SAR PROCESSING USING PVM

Shirley Lynne Wuyts

A project report submitted to the Faculty of Engineering, University of Cape Town, in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Cape Town, 1997

DECLARATION

I declare that this project report 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.

_____.

Shirley Lynne Wuyts

_____. day of _____. 199__.

ABSTRACT

This thesis explores various methods of using PVM (Parallel Virtual Machine) to improve the speed of processing a SAR (Synthetic Aperture Radar) image.

A network of heterogeneous machines were set up as the basis of the parallel virtual machine. SAR processing software was written for testing the PVM.

The software performed simplified range and azimuth compression on simulated SAR images of a point target. The theory and results were examined as part of the thesis. Complications such as range curvature, range migration and range dependent focusing were not addressed.

Tests were performed by running the compression program on this PVM system to gauge the performance of processing SAR images. Certain factors that affect the performance were taken into account and investigated. These factors were the task granularity, the total number of messages conveyed, the number of slaves, system resources, the method of parallelism, the network considerations, load balancing and the time spent on specific functions.

The tests proved that PVM improves the time taken to process a SAR image once these factors are optimised.

ACKNOWLEDGEMENTS

I wish to express my gratitude and appreciation to those who have contributed either directly or indirectly to this thesis project.

Thanks to Professor M. Inggs, J. Horrell, R. Lengenfelder and C.Fulton.

Special thanks to D. Coetzee.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Background

This thesis explores various methods of using PVM (Parallel Virtual Machine) to optimise the speed of processing a SAR (Synthetic Aperture Radar) image.

Processing SAR images is a computationally intensive exercise. PVM is investigated here for speeding up this process as PVM provides the means to process any application in parallel over a network of machines.

1.2 Objectives

The aim of this thesis is to demonstrate that the use of PVM does indeed improve the time taken to process SAR images.

1.3 Procedure

SAR image processing software was designed for this thesis. The software consists of three programs to perform range compression, matrix corner-turning and azimuth compression respectively. See Figure 1 for the procedure used to run these programs.

The two compression programs are the programs used for testing PVM. Each compression program has a master which sets up the PVM environment, starts a specified number of slaves and manages the processing work of each. The slaves perform the actual compression but on small sections of the data thereby minimising the processing load.

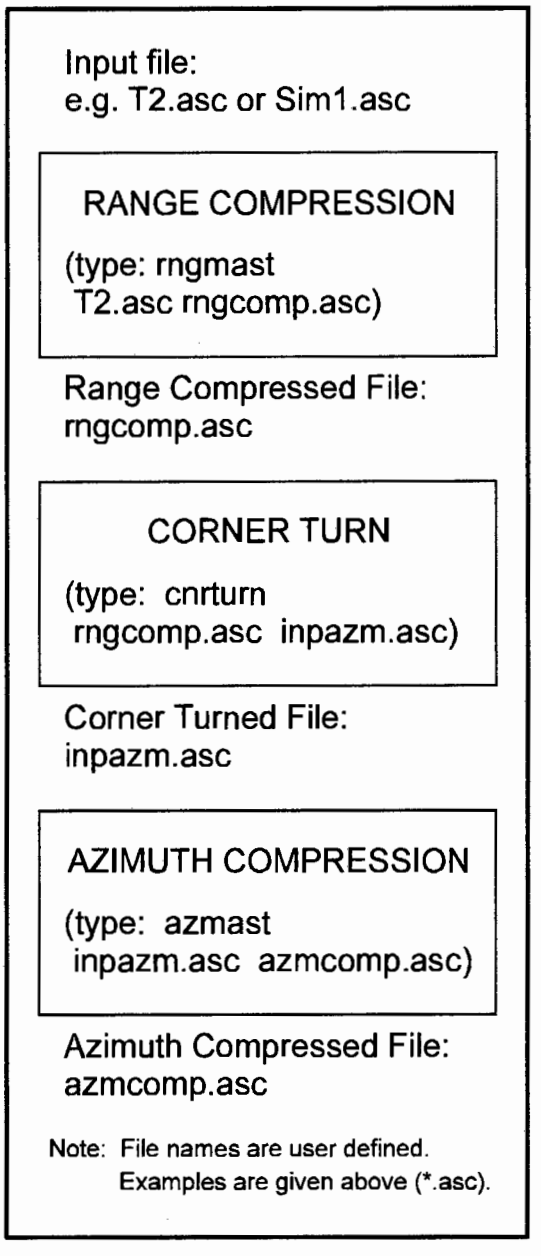


Figure 1: Procedure for running the PVM compression programs

1.4 Scope

This document introduces the subject, explains the theory, methods and results both of the PVM tests and of the SAR image processing software then provides conclusions to the investigation.

