program to process 1 range line of 65536 sample points (1 high resolution profile).

0 -1:Time to read and zeropad the range compressed calibrated data from 200 to 256 samples.

1 - 2 :Time to perform 161 FFT of 256 sample points on that data.

2 - 3 :Time to make compensation filter.

3 - 4 :Time to Read and zeropad the range compressed data of same size as calibrated data.

4 - 5 :Time to perform 161 FFT of 256 sample points on that data.

5 - 6 :Time to position and add the subspectrums.

6 - 7 :Time to apply the compensation filter.

7 - 8 :Time to position the final spectrum symmetrical to zero

8 - 9 :Time to apply the inverse Fourier Transform to the final spectrum

Looking at the Figure 7.4 it can be inferred that to produce all 1238 profiles, the time spent to make the compensation filter is negligible in compared to the overall time. The total execution time to process 1238 range lines was 1238.30 secs or 21 minutes in which it spends 1237 seconds within the main loop. This is because the calibrated data is loaded only once in the serial code to produce the compensation filter. Whereas, the range compressed data (low resolution profile) obtained from scene is loaded (refer Figure 6.1) each time it reconstructs the targets reflectivity spectrums and applies the compensation filter to produce 1 high resolution profile. It is interesting to note that FFT of an array of radix-2 number of elements (e.g. 256 sample points) takes less time than an array with elements of non radix-2 number (e.g. 200 sample points) even if the size of the former is bigger than the later.

7.2.2 Performance analysis of the parallel code

The timing diagram in Figure 7.4 is also valid for each worker process as the each worker process runs the same algorithm.

From Table 7.3 it can be seen that the maximum speedup achieved using 5 nodes is $S = 2.8$, corresponding to the 5 node efficiency of $E = 0.56$. The serial code was run on a single 350 MHz node in Gollach cluster.

Processor	Time(sec)	Speedup
Serial code (timed)	1238.30	
Theoretical performance for 5 node(100% efficiency)	247	5
Theoretical performance for 5 node(60% efficiency)	412	3
Parallel code-5 node(timed)	460	2.8

Table 7.3: *Comparison of serial to parallel code (for real data set).*

Table 7.4 shows the speedup and efficiency achieved spawning the master and one worker on dual-CPU node and 4 workers separately on the other 4 single-CPU nodes. In Table 7.4 the first node refers to the dual Pentium.

Nodes(n)	Time(T_p)	Speedup(S)	node efficiency(E_1)	CPU efficiency(E_2)
1	1612 sec	0.76	0.76	0.38
2	955 sec	1.24	0.62	0.41
3	717 sec	1.72	0.57	0.43
4	549 sec	2.25	0.56	0.45
5	438 sec	2.82	0.56	0.45

Table 7.4: *Changing the number of nodes in the parallel system with constant number of range lines (results for real data set)*

The Speed-up plot in Figure (7.5) shows linear scaling. Figure 7.6 shows the efficiency (Node and CPU) against number of nodes. Note that the 5-node efficiency decreases steeply from 2 to 3 nodes but then it decreases slowly achieving an efficiency of 0.56 for 5 node.

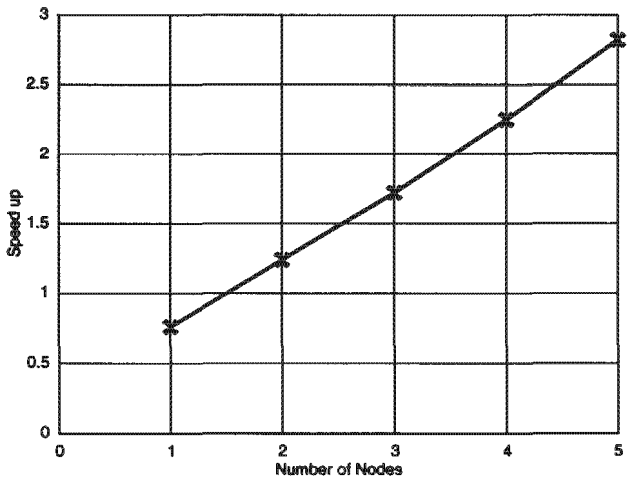

Figure 7.5: *The speed-up graph against number of nodes*

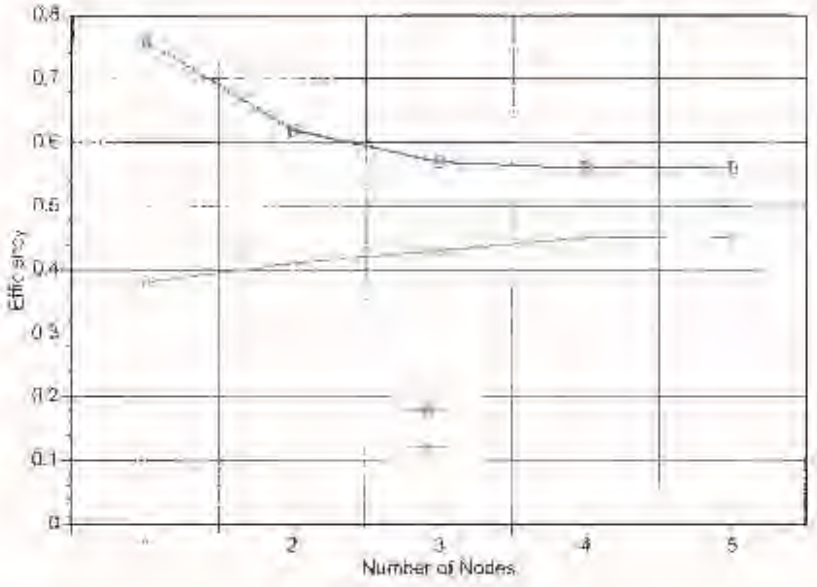

Figure 7.6: *The 5-node and 6-CPU efficiency graph against 5 node*

The main reason for not achieving a speedup above 3 is the high communication overhead of the parallel system. From Figure 6.2 it can be seen that the master first multicasts the calibration data of size 0.25 MB (8 bytes×161profiles×200samples) to all workers and then again it sends the same amount of data (enough to produce 1 high resolution profile) to each worker and receives 1 high resolution profile of size 0.52 MB (16 bytes×32768 samples) from each worker in a loop. This implies that to process all 1238 range lines the master needs to communicate 247 times with 5 workers plus 3 times more with 3 workers. In each communication cycle, master sends 161 low resolution profiles to each 5 worker and receives 5 high resolution profiles from 5 workers in a loop. Pvm_send is a *nonblocking* call so it continue to perform other operation before the communication has completed. But since pvm_recv is a *blocking* call the worker processor waits until the task is delivered to master processor.

Also, the time to write each processed high resolution profile to the disk was included with the timing results of the parallel code even if that operation was purely sequential. Another reason for not obtaining a optimum speedup

is that the PVM communication subroutines used in the code are regarded as one of the the slowest among all other available message passing model.