1.5 Plan of development

Chapter 2 **PARALLEL VIRTUAL MACHINE** discusses the background and general details to creating and using a parallel virtual machine.

Chapter 3 **RANGE COMPRESSION** explains the need and use for range compression of SAR images. It explains the assumptions and parameter values chosen for generating and processing the input files. It then goes on to explaining how the software is designed and compiled to perform the compression processing. The chapter runs through each method and shows the corresponding result for each step of the range compression process.

Chapter 4 **AZIMUTH COMPRESSION** explains the need and use for azimuth compression of SAR images. It also explains the need and use of the matrix corner turning process. It then goes onto explaining how the software is designed and compiled to perform the compression processing. The chapter runs through each method and shows the corresponding result for each step of the azimuth compression process.

Chapter 5 **TESTING PVM** explains and makes observations from the PVM implementation used in this case. The chapter shows the various tests performed on the SAR compression software in PVM. It explains the test procedures, shows the corresponding result and draws conclusions from each test.

Chapter 6 **CONCLUSIONS** discusses the conclusions drawn from the results of the thesis and suggests future actions to further the use of PVM for SAR image processing.

2. PARALLEL VIRTUAL MACHINE

2.1 How PVM Works

2.1.1 Description

PVM (Parallel Virtual Machine) **[7]** is a software system that enables a user-configurable pool of heterogeneous computers to be used as a unified and flexible concurrent computational resource.

2.1.2 Major Features

The major features of PVM are:

- Portable - PVM runs on many machines using many different architectures, plus many shared and distributed memory multiprocessors.
- Heterogeneous - Any type of machines can be combined in a single virtual machine.
- Scaleable - Virtual machines can include hundreds of host computers and run thousands of tasks.
- Dynamic configuration - Computers can be dynamically added and deleted from the PVM by the application or manually.
- Hooks for fault tolerance - Application can be notified via messages of lost processes, processors, or addition of new resources.
- Dynamic process groups - User-defined process grouping for functions like broadcast to allow single messages to multiple processes. Groups can overlap and change dynamically during an application.
- Signals - PVM tasks can send signals to other tasks.

- Multiple message buffers - Allows easier development of PVM math libraries and graphical interfaces each communicating with a separate buffer stream.
- Tracing - Call-level tracing built into PVM library.
- Can be customised - Users can write manager tasks to implement custom scheduling policies.

2.1.3 Applications

Applications where PVM has been used include molecular dynamics simulations, superconductivity studies, distributed fractal computations and matrix algorithms.

2.1.4 PVM History

PVM was publicly released in March 1991 and has gone through a number of updates. The version of software used in this thesis was version 3.3.1. PVM was produced by the Heterogeneous Network Project - a collaborative effort by researchers at Oak Ridge National Laboratory, the University of Tennessee and Emory University specifically to facilitate heterogeneous computing.

PVM was one of the first software systems to enable machines with widely different architectures and floating point representations to work together on a single computational task. PVM can be used on its own or as a foundation upon which other heterogeneous network software can be built.

PVM is a continually evolving experimental research project. Several enhancements are in progress, foremost of which is the development of a generalised distributed computing (GDC) layer for PVM.

The GDC layer will support:

1) parallel input-output with enhanced filesystem semantics for shadowing, interleaved access, and rollback. Shadowing will allow multiple disks to be kept in step with each other by receiving the same I/O operations, interleaved access will allow performance enhancements similar to RAID technology where multiple disk spindles are used to spread each I/O operation and rollback is a form of journalling to allow changes to be undone if errors occurred.

2) access control, authentication and abstract mutual exclusion mechanisms. The abstraction of the mutual exclusion mechanism will hide the implementation details from the user.