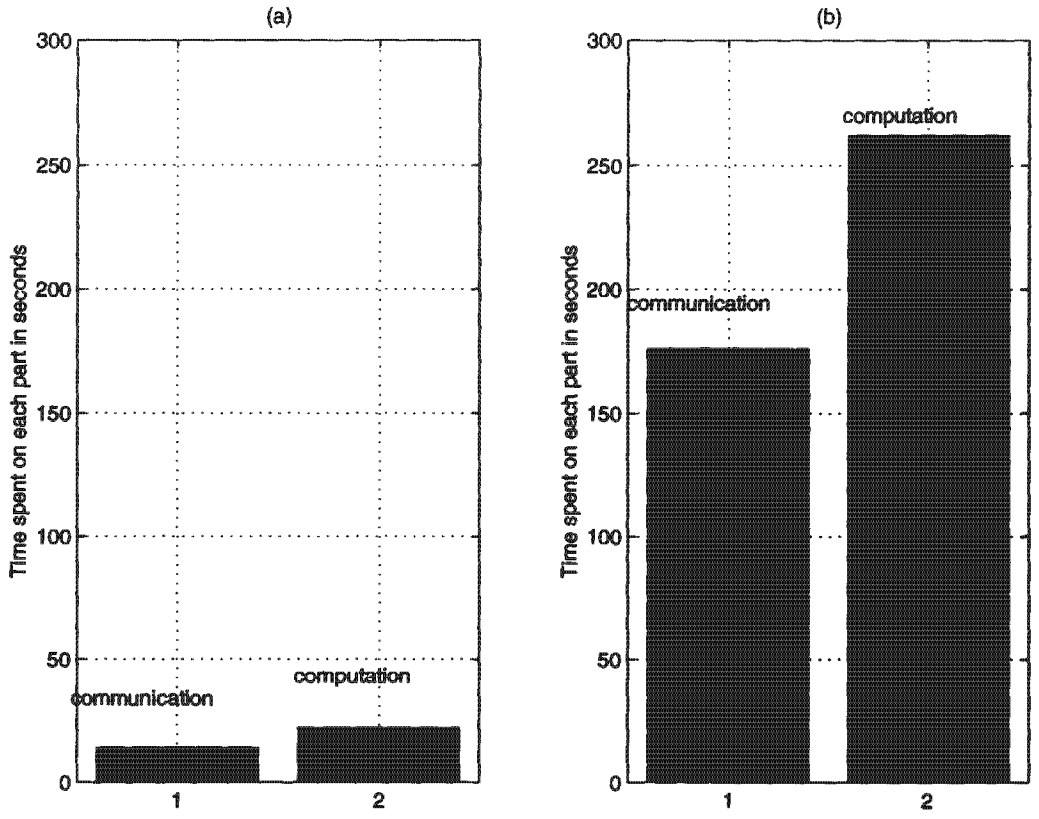

Figure 7.7: *The communication to computation ratio to produce a) 100 and b)1238 range lines.*

Figure 7.7 (a) and (b) shows the communication versus computation time ratio to process 100 and 1238 range lines respectively. To produce 100 range lines the code spent 22 seconds in computation and 14 seconds in communication which corresponds to a *granularity* (computation to communication ratio) of 1.56 whereas to produce 1238 line the code spent 262 seconds in computation and 176 seconds in communication i.e. a *granularity* of 1.47. Ideally, keeping the number of nodes fixed as we increase the size of the problem, the granularity remains almost constant. In 1238 line case, the granularity becomes a slightly lower than the 100 line case because the com-

munication cycle per node increases with the problem size and subsequently increases the total waiting time after each communication cycle.

7.2.3 XPVM output

The communication and computation time was measured using a PVM visualization tool called xpvm. Figure 7.8 shows a typical XPVM window with network (above) and task-time (below) view. Although the experiment was carried out using 1 master and 5 workers, Figure 7.8 displays the output of 1 master and 1 worker process. In the network window, we can see that one master (g8) and one worker (g2) node is active and the task-time window shows that the master and the worker is exchanging data between themselves. Each line between the two nodes show that the master and the worker is either sending or receiving a message. The slope of each line as shown in Figure 7.8 indicates the time spent in sending or receiving a message. We can see that the each worker process waits some time (the white part) after each communication cycle i.e. after it hand overs the final high resolution profile to master.

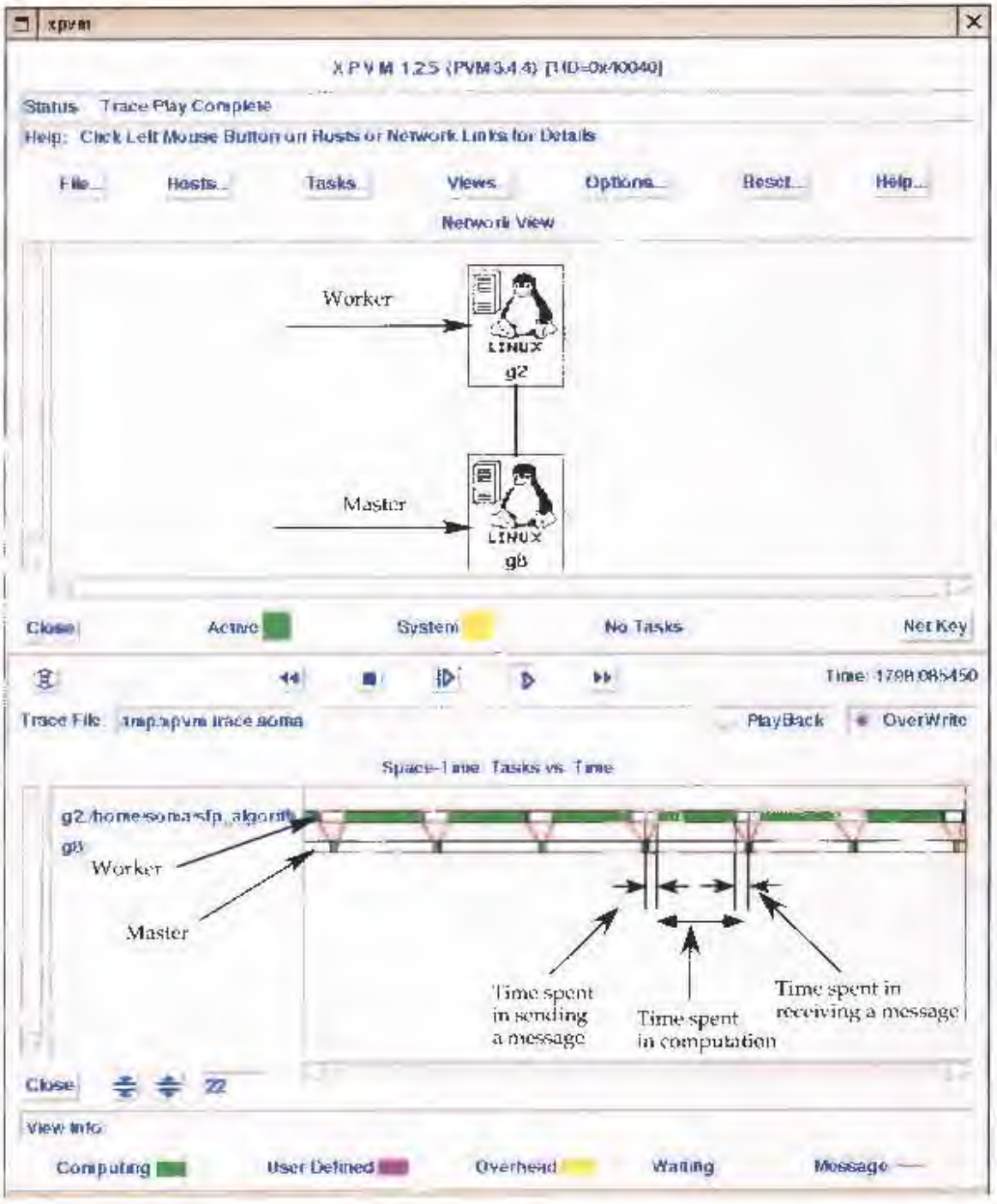

Figure 7.8: *XPVM window with network and task-time view*

It is obvious that the parallel system experienced high communication overhead penalties due to transfer of large number of small messages, but it made efficient use of available processor cycles. It was not possible to perform further experiments with increased nodes due to unavailability of any more nodes in the cluster.

7.2.4 Discussions

The above results (Table 7.2 and 7.4) show that the reason for obtaining a low efficiency (less than 60%) is the high communication overhead of the parallel system. In order to improve the performance of the parallel system a few changes were made. Firstly, the Nyquist oversample factor was reduced from 4 to 2 to decrease the output data size. Next, instead of sending a small packet of data to each worker, the data were sent and received in larger packets. For example, to process 1238 high resolution profiles using 5 nodes, the master process sends one data block containing data to form 247 high resolution profiles (63.62 MB) to each 4 workers and the last worker receives the remaining data i.e. 64.4 MB. Each worker then processes the required number of high resolution profiles i.e. 247 for 4 workers and 250 for the 5th worker in a loop and collectively sends them back to master. This is the best possible way to minimize the communication cycle because after sending the calibration data to each worker, the master needs to communicate only two times with each worker i.e. while sending the packets of low resolution profiles and receiving the packets of high resolution profiles. The final high resolution profiles were sent in single precision instead of double precision so as to reduce the byte size. The following subsubsection presents the timing results and does the performance analysis of the modified parallel system applied to the real data set to produce 100 high resolution profiles.

7.2.4.1 Timing results of the modified version of the parallel code

The serial code was run on a single 350 MHz node in Gollach cluster. The master process and one worker process in the parallel code was run on the Dual-CPU node and 4 worker processes separately on the other single-CPU

nodes.

The master process sent 20 packets of data to each 5 workers and received 20 packets of processed high resolution profiles from them. The input and output data size was 25 MB and 26 MB respectively.

The serial code on a single node took 47 seconds compared to 12 seconds for the parallel code on the 5 node cluster (see Table 7.5).

Table 7.5 shows that the maximum speedup achieved to produce 100 high resolution profiles using 5 nodes is $S = 3.75$, corresponding to the node efficiency of $E = 75\%$.

Processor	Time(sec)	Speedup
Serial code (timed)	47	
Theoretical performance for 5 node(100% efficiency)	9.4	5
Theoretical performance for 5 node(60% efficiency)	15.6	3
Parallel code-5 node(timed)	12	3.75

Table 7.5: *Comparison of serial to parallel code for the modified version of parallel algorithm (for real data set).*

Table 7.6 shows the speedup and efficiency achieved by changing the number of nodes but keeping the total number of range lines constant (i.e. 100 high resolution profiles). It can be seen from graph 7.9 that the system did not achieve a linear speedup from 1 to 2 processors but the speedup shows linear scaling as more processors are added in the system. Graph 7.10 shows the Node and CPU Efficiency of the parallel system. Node efficiency remains almost constant from 3 to 5 node.

Nodes(n)	Time(T_p)	Speedup(S)	Node efficiency(E_1)	CPU efficiency(E_2)
1	47 sec	0.95	0.95	0.47
2	28.2 sec	1.66	0.83	0.55
3	21 sec	2.23	0.74	0.55
4	16 sec	2.82	0.70	0.56
5	12 sec	3.75	0.75	0.62

Table 7.6: *Changing the number of nodes in the modified parallel algorithm with constant number of range lines (100 range lines)*

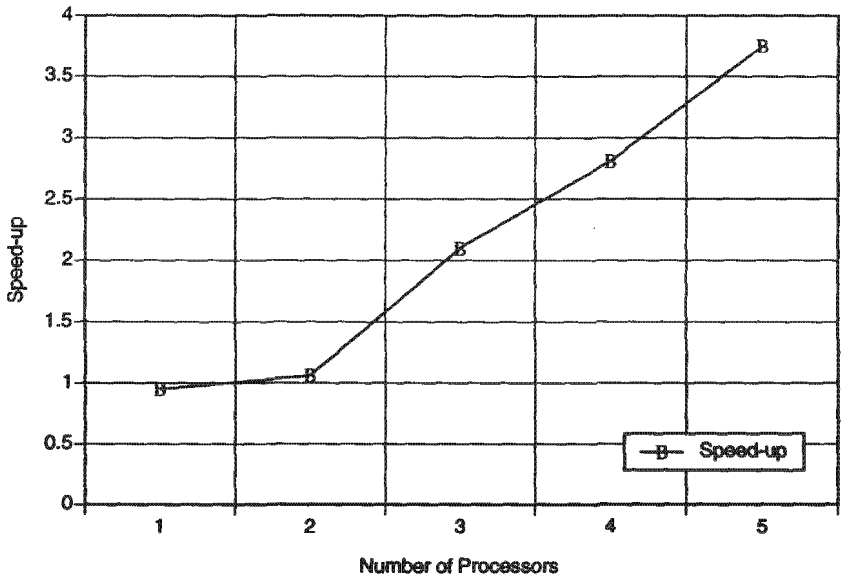

Figure 7.9: *Graph showing speedup versus the number of nodes.*

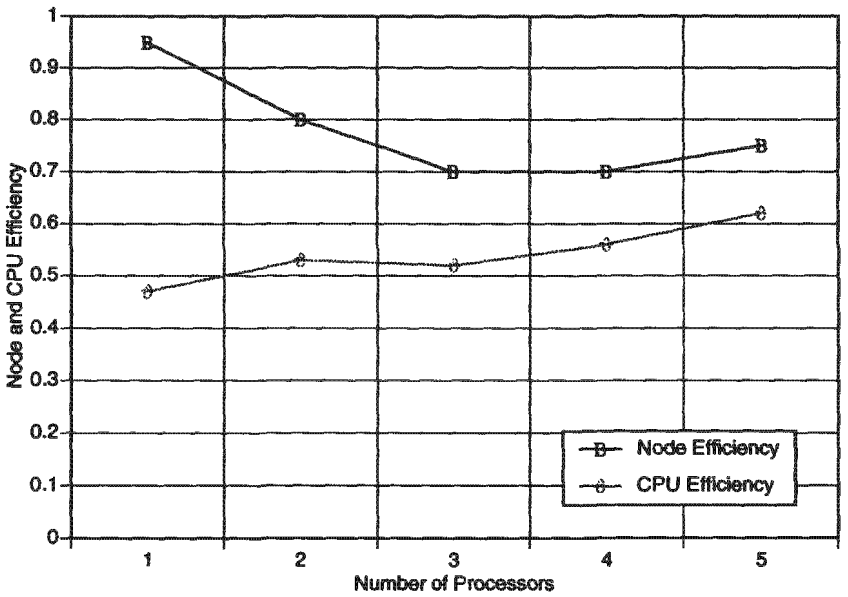

Figure 7.10: *The Node Efficiency and CPU Efficiency graph versus the number of nodes*

Chapter 8

Conclusions and future work

The purpose of this dissertation has been fulfilled by 1) successfully implementing the stepped-frequency algorithm on simulated and real data 2) by developing and testing of a parallel version of the stepped-frequency processing algorithm.

It has been verified that the high resolution range profiles can be obtained by combining the narrow-band pulses in the frequency domain. The signal processing steps in the frequency domain method are fast, since only FFTs and phase multiplications are used and also it has been concluded that the integer number of shift in frequency is easy to implement in software. The integer shift method efficiently positions all the sub-spectrums into their correct places. The resolution gets finer as we increase the number of steps. Care should be taken in constructing the compensation filter the function of which is to compensate for amplitude "ripples" at the sub-spectra boundaries. The algorithm has been successfully tested on simulated data, showing that the high resolution is achieved both by overlapping and splicing the sub-spectra. But, in the case of the RoofSAR real data the overlapping method was not quite successful. The splicing method showed better results in the case of the processing of real data although few repeats appeared near the left edge of the high resolution profile. This was probably due to the fact that data was range compressed and provided to me in this format using an ideal chirp waveform to do the matched filtering which did not account for

all linear system effects in the receiver chain.

The feasibility of distributed parallel computing has been shown by implementing the SFP algorithm on a cluster of low-cost desktop computers each running Linux.

Two approaches were taken in order to parallelize the SFP algorithm. In the 1st implementation, data were sent to each worker process in small blocks i.e. enough to produce 1 high resolution profile. The timing results obtained from both simulated and real data shows that the parallel system is scalable because the speedup increased as the nodes were increased. The Node Efficiency achieved to process 1238 high resolution profiles was nearly 60%. The reason for not achieving an efficiency above 60% is that the system suffers from high network overhead due to transfer of a large amount of small messages.

In order to minimize the communication overhead and the disk I/O inefficiencies, the previously developed algorithm was modified. The receiving and sending packet size was increased to it's maximum possible value so that the master needs to communicate only once in order to send and receive the scene data. This algorithm was applied on the real data and the efficiency achieved to produce 100 high resolution profiles was 75%. But the limitation of this algorithm was that if the sending and receiving block size is too large (for example 100 MB) with respect to the memory space of the processor then the processor starts page swapping the data and shows a performance degradation.

Scope for future work includes:

- Closely investigating the signal processing steps in the SFP algorithm applied to the real data so that the high resolution can be achieved for the overlap case and the repeats are reduced in the splicing case.
- Implementing stepped-frequency processing on moving targets.
- Implementation in real-time radar context.
- An improvement can be made in overall performance of the parallel system by sending and receiving the data in large packets in a loop to

minimize the communication and disk I/O operation. The packet size is however limited to a maximum by the memory constraints on each node.

- If the algorithm is implemented in a heterogeneous system, the data can be sent dynamically to the processors, i.e, instead of distributing the data sequentially in a loop, the master can send the data to whichever processor finishes it's job first. In that way, the time spent in waiting can be reduced.