3) support for the client-server model of distributed computing, with facilities for the transparent exporting and invocation of services allowing client machines to call services and not care where they are actually executed.

4) distributed transaction processing primitives.

Early results from **[21]** indicate that these facilities can be provided at high levels of efficiency and that the enhanced functionality will prove beneficial for many new classes of application domains.

To obtain a more detailed description of PVM's features, such as a copy of the PVM User's Guide or source code, simply send email to netlib@ornl.gov with the message "send index from pvm".

2.1.5 Environments

The machines used by PVM include:

- Shared or local memory multiprocessors
- Vector supercomputers
- Specialised graphics engines
- Scalar workstations

The PVM system components have been compiled and tested **[21]** and **[18]** on the architectures shown in Table 1.

2.1.6 Modes

Three different modes can be used by PVM when defining the use of tasks on the heterogeneous network:

- Transparent mode - Tasks are automatically executed on the most appropriate computer.
- Architecture-dependent mode - User specifies which type of computer is to run a particular task.
- Low-level mode - User may specify a particular computer to execute a task.

In all modes, PVM takes care of data conversions from computer to computer as well as low-level communication issues.

Table 1: Architectures tested with PVM 3

PVM runs on				
Software	Networks	Workstations	Operating Systems	Supercomputers
C	Ethernet	DEC	BSD	Alliant FX/8
C++	FDDI	IBM	System-5	BBN Butterfly TC2000
Fortran77		HP	Linux	Convex C2, Exemplar
		Silicon Graphics	MS Windows 95/NT (beta)	Cray C90, T-3D, YMP, Cray-2, Cray S-MP
		Sun		Intel Paragon, Hypercube, iPSC/860, iPSC/2 386
		NeXT		IBM SP-2, 3090
		Data General		TMC CM-2, CM-5
		386/486/586 Unix boxes		Kendall Square
		386/486/586 MS Windows 95/NT (beta)		Sequent
				Maspar
				Encore
				Fujitsu
				Stardent

2.1.7 PVM program installation and structure

The PVM package is small (about 1 Mbyte of C source code) and easy to install. It needs to be installed only once on each machine to be accessible to all users provided that it is installed in a public directory such as `/usr/local`. Moreover, the installation does not require special privileges on any of the machines and thus can be done by any user.

The PVM system is composed of two parts. The first is a daemon, called pvmd3, that executes on all the computers making up the virtual machine. Pvmd3 is designed so any user with a valid login can install this daemon on a machine. A user wishing to use PVM first configures a virtual machine by specifying a host-pool list, the daemons are started on each, and cooperate to emulate a virtual machine. The PVM application can be started from a shell command line prompt on any of these computers.

The second part of the system is a library of PVM interfaces. This library contains user callable routines accessible from C, C++ and FORTRAN for message passing, spawning processes, coordinating tasks, and modifying the virtual machine. Application programs must be linked with this library to use PVM.

2.1.8 PVM application calls

Below are a few application calls common to all PVM programs. These examples are written in C.

Include files

#include "pvm3.h" - header file required for each source code module

Tasks

pvm_mytid()	- enrolls the process in PVM.
pvm_exit()	- exits from the PVM environment.
pvm_spawn() slave task on	- spawns a requested number of copies of a the computers specified to PVM.

Sending messages

pvm_initsend()	- clears the default send buffer and specifies the message encoding.
pvm_pack**X**()	- packs the data of type **X** into the sending buffer.
pvm_send(**tid, msgtag**)	- send the message of ID **msgtag** to the process **tid**.
pvm_mcast()	- same as pvm_send but to multiple tasks specified in an array of tids.

Receiving messages

pvm_recv(**tid, msgtag**)	- receive a message of msgid **msgtag** from the process **tid**.
pvm_unpack**X**()	- unpacks the data of type **X** from the receiving buffer.

2.1.9 PVM implementation

1. Once the program has been written, compile the program separately for every architecture in your virtual machine using a command similar to this:

```
cc -L~/pvm3/lib/ARCH file.c -lpvm3 -o file
```

where **file** is the file to be compiled and **ARCH** is the architecture name of the computer.