Appendix A

Source Code

A.1 Reconstruction Algorithm (C code)

```
% Load simulator parameters
   setup_simulator;
   debug_level =2; % all debug info of level'standard'
   debug_message('\nStepped Frequency Simulator - Author:  Soma\n\n\n',debug_level,se
   debug_message('\nLoaded setup parameters',debug_level,setup);
   % Initialise timer ============================
   mark_time = 0;
   % reset matlab's timer=======================
   display_time(mark_time,setup); % display the time and write to the
log file
   % Load radar parameters======================
   debug_message('\n\nLoading radar parameters',debug_level,setup)
   setup_radar_real
   mark_time = display_time(mark_time,setup);
   p = compute_next_pow_2(Radar.N_samples);
   fid = fopen('data.rnc','r');
   if (fid ==-1)
   disp('Error opening file!');
   end;
   NewSamples_1=fread(fid,2*Radar.N_samples*Radar.N_steps,'float');
```

```
NewSamples_1= reshape(NewSamples_1,2*Radar.N_samples,Radar.N_steps).';
NewSamples_1 = complex_data(NewSamples_1);
p = compute_next_pow_2(Radar.N_samples);
fid = fopen('calib.rnc','r');
if (fid ==-1)
disp('Error opening file!');
end;
NewSamples_2 = fread(fid,2*Radar.N_samples*Radar.N_steps,'float');
NewSamples_2= reshape(NewSamples_2,2*Radar.N_samples,Radar.N_steps).';
NewSamples_2 = complex_data(NewSamples_2);
debug_message('\n\nLoading real and calibration data and applying zeropadding',deb
%Zero-padding the vector in time domain==========
p = compute_next_pow_2(Radar.N_samples);
V_t_output_padded = zeros(Radar.N_steps,p);
V_t_single_output_padded = zeros(Radar.N_steps,p);
for i = 1:Radar.N_steps
V_t_output_padded(i,1:Radar.N_samples) = NewSamples_1(i,1:Radar.N_samples);
V_t_single_output_padded(i,1:Radar.N_samples) = NewSamples_2(i,1:Radar.N_samples);
end;
profiles =(fft(V_t_output_padded.').');
% Converting to time domain================
single_t_profiles = (fft(V_t_single_output_padded.').');
t_start = 0;
dt = Radar.dt;
if rem(p,2)==0
% case even ===========
t = [(0:(p)/2),-(p)/2+1:-1]*dt+t_start;
else
% case off ================
t = [(0:(p-1)/2),-(p-1)/2:-1]*dt+t_start;
end
% calculating the number of samples=========
df_1 = 1/(p*Radar.dt);
df_1
if rem(p,2)==0
```

```
f = [0:1:(p)/2, -((p)/2)+1:1:-1]*df_1;
else
% Case odd number of samples====================
f= [0:1:(p)/2-1,-(p+1)/2:1:-1]*df_1;
end
N_total = round((Radar.Btotal/df_1).*Radar.oversample_factor);
N_total = compute_next_pow_2(N_total);
dt_1 = 1/(N_total*df_1);
t_start = 0;
if rem(N_total,2)==0
% case even ====================
t1 = [(0:(N_total)/2),-(N_total)/2+1:-1]*dt_1+t_start;
else
% case off
t1 = [(0:(N_total-1)/2),-(N_total-1)/2:-1]*dt_1+t_start;
end
R = Radar.c*t1./2; % creating range vector lebel from time vector
% creating frequency lebel in vector==========
df_2 =1/(N_total*dt_1);
if rem(N_total,2)==0
f_1 = [0:1:(N_total)/2, -((N_total)/2)+1:1:-1]*df_2;
else
% Case odd number of samples=========
f_1 = [0:1:(N_total)/2-1,-(N_total+1)/2:1:-1]*df_2;
end
s_chirp =round(Radar.Bchirp/df_1);
mark_time = display_time(mark_time,setup);
% creating time vector========
% creating a vector to store the data for each subspectra after shift============
V = zeros(Radar.N_steps,N_total);
% creating a vector to store the subspectra for calibration point ===============
V_single_target= zeros(Radar.N_steps,N_total);
r = N_total;
n= Radar.N_steps;
%s=radar.Bchirp/radar.df; %bandwidth in terms of number of samples
```

```
debug_message('\n\nShifting and copying each subspectra in final array',debug_leve
V(1,1:s_chirp/4) = profiles(1,1:s_chirp/4);
V(1,r-s_chirp/4:r) = profiles(1,p-s_chirp/4:p);
sum =V(1,:);
for i= 2:n
Radar.number_shift =(Radar.delta_f/df_1);
fshift = Radar.number_shift;
fshift = (i-1)*(Radar.number_shift);
fshift= fshift+(i-2);
V(i,fshift+1-s_chirp/4:fshift+1)= profiles(i,p-s_chirp/4:p);
V(i,fshift+2:fshift+1+s_chirp/4)=profiles(i,1:s_chirp/4);
sum =sum+V(i,:);
result_1 =(sum);
end
%converting into time domain to see it's resolution======
v_result=(ifft(result_1.')).';
mark_time = display_time(mark_time,setup);
debug_message('\n\n Creating the compensation filter',debug_level,setup)
%Creating the compensation filter to smoothen the ripples
%shifting the subspectrums obtained from callibration point=====================
V_single_target(1,1:s_chirp/4) =single_t_profiles(1,1:s_chirp/4);
V_single_target(1,r-s_chirp/4:r)=single_t_profiles(1,p-s_chirp/4:p);
sum = V_single_target(1,:);
for i= 2:n
fshift = Radar.number_shift;
fshift = (i-1)*(Radar.number_shift);
fshift= fshift+(i-2);
V_single_target(i,fshift+1-s_chirp/4:fshift+1)= single_t_profiles(i,p-s_chirp/4:p)
V_single_target(i,fshift+2:fshift+1+s_chirp/4) = single_t_profiles(i,1:s_chirp/4);
sum =sum+V_single_target(i,:);
U_calibration =(sum);
end
%making the compensation filter==============
U_compensation = 1./U_calibration;
```

```
   s_total=round(Radar.Btotal/df_1); % Total Radar bandwidth in terms
of number of samples
   U_compensation(s_total-s_chirp/4:r-s_chirp/4) = 0;
   %Applying the compensation filter to the target spectrum==========
   U_final_result = result_1.*U_compensation;
   mark_time = display_time(mark_time,setup);
   debug_message('\n\nPositioning the spectra symmetrical to zero',debug_level,setup)
   %positioning the spectra symmetrical to zero
   % Converting the spectrum to time domain
   u_final_result = (ifft(U_final_result.')).';
   % multiplying it with exp(-j*2*pi*fshift_1*t) to shift the spectra
left
   i=1; % one reconstructed wideband spectrum
   fshift_hz = ((n+1)/2-i).*Radar.delta_f;
```

A.2 Parallel version of Reconstruction Algorithm (C code)

Master

```
   #include <stdio.h>
   #include <stdlib.h>
   #include <pvm3.h>
   #include <time.h>
   #include <math.h>
   #include "globals.h"
   #define SLAVE "/home/soma/TEMP/soma/sfp_algorithm/sfp_worker_real1"
   #include <sys/time.h>
   #define STOP 0
   setup_radar_real()
   {
   radar.f1 = 8.9e9; //start frequency
   radar.delta_f = 7.5e6; //Frequency step
```

```
radar.Tchirp = 3.125e-7; //pulse length (not used in reconstructionalg)
radar.Bchirp = 15e6;
radar.N_steps = 161;
radar.c = 3e8;
radar.overlap = (radar.Bchirp-radar.delta_f);
radar.oversample_factor = 4;
radar.sample_frequency = 20e6;
radar.dt = 1/radar.sample_frequency;
radar.N_samples = 200; //Number of range bins
radar.df = 1/(radar.N_samples*radar.dt);
// Total chirp bandwidth
radar.Btotal = 1.20750e9;
}
/*********START OF MAIN***************/
main(int argc, char **argv)
{
FILE *fpin,*fpout;
int i,p,l,r,k,j,NumElements,Num_Elements,N_total,nproc = PROC;
int no,bufid, /*PVM MESSAGE BUFFER ID*/
msgtype,bytes,rc,status;
int tid, /*PVM TASK ID*/
tids[PROC],index,times,Remainder;
char InFileName_1[80] = "N_lowresprofiles.dat";
char InFileName_2[80] = "singlelowresprofiles.dat";
char OutFileName[80] = "N_highresprofiles.dat";
double t_start,dt,df_1,dt_1,df_2;
float *Fsamples1,*Fsamples2;
double *NewSamples1,*NewSamples2;
_complex_data *u_final;
struct timeval StartTime;
struct timeval EndTime;
//call the setup radar data
setup_radar_real( );
if (argc != 3)
{
```

```
//printf("Usage:  ./sfp_master_real1 <file1> <file2>\n");
exit(0);
}
sprintf(InFileName_1,argv[1]);
sprintf(InFileName_2,argv[2]);
// factor 2 is for complex
NumElements = radar.N_steps * radar.N_samples * 2;
Fsamples1 = (float*)malloc(sizeof(float)*NumElements);
Fsamples2 = (float*)malloc(sizeof(float)*NumElements);
if (!Fsamples1 || !Fsamples2 )
{
printf("Memory Allocation Error 1\n");
exit(1);
}
//Opening the other input file
fpin = fopen(InFileName_2,"rb"); //open input file.
if (!fpin)
{
printf("Error Opening Input File[%s]",InFileName_2);
exit(1);
}
//read the double binary data into double array
fread(Fsamples2,sizeof(float),NumElements,fpin);
fclose(fpin);
/*****COMPUTING N_total******************/
t_start = 0;
dt = 1.0/(radar.sample_frequency);
p =compute_next_2_pow_n(radar.N_samples);
df_1 = 1.0/(p*radar.dt);
N_total = ceil((radar.Btotal/df_1) * radar.oversample_factor);
N_total = compute_next_2_pow_n(N_total);
/************Enroll the process in PVM ***/
rc = pvm_mytid();
if(rc < 0)
{
```

```
printf("MASTER:Unable to enroll this task.\n");
printf(" Enroll return = %d.Quitting.\n",rc);
exit(0);
}
else
pvm_catchout(stdout);
/*********start Slave Tasks******/
rc = pvm_spawn(SLAVE,NULL,PvmTaskDefault, "",nproc,tids);
if (rc == nproc)
printf("MASTER: Successfully spawned %d worker tasks.\n", rc);
else
{
printf("MASTER: Not able to spawn requested number of tasks!\n");
printf("MASTER: Tasks actually spawned:  %d.  Quitting.\n",rc);
exit(0);
}
//send initialisation data to slave processes
Remainder = MAX_HIGHRES_PROFILES%nproc;
times = (MAX_HIGHRES_PROFILES-Remainder)/nproc;
fprintf(stderr,"times = %d\n",times);
pvm_initsend(PvmDataDefault);
pvm_pkint(&times,1,1);
pvm_pkint(&Remainder,1,1);
pvm_mcast(tids,nproc,INIT_MSG);
u_final = (_complex_data*)malloc(sizeof(_complex_data)*N_total);
//Allocating memory to store the final high resolution profile
/******************Broadcast Calibration data to slave*************/
fpin = fopen(InFileName_1,"rb"); //open input file.
if (!fpin)
{
printf("Error Opening Input File[%s]",InFileName_1);
exit(1);
}
fpout = fopen(OutFileName,"wb"); //open output file
if (!fpout)
```

```
{
printf("Error Opening Output File [%s]",OutFileName);
exit(1);
}
/***********get start time*******************/
gettimeofday(&StartTime, NULL);
msgtype = FROMMASTER_MSG;
rc = pvm_initsend(PvmDataDefault);
rc = pvm_pkint(&nproc,1,1);
rc = pvm_pkint(tids,nproc,1);
rc = pvm_pkint(&NumElements,1,1);
rc = pvm_pkfloat(Fsamples2,NumElements,1);
rc = pvm_mcast(tids,nproc,msgtype);
/*Broadcast data to slaves*/
index =0;
msgtype = FROMMASTER_MSG;
for(j =0 ; j < times; j++)
{
index = j*nproc;
for(i =0;i<nproc;i++)
{
fread(Fsamples1,sizeof(float),NumElements,fpin);
index += i;
pvm_initsend(PvmDataDefault);
pvm_pkint(&index,1,1);
pvm_pkint(&NumElements, 1, 1);
pvm_pkfloat(Fsamples1, NumElements, 1);
pvm_send(tids[i],FROMMASTER_MSG);
}
index = j*nproc;
for(i =0;i<nproc;i++)
{
index += i;
bufid = pvm_recv(tids[i],FROMSLAVE_MSG);
rc = pvm_upkint(&index,1,1);
```

```
rc = pvm_upkint(&N_total,1,1);
rc = pvm_upkdouble((double*)u_final,N_total*2,1);
fwrite(u_final,sizeof(_complex_data),N_total,fpout);
}
}
index = times*nproc;
for(i =0;i<nproc;i++)
{
if (Remainder > 0)
{
fread(Fsamples1,sizeof(float),NumElements,fpin);
index += i;
pvm_initsend(PvmDataDefault);
pvm_pkint(&index,1,1);
pvm_pkint(&NumElements, 1, 1);
pvm_pkfloat(Fsamples1, NumElements, 1);
pvm_send(tids[i],FROMMASTER_MSG);
}
else // Send ZERO.
{
//STOP = 0;
status = STOP;
pvm_initsend(PvmDataDefault);
pvm_pkint(&status,1,1);
pvm_send(tids[i],FROMMASTER_MSG);
}
}
index = times*nproc;
for(i =0;i<Remainder;i++)
{
index += i;
bufid = pvm_recv(tids[i],FROMSLAVE_MSG);
rc = pvm_upkint(&index,1,1);
rc = pvm_upkint(&N_total,1,1);
rc = pvm_upkdouble((double*)u_final,N_total*2,1);
```

```
rwrite(u_final,sizeof(_complex_data),N_total,fpout);
}
fclose(fpin);
fclose(fpout);
pvm_exit();
/*get end time*/
gettimeofday(&EndTime, NULL);
printf("\nTime = %f secs.\n", ((EndTime.tv_sec -
StartTime.tv_sec) + (EndTime.tv_usec - StartTime.tv_usec)*0.000001));
return(0);
}
```