2. After compiling, move the executable to the directory ~/pvm3/bin/**ARCH**.

3. Before running the program, start PVM.

4. Run the executable from the UNIX command line.

Example for a LINUX on Intel machine:

1. Compile the program test.c which contains PVM calls:

```
cc -L~/pvm3/lib/LINUX test.c -lpvm3 -o test
```

2. Now move the executable to the PVM executable directory:

```
mv test ~/pvm3/bin/LINUX/.
```

3. Before running the program, start PVM and then exit to UNIX leaving PVM running by typing quit (not halt):

```
pvm

quit
```

4. Run the executable from the UNIX command line:

```
test
```

5. Type halt to shut down PVM when finished:

```
halt
```

2.1.10 Performance Considerations

For PVM programs to fully exploit the strengths of different machines, the following considerations must be taken into account:

1. **Task granularity** - the ratio of the number of bytes received by a process to the number of floating point operations it does i.e. incoming message size vs. work done. Increasing the granularity will speed up the application but the tradeoff is a reduction in the available parallelism.

2. **The total number of messages sent** - sending a small number of large messages takes less time than sending a large number of small messages. This statement is a generalisation and the problem is really application specific. Some applications can overlap sending small messages with other computation.

3. **Functional parallelism vs data parallelism** - compare functional parallelism where different machines do different tasks based on their specific capabilities to data parallelism where the data is distributed to all of the tasks with similar operations in the virtual machine.

4. **Network considerations** - different computers have different processing power. Machines with the same initial processing power may be disadvantaged due to multitasking by others. Even in a dedicated networked environment, with no external use, raw performance is hard to measure since operating system activity, window and filesystem overheads, and administrative network traffic can influence results. The processing power available may change

dynamically based on machine loads. To combat these problems, load balancing should be applied.

2.1.11 Load Balancing

Load balancing enhances performance by ensuring that each task is doing its fair share of work.

1. **Static load balancing** - work is assigned in the beginning. The size and number of tasks can be varied depending on the processing power of a given machine. This scheme is effective on a lightly loaded network.

2. **Dynamic load balancing** - various methods are used depending on the specific application. Two of the most common are:

a) master/slave program where the master sends slaves jobs to do when they become idle. This method is not suited for applications which require task to task communication since tasks will start and stop at arbitrary times.

b) At some predetermined time all processes stop, the work loads are then reexamined and redistributed as needed.

2.1.12 Raw Communication Performance

The time taken for processes to exchange messages is dependent on several factors, including the host machines, network speeds and, most predominately, the message size.

Experiments were conducted by Oak Ridge National Laboratory **[21]** on unloaded workstations rated at 50 MIPs. Table 2 presents the data transfer times (milliseconds) for differing message lengths and network types.

Table 2: Data Transfer Times (milliseconds)

Message Length	0	128	512	1K	4K	16K	64K	1M
Ethernet	1.2	1.5	2.1	3.2	7.2	24.5	82.3	1211.1
FDDI	1.2	1.5	1.9	2.5	5.9	16.1	60.3	665.7