Worker

```
#include"stdio.h"
#include <stdlib.h>
#include<math.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_fft_complex.h>
#include "globals.h"
#include "pvm3.h"
#include <time.h>
#include <sys/time.h>
#define STOP 0
setup_radar_real()
{
radar.f1 = 8.9e9; //start frequency
radar.delta_f = 7.5e6; //Frequency step
radar.Bchirp = 15e6;
radar.N_steps = 161;
radar.c = 3e8;
radar.overlap = (radar.Bchirp-radar.delta_f);
radar.oversample_factor = 4;
radar.sample_frequency = 20e6;
radar.dt = 1/radar.sample_frequency;
radar.N_samples = 200; //Number of range bins
```

```
   radar.df = 1/(radar.N_samples*radar.dt);
   radar.Btotal = 1.20750e9;
   }
   // Routine to perform complex multiplication
   void complex_mult(_complex_data val1, _complex_data val2, _complex_data
*result)
   {
   result->real = (val1.real * val2.real) - (val1.img * val2.img);
   result->img = (val1.real * val2.img) + (val1.img * val2.real);
   }
   // Routine to perform complex addition
   void complex_add(_complex_data val1, _complex_data val2, _complex_data
*result)
   {
   result->real = val1.real + val2.real;
   result->img = val1.img + val2.img;
   }
   // Routine to compute the next 2^n value for a given number
   // e.g.  if n = 853 the returned val = 1024
   int compute_next_2_pow_n(int n)
   {
   int val = 1, flag = 1;
   while (flag)
   {
   val *= 2;
   if (val > n)
   flag = 0;
   }
   return(val);
   }
   //START OF MAIN
   main(int argc, char **argv)
   {
   FILE *fpin,*fpout;
   int i,p,n,l, r, j,k,NumElements,Num_Elements;
```

```
    int nproc = PROC;
    int mytid; /*PVM TASK ID*/
    int tids[PROC];
    int masterid,rc,index,msgtype,times,Remainder,status;
    char InFilename_1[120],InFilename_2[80],OutFileName[80];
    double t_start,dt,df_1,dt_1,df_2,fshift_hz,start_time,End_time,module_time;
    double ftmpVal1,ftmpVal2;
    int N_total,s_total,s_chirp, fshift;
    double *t1,*R,*f_1, *fptr_1,*fptr;
    float *Fsamples1;
    float *Fsamples2;
    double *NewSamples1,*NewSamples2;
    _complex_data cVal1,cVal2,cResultVal1,cResultVal2;
    _complex_data *V,*V_single_target,*sum,*sum_2, *Vptr,*Vptr_2,*result_1,
*U_calibration, *U_compensation;
    _complex_data *U_final_result, *u_final, *final_result;
    struct timeval StartTime;
    struct timeval EndTime;
    //call the setup radar data
    setup_radar_real ( );
    // factor 2 is for complex
    NumElements = radar.N_steps * radar.N_samples * 2;
    Fsamples1=(float*)malloc(sizeof(float)*NumElements);
    Fsamples2 = (float*)malloc(sizeof(float)*NumElements);
    p = compute_next_2_pow_n(radar.N_samples);
    Num_Elements = radar.N_steps * p * 2;
    NewSamples2 = (double *)malloc(sizeof(double)*Num_Elements);
    NewSamples1 = (double *)malloc(sizeof(double)*Num_Elements);
    if (!Fsamples1||!Fsamples2||!NewSamples1||!NewSamples2)
    {
    printf("Memory Allocation Error\n");
    exit(1);
    }
    mytid = pvm_mytid();
    masterid = pvm_parent();
```

```
msgtype = FROMMASTER_MSG;
//receive init data from master process
rc = pvm_recv(masterid,INIT_MSG);/*masterid = -1*/
pvm_upkint(&times, 1, 1);
pvm_upkint(&Remainder, 1, 1);
/*****************Receive calibration data from master*************/
pvm_recv(masterid,FROMMASTER_MSG);
pvm_upkint(&nproc, 1, 1);
pvm_upkint(tids, nproc, 1);
pvm_upkint(&NumElements,1, 1);
pvm_upkfloat(Fsamples2, NumElements, 1);
for(i =0;i<radar.N_steps*p;i++)
{
NewSamples2[2*i] = 0.0;
NewSamples2[2*i+1] = 0.0;
}
for(i =0;i<radar.N_steps;i++)
{
for(j = 0;j<radar.N_samples;j++)
{
NewSamples2[(i*p*2)+(2*j)] =Fsamples2[(i*radar.N_samples*2)+(2*j)];
NewSamples2[(i*p*2)+(2*j+1)] =
Fsamples2[(i*radar.N_samples*2)+(2*j+1)];
}
}
gettimeofday(&StartTime, NULL);
df_1 = 1.0/(p*radar.dt);
s_chirp =ceil(radar.Bchirp/df_1);
for(i=0;i<radar.N_steps;i++)
{
gsl_fft_complex_radix2_forward(&NewSamples2[i*p*2],1,p);
}
for (i = 0; i < radar.N_steps; i++)
{
for(j = s_chirp/4; j < p-(s_chirp/4-1); j++)
```

```
{
NewSamples2[(i*p*2)+(2*j)] = 0;
NewSamples2[(i*p*2)+(2*j+1)] = 0;
}
}
//creating time vector
t_start = 0;
N_total = ceil((radar.Btotal/df_1) * radar.oversample_factor);
N_total = compute_next_2_pow_n(N_total);
dt_1 = 1/(N_total*df_1);
df_2 =1/(N_total*dt_1);
radar.number_shift = radar.delta_f/df_1;
t1 = (double *) malloc(sizeof(double) * N_total);
f_1= (double *) malloc(sizeof(double) * N_total);
R= (double *) malloc(sizeof(double) * N_total);
sum = (_complex_data*) malloc(sizeof(_complex_data) * N_total);
if (!t1 || !f_1 || !R || !sum)
{
printf("Memory Allocation Error\n");
exit(1);
}
for (i = 0; i<N_total; i++)
{
t1[i] = ((i-N_total/2)*dt_1)+t_start;
f_1[i] = (i- N_total/2)*df_2;
R[i] = radar.c * t1[i]/2;
}
// memory allocation for the shifted spectra from single target
V_single_target = (_complex_data*)calloc(sizeof(_complex_data),N_total*radar.N_ste
// memory allocation to store the data of the spectra from calibration
U_calibration = (_complex_data*)malloc(sizeof(_complex_data)*N_total);
//Initializing the array for storing the data for compensation filter
U_compensation = (_complex_data*)calloc(sizeof(_complex_data),N_total);
if (!V_single_target || !U_calibration || !U_compensation)
{
```

```
    printf("Memory Allocation Error\n");
    exit(1);
    }
    // Shifting and adding the spectras from real and calibrated point
    r = N_total;
    n = radar.N_steps;
    //copying the elements from 1st row
    for(i= 0; i<s_chirp/2; i++)
    {
    V_single_target[i].real = NewSamples2[2*i];
    V_single_target[i].img = NewSamples2[2*i+1];
    }
    // Shifting and adding the subspectrums from calibration point
    for(i= 0; i<s_chirp/2; i++)
    {
    V_single_target[(r - s_chirp/2) + i].real = NewSamples2[2*((p - s_chirp/2)
+ i)];
    V_single_target[(r - s_chirp/2) + i].img = NewSamples2[2*((p - s_chirp/2)
+ i) + 1];
    }
    //This is a complex addition
    for(l=0; l < 1; l++)
    {
    for(j =0; j < N_total; j++)
    {
    sum[j].real = V_single_target[l*r + j].real;
    sum[j].img = V_single_target[l*r + j].img;
    }
    }
    //copying the elements from other row
    for(i = 1 ;i<n ;i++)
    {
    fshift = i * radar.number_shift;
    Vptr = &V_single_target[fshift];
    fptr = &NewSamples2[2*p*i];
```

```
for(j = 0;j < s_chirp/2;j++)
{
Vptr[(-s_chirp/2) + j].real = fptr[2*((p - s_chirp/2) + j)];
Vptr[(-s_chirp/2) + j].img = fptr[2*((p - s_chirp/2) + j) +1];
Vptr[j].real = fptr[2*j];
Vptr[j].img = fptr[2*j + 1];
}
for(j = -s_chirp/2; j < s_chirp/2; j++)
{
sum[j+fshift].real += Vptr[j].real;
sum[j+fshift].img += Vptr[j].img;
}
}
// copying sum array onto result_1 and U_calibration
for(j = 0; j < N_total; j++)
{
U_calibration[j].real = sum[j].real;
U_calibration[j].img = sum[j].img;
}
for(j = 0; j < N_total; j++)
{
U_compensation[j].real =
U_calibration[j].real/(pow(U_calibration[j].real,2)+pow(U_calibration[j].img,2));
U_compensation[j].img
=-(U_calibration[j].img/(pow(U_calibration[j].real,2)+pow(U_calibration[j].img,2))
}
//Inverting the spectra to get the compensation filter
//Making all the infinity zero
s_total = ceil(radar.Btotal/df_1);
for(i = s_total-s_chirp/4;i<r-s_chirp/4+1;i++)
{
U_compensation[i].real =0;
U_compensation[i].img =0;
}
for(i = 0; i < radar.N_steps-1; i++){
```

```
U_compensation[(2*i+1)*s_chirp/4].real = 0;
U_compensation[(2*i+1)*s_chirp/4].img = 0;
}
U_final_result = (_complex_data*)malloc(sizeof(_complex_data)*N_total);
u_final = (_complex_data*)malloc(sizeof(_complex_data)*N_total);
if (!U_final_result || !u_final )
{
printf("Memory Allocation Error\n");
exit(1);
}
//RUN SLAVE TIMES NUMBER TIMES
for(k =0; k<times;k++)
{
printf("times = %d\n",times);
//Receive real data from master
msgtype = FROMMASTER_MSG;
rc = pvm_recv(masterid, FROMMASTER_MSG);
rc = pvm_upkint(&index, 1, 1);
rc = pvm_upkint(&NumElements, 1, 1);
rc = pvm_upkfloat(Fsamples1, NumElements, 1);
result_1 = (_complex_data*)calloc(sizeof(_complex_data),N_total);
//Processing high resolution profile
for(i =0;i<radar.N_steps*p;i++)
{
NewSamples1[2*i] = 0.0;
NewSamples1[2*i+1] = 0.0;
}
for(i =0;i<radar.N_steps;i++)
{
for(j = 0;j<radar.N_samples;j++)
{
NewSamples1[(i*p*2)+(2*j)] =Fsamples1[(i*radar.N_samples*2)+(2*j)];
NewSamples1[(i*p*2)+(2*j+1)] =
Fsamples1[(i*radar.N_samples*2)+(2*j+1)];
}
```

```
}
//Function to perform FFT
for(i=0;i<radar.N_steps;i++)
{