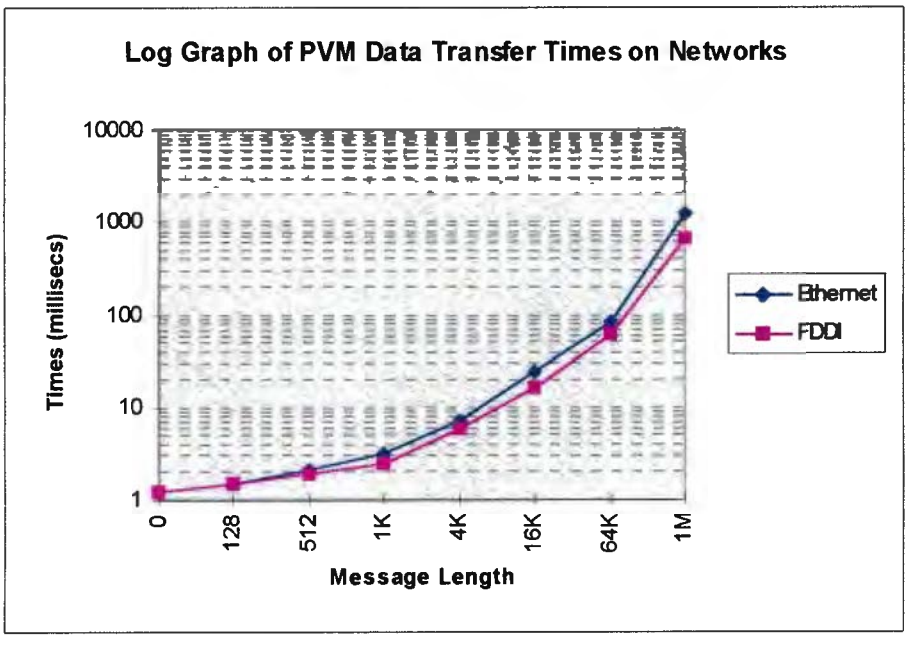


Figure 2: PVM Data Transfer Times on Networks

Two important observations were made from this data:

- There is a latency, or minimal time required to send a zero-length message, measured in the order of a millisecond. This factor depends largely on the speed of the host machines as the protocol processing made up a significant fraction of this time.

- Throughput showed the Ethernet network could be driven at almost peak capacity for large messages. Similar ratios are assumed to be

possible for fast networks with increases in host speeds and protocol optimisations.

With these performance considerations in mind we now turn to the actual testing done.

3. RANGE COMPRESSION

3.1 Why Perform Range Compression?

A real aperture radar determines target range by emitting brief intense rectangular pulses. The sampled returned signals are averaged over time intervals no shorter than the emitted pulses. In the representation given Figure 3, the Tx emissions would result in Rx returns that would be averaged over the pulse duration.

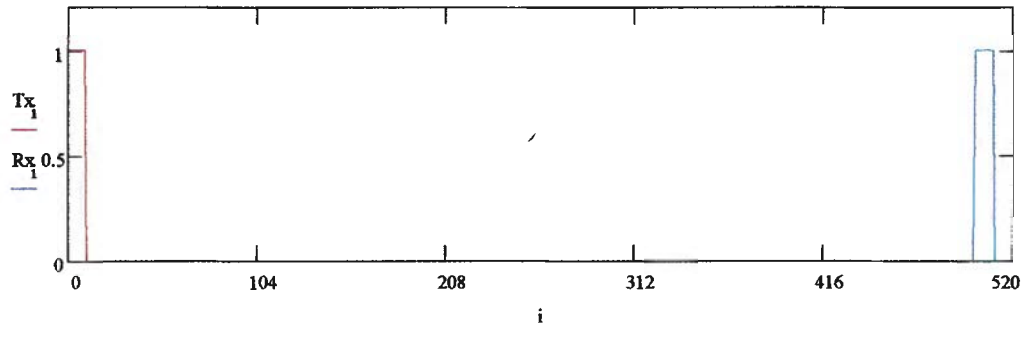


Figure 3: Transmit and Receive Pulses

Each averaged value is then the backscattered intensity from the surface at the slant range corresponding to half the round trip time of the signal. i.e.:

$$r = \frac{c \cdot t}{2}$$

Since the averaging interval is bounded below by the pulse width, the range resolution is directly proportional to the pulse width T, i.e.:

$$\Delta r \geq \frac{c \cdot T}{2}$$

In a simple pulse radar, the bandwidth B corresponds to the inverse of the pulse width T, so

$$\Delta r \geq \frac{c}{(2 \cdot B)}$$

High resolution requires short pulse width and, as energy is proportional to peak power,

$$E = P_{pk} \cdot T$$

high resolution requires very high intensity levels in order to obtain adequate energy in the return signal for typical radar remote sensor ranges. As a result the power requirements for orbiting SAR systems would appear to be high.

Linear Frequency Modulation pulse radar or "Chirp" radar was introduced by R.H.Dicke (patent 1945), in which pulses are frequency modulated thereby making bandwidth independent of pulse width. Large bandwidth and large pulse width make possible high resolution and high energy.

Although the returns from points at adjacent range intervals overlap in time, the frequency modulation of the pulse is distinctive enough for signal analysis to enable the components of the superimposed signals to be resolved. This is represented in Figure 4.

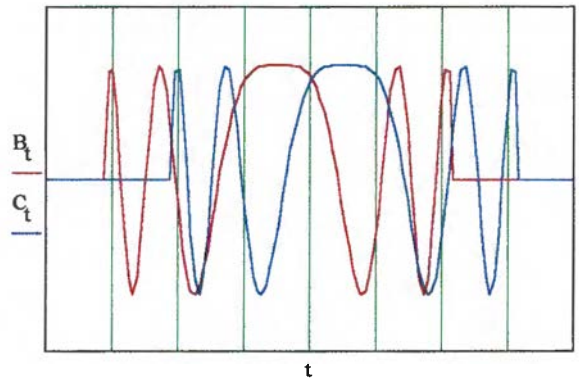


Figure 4: Received Pulses from Adjacent Ranges (real)

Upon reception, the pulses must be compressed to permit separation of adjacent range resolution cells. Two received expanded pulses overlap but, after compression, they are time-separated and can be resolved exactly as if a short pulse had been transmitted. This is represented in Figure 5.

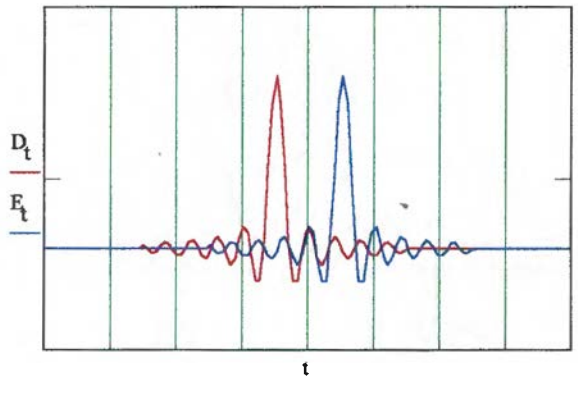


Figure 5: Compressed Pulses from Adjacent Ranges (real)

The range compressor is a form of *matched filter* or *correlator*. This method is such that when the returned echo correlates with the transmitted signal, a narrow pulse results at the lag corresponding to the round trip travel time. Thus we match the echo to the original pulse to the delay appropriate to the range of the target. This is described further in Hovanessian **[11]**.

3.2 The Range Compression Software

The Range Compression Software consists of the *pvmrngin.c*, *rng.c* and *rngcomp.c* files as shown in Figure 6.