gsl_fft_complex_radix2_forward(&NewSamples1[i*p*2],1,p);
}
for (i = 0; i < radar.N_steps; i++)
{
for(j = s_chirp/4; j < p-(s_chirp/4-1); j++)
{
NewSamples1[(i*p*2)+(2*j)] = 0;
NewSamples1[(i*p*2)+(2*j+1)] = 0;
}
}
U_final_result = (_complex_data*)calloc(sizeof(_complex_data),N_total);
u_final = (_complex_data*)calloc(sizeof(_complex_data),N_total);
if (!U_final_result || !u_final )
{
printf("Memory Allocation Error\n");
exit(1);
}
//creating a vector to store shifted spectra from real target
V = (_complex_data*)calloc(sizeof(_complex_data),N_total*radar.N_steps);
if (!V)
{
printf("Memory Allocation Error\n");
exit(1);
}
//copying the elements from 1st row
for(i= 0; i<s_chirp/2; i++)
{
V[i].real = NewSamples1[2*i];
V[i].img = NewSamples1[2*i+1];
}
for(i= 0; i<s_chirp/2; i++)
```

```
    {
    V[(r - s_chirp/2) + i].real = NewSamples1[2*((p - s_chirp/2) + i)];
    V[(r - s_chirp/2) + i].img = NewSamples1[2*((p - s_chirp/2) + i) +
1];
    }
    //This is a complex addition
    for(l=0; l < 1; l++)
    {
    for(j =0; j < N_total; j++)
    {
    sum[j].real = V[l*r + j].real;
    sum[j].img = V[l*r + j].img;
    }
    }
    //copying the elements from other row
    for(i = 1 ;i<n ;i++)
    {
    fshift = i * radar.number_shift;
    Vptr = &V[fshift];
    // Point to ith Fsamples1[ ] 1024 samples
    fptr = &NewSamples1[2*p*i];
    for(j = 0;j < s_chirp/2;j++)
    {
    Vptr[(-s_chirp/2) + j].real = fptr[2*((p - s_chirp/2) + j)];
    Vptr[(-s_chirp/2) + j].img = fptr[2*((p - s_chirp/2) + j) +1];
    Vptr[j].real = fptr[2*j];
    Vptr[j].img = fptr[2*j + 1];
    }
    for(j = -s_chirp/2; j < s_chirp/2; j++)
    {
    sum[j+fshift].real += Vptr[j].real;
    sum[j+fshift].img += Vptr[j].img;
    }
    }
    // copying sum array onto result_1 and U_calibration
```

```
    for(j = 0; j < N_total; j++)
    {
    result_1[j].real = sum[j].real;
    result_1[j].img = sum[j].img;
    }
    free(V);
    //Applying the compensation filter to the target spectrum
    //Complex multiplication
    for (j = 0; j < N_total; j++)
    {
    cVal1.real = result_1[j].real;
    cVal1.img = result_1[j].img;
    cVal2.real = U_compensation[j].real;
    cVal2.img = U_compensation[j].img;
    complex_mult(cVal1,cVal2,&cResultVal1);
    // copying the result of the multiplication to the array U_final_result
    U_final_result[j].real = cResultVal1.real;
    U_final_result[j].img = cResultVal1.img;
    }
    free(result_1);
    //Converting the spectrum to time domain to see it's resolution
    //Function to perform IFFT
    gsl_fft_complex_radix2_backward((gsl_complex_packed_array) U_final_result,1,N_tota
    //positioning the spectra symmetrical to zero
    //Multiplying it's time domain with exp(-j*2*pi*fshift*t1) to shift
the spectra
    // to left
    i =1; // single combined spectrum
    fshift_hz = ((radar.N_steps+1.0)/2.0-i)*radar.delta_f;
    for (j = 0; j<N_total;j++)
    {
    ftmpVal2= 2*PI*fshift_hz*t1[j];
    cVal1.real = cos(ftmpVal2);
    cVal1.img = -sin(ftmpVal2);
    cVal2.real = U_final_result[j].real;
```

```
cVal2.img = U_final_result[j].img;
complex_mult(cVal1,cVal2,&cResultVal2);
u_final[j].real = cResultVal2.real;
u_final[j].img = cResultVal2.img;
}
free(U_final_result);
//convert to frequency domain
gsl_fft_complex_radix2_forward((gsl_complex_packed_array) u_final,1,N_total);
//converting to time domain to see the final resolution
gsl_fft_complex_radix2_backward((gsl_complex_packed_array)u_final,1,N_total);
/**********************send the processed data back to master**********/
msgtype = FROMSLAVE_MSG;
rc = pvm_initsend(PvmDataDefault);
rc = pvm_pkint(&index, 1, 1);
rc = pvm_pkint(&N_total,1,1);
rc = pvm_pkdouble((double*)u_final,2*N_total,1);
rc = pvm_send(masterid, FROMSLAVE_MSG);
free(u_final);
}
// CHECK FOR REMAINDER PROFILES
// Each worker needs to process only one more profile.
//Receive real data from master
msgtype = FROMMASTER_MSG;
rc = pvm_recv(masterid, FROMMASTER_MSG);
rc = pvm_upkint(&index, 1, 1);
if (index > 0)
{
rc = pvm_upkint(&NumElements, 1, 1);
rc = pvm_upkfloat(Fsamples1, NumElements, 1);
result_1 =(_complex_data*)calloc(sizeof(_complex_data),N_total);
//Processing high resolution profile
for(i =0;i<radar.N_steps*p;i++)
{
NewSamples1[2*i] = 0.0;
NewSamples1[2*i+1] = 0.0;
```

```
}
for(i =0;i<radar.N_steps;i++)
{
for(j = 0;j<radar.N_samples;j++)
{
NewSamples1[(i*p*2)+(2*j)] =Fsamples1[(i*radar.N_samples*2)+(2*j)];
NewSamples1[(i*p*2)+(2*j+1)] =
Fsamples1[(i*radar.N_samples*2)+(2*j+1)];
}
}
//Function to perform FFT
for(i=0;i<radar.N_steps;i++)
{
gsl_fft_complex_radix2_forward(&NewSamples1[i*p*2],1,p);
}
for (i = 0; i < radar.N_steps; i++)
{
for(j = s_chirp/4; j < p-(s_chirp/4-1); j++)
{
NewSamples1[(i*p*2)+(2*j)] = 0;
NewSamples1[(i*p*2)+(2*j+1)] = 0;
}
}
U_final_result = (_complex_data*)calloc(sizeof(_complex_data),N_total);
u_final = (_complex_data*)calloc(sizeof(_complex_data),N_total);
if (!U_final_result || !u_final )
{
printf("Memory Allocation Error\n");
exit(1);
}
//creating a vector to store shifted spectra from real target
V = (_complex_data*)calloc(sizeof(_complex_data),N_total*radar.N_steps);
if (!V)
{
printf("Memory Allocation Error\n");
```

```
    exit(1);
    }
    //copying the elements from 1st row
    for(i= 0; i<s_chirp/2; i++)
    {
    V[i].real = NewSamples1[2*i];
    V[i].img = NewSamples1[2*i+1];
    }
    for(i= 0; i<s_chirp/2; i++)
    {
    V[(r - s_chirp/2) + i].real = NewSamples1[2*((p - s_chirp/2) + i)];
    V[(r - s_chirp/2) + i].img = NewSamples1[2*((p - s_chirp/2) + i) +
1];
    }
    //sum = V(1,:)
    //This is a complex addition
    for(l=0; l < 1; l++)
    {
    for(j =0; j < N_total; j++)
    {
    sum[j].real = V[l*r + j].real;
    sum[j].img = V[l*r + j].img;
    }
    }
    //copying the elements from other row
    for(i = 1 ;i<n ;i++)
    {
    fshift = i * radar.number_shift;
    Vptr = &V[fshift];
    // Point to ith Fsamples1[ ] 1024 samples
    fptr = &NewSamples1[2*p*i];
    //printf("I = %d\n",i);
    for(j = 0;j < s_chirp/2;j++)
    {
    //V(i,fshift-s_chirp/2:fshift)= profiles_t(i,p-s_chirp/2:p);
```

```
Vptr[(-s_chirp/2) + j].real = fptr[2*((p - s_chirp/2) + j)];
Vptr[(-s_chirp/2) + j].img = fptr[2*((p - s_chirp/2) + j) +1];
//V(i,fshift+1:fshift+s_chirp/2) = profiles_t(i,1:s_chirp/2);
Vptr[j].real = fptr[2*j];
Vptr[j].img = fptr[2*j + 1];
}
// sum =sum+V(i,:);
for(j = -s_chirp/2; j < s_chirp/2; j++)
{
sum[j+fshift].real += Vptr[j].real;
sum[j+fshift].img += Vptr[j].img;
}
}
// copying sum array onto result_1 and U_calibration
for(j = 0; j < N_total; j++)
{
result_1[j].real = sum[j].real;
result_1[j].img = sum[j].img;
}
free(V);
//Applying the compensation filter to the target spectrum
for (j = 0; j < N_total; j++)
{
cVal1.real = result_1[j].real;
cVal1.img = result_1[j].img;
cVal2.real = U_compensation[j].real;
cVal2.img = U_compensation[j].img;
complex_mult(cVal1,cVal2,&cResultVal1);
// copying the result of the multiplication to the array U_final_result
U_final_result[j].real = cResultVal1.real;
U_final_result[j].img = cResultVal1.img;
}
free(result_1);
//Converting the spectrum to time domain to see it's resolution
//Function to perform IFFT
```

```
gsl_fft_complex_radix2_backward((gsl_complex_packed_array) U_final_result,1,N_tota
//positioning the spectra symmetrical to zero
i =1; // single combined spectrum
fshift_hz = ((radar.N_steps+1.0)/2.0-i)*radar.delta_f;
for (j = 0; j<N_total;j++)
{
ftmpVal2= 2*PI*fshift_hz*t1[j];
cVal1.real = cos(ftmpVal2);
cVal1.img = -sin(ftmpVal2);
cVal2.real = U_final_result[j].real;
cVal2.img = U_final_result[j].img;
complex_mult(cVal1,cVal2,&cResultVal2);
u_final[j].real = cResultVal2.real;
u_final[j].img = cResultVal2.img;
}
free(U_final_result);
//convert to frequency domain
gsl_fft_complex_radix2_forward((gsl_complex_packed_array) u_final,1,N_total);
//converting to time domain to see the final resolution
gsl_fft_complex_radix2_backward((gsl_complex_packed_array)u_final,1,N_total);
/**********************send the processed data back to master**********/
msgtype = FROMSLAVE_MSG;
rc = pvm_initsend(PvmDataDefault);
rc = pvm_pkint(&index, 1, 1);
rc = pvm_pkint(&N_total,1,1);
rc = pvm_pkdouble((double*)u_final,2*N_total,1);
rc = pvm_send(masterid, FROMSLAVE_MSG);
free(u_final);
}
/****************Exit PVM**********/
pvm_exit();
return(0);

}// End of Main
```