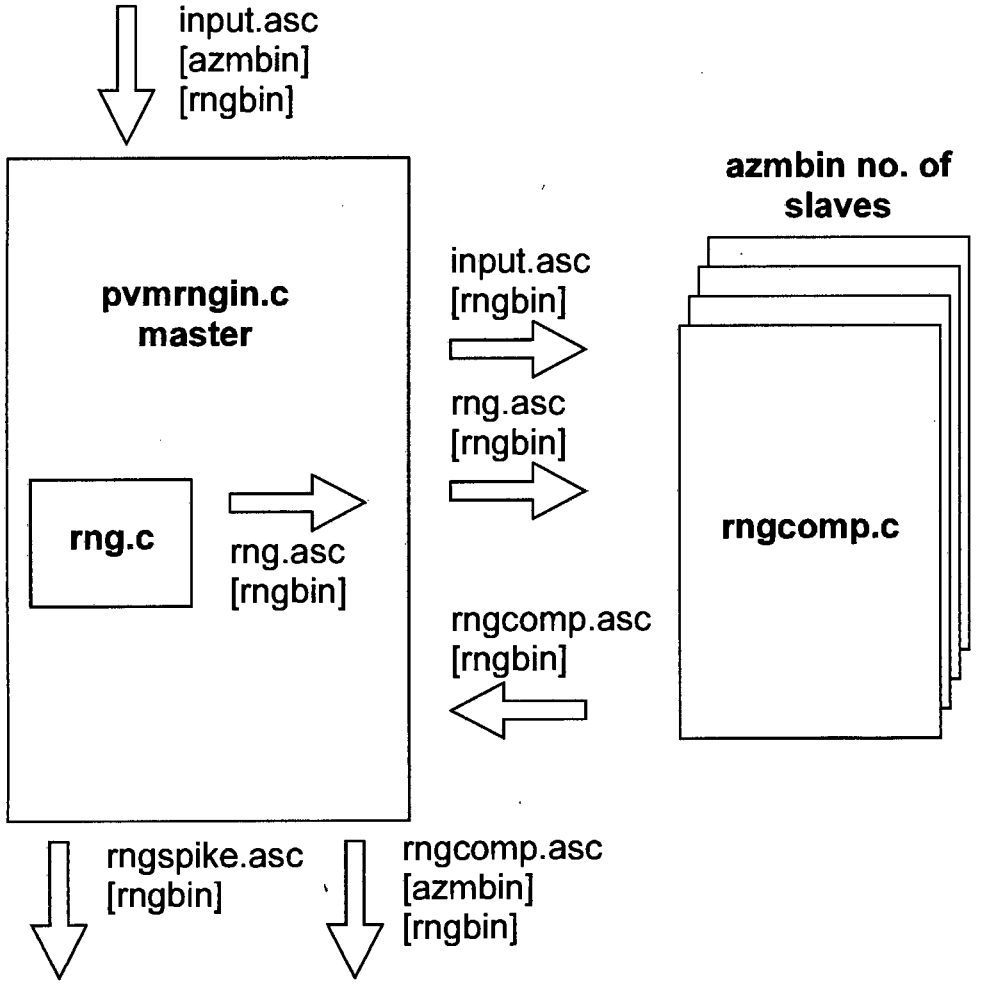


Figure 6: Range Compression Algorithm

3.3 The input files

The input files used for testing the thesis software are *T2.asc* and *sim1.asc* which are outputs from the radar simulators RADSIM **[8]** and SARSIM **[15]** respectively.

In the tests used here, the target is a stationary point target and the radar source travels past it in a straight line perpendicular to the target at a constant height, (see Figure 7).

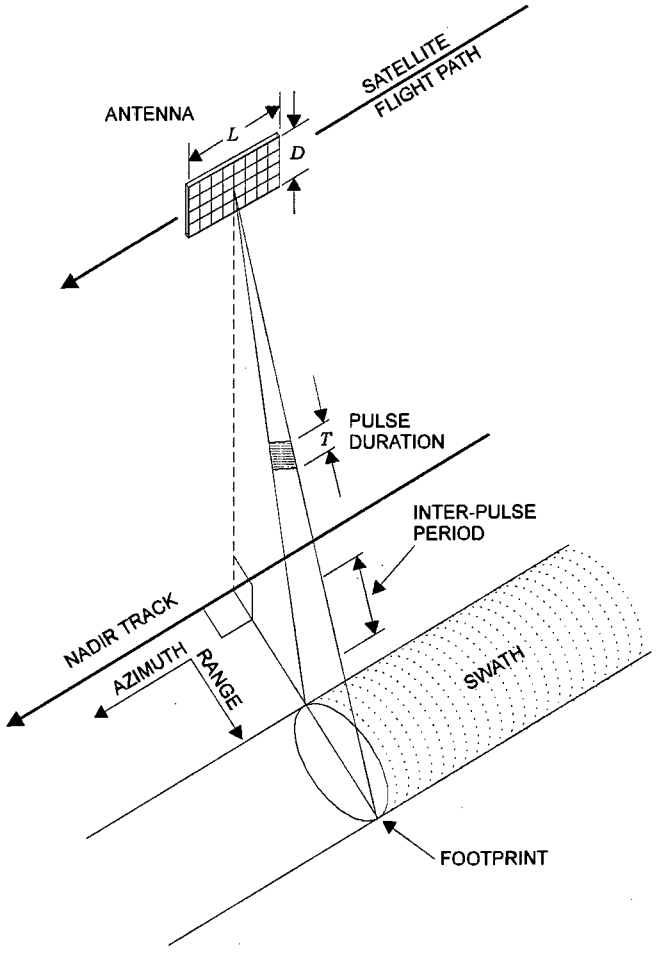


Figure 7: Radar Source travels perpendicularly past Stationary Target at constant height

Various factors affecting the processing such as antenna shape, beamwidth, STC and reflectivity have been excluded for simplicity.

The same waveform parameters were used to generate both files, see Table 3.

Table 3: Parameter Values used when generating