Bibliography

[1] Sandia National Laboratories, http://www.sandia.gov/radar/whatis.html

[2] Hovland, H. A., Johannassen, J. A. and Digranes, G. (1994), Slick Detection in SAR Images, *Proc. IEEE geosci. Remote Sensing Symp., IGARSS'94, Pasadena*, USA, pp. 2038-2040.

[3] G. J. Lynne and G. R. Taylor (1986), Geological Assessment of SIR-B Imagery of Amadeus Basin, N. T., Australia, *IEEE Transactions on Geo science and Remote Sensing* 24(4): 575-581

[4] M. R. Ingss (1996), The SASAR VHF Sensor, *Proc. European Conference on Synthetic Aperture Radar, EUSAR'96*, Konigswinter, Germany, pp. 317-320.

[5] J. M. Horrel (1999), *Range-Doppler Synthetic Aperture Radar Processing at VHF Frequencies*, PhD thesis, University of Cape Town.

[6] R. T. Lord (2000), *Aspects of Stepped-Frequency Processing for Low-Frequency SAR Systems*, PhD thesis, University of Cape Town.

[7] J. M. Horrell (1997) *Selected SAR Topics*, Lecture note, University of Cape Town, Department of Electrical Engineering, Cape Town..

[8] R. G. White, A. Blake, A. M. Horne (1996), Defence Reseach Agency, Malvern, UK; *Very high resolution SAR data, Proc. European Conference on Synthetic Aperture Radar, EUSAR'96*, Konigswinter, pp. 393-396.

[9] D. R. Wehner (1995), *high resolution Radar*, 2nd edn, Artech House, Norwood, MA.

[10] A. J. Wilkinson (1996), Notes on Synthetic Range Profiling, *Technical report*, University of Cape Town, Department of Electrical Engineering, Cape Town.

[11] R. T. Lord and M. R. Ingss (1996), High Resolution VHF SAR Processing Synthetic Range Profiling, *Proc. IEEE Geosci. Remote Sensing Symp., IGARSS'96*, Vol. 1, Lincoln, Nebraska, pp. 454-456.

[12] R. T. Lord and M. R. Ingss, *High Range Resolution Radar using Narrowband Linear Chirps offset in Frequency, Proc. IEEE South African Symp.on Communications and Signal Processing,* COMSIG'97, Grahamstown, South Africa, pp. 9-12, September 1997, comsig97.pdf.

[13] T. H. Einstein (2000), *Impact of New Microprocessor and Dram memories on Real-Time Airborne SAR Image Formation,* Proc. *European Conference on Synthetic Aperture Radar, EUSAR'2000,* Munich, Germany pp. 583-590.

[14] M. R. Ingss, M. Gebhardt, G. Pollock (2000), *Beowulf Computational Techniques Applied to SAR Processing*, Proc. *European Conference on Synthetic Aperture Radar, EUSAR'2000,* Munich, Germany, pp. *189-192.*

[15] T. G. H. Bennett (2001), *Development of a Parallel SAR processor on a Beowulf Cluster,* MSc. thesis, University of Cape Town.

[16] S. Kingsley, and Quegan, S. (1999), *Understanding Radar Systems,* SciTech Publishing, Mendham.

[17] M. I. Skolnik (1990), *Radar Handbook,* 2nd edn, McGraw-Hill, New York.

[18] C. Elachi (1998,) *Spaceborne Radar Remote Sensing: Applications and Techniques*, IEEE Press, New York.

[19] A. J. Wilkinson (2001), *Notes on Radar Signal Processing*, University of Cape Town, Department of Electrical Engineering, Cape Town.

[20] L. Litwin (2001), Matched filtering and timing recovery in digital receivers *http:// www.rfdesign.com*

[21] D. L. Mensa (1991), *High Resolution Radar Cross-Section Imaging*, Artech House, Inc., Norwood, MA.

[22] Yonghong Huang, Zuomin Ma, Shiyi Mao (1996), Beijing University of Aeronautics and Astronautics, *Stepped-Frequency SAR System Design and Signal Processing, EUSAR'96*, Konigswinter, pp. 565-568.

[23] A. J. Wilkinson, R. T. Lord and M. R. Inggs (1998), Stepped-Frequency Processing by Reconstruction of Target Reflectivity Spectrum, *IEEE Proc. of the South African Symp. on Communications and Signal Processing, COMSIG'98,* Cape Town, South Africa, pp. 101-104.

[24] L. M. H. Ulander (1998), Performance of Stepped-Frequency Waveform for Ultra-Wideband VHF SAR, Proc. EUSAR'98, Friedrichshafen, Germany, pp.323-326

[25] L. M. H. Ulander and Frolind P.-O.(1998), Precision Processing of CARABAS HF/VHF-band SAR Data, *Proc. IEEE IGRASS'99,* Vol. 1, Hamburg, Germany.

[26] R. Jain, Survey on distributed computing networks -networks of workstations. *http://www.cis.ohio.state.edu.*

[27] S. Ragsdale (1992), *Parallel Programming*, McGraw-Hill, Inc.

[28] Demmel, Cs267: lecture 9 part1 - a closer look at parallel architectures. http:// www.cs.berkley.edu/demmel/Cs267, 1996.

[29] B. Wilkinson, M. Allen, *Techniques and Application Using Workstations and Parallel Computers*, Prentice Hall Inc., 1999 http://www.cs.uncc.edu/~abw/CSCI5145/

[30] H. Dietz. Linux parallel processing using clusters. http://www.yara.ecn.purdue.edu, 1997.

[31] A. Geist, A. Beguelin, J. Dongara...W. Jiang, *Parallel Computing*, MIT press, Massachusetts.

[32] A. Downton and D. Crookes (1998), parallel architectures for image processing. *Electronics and Communication Engineering Journal, Vol 10.*

[33] PVM basics, http://golum.riv.csu.edu.au/~ialtas/module2/section_3.html

[34] G. A. Geist, J. A. Kohl (1996), PVM and MPI: a Comparison of Features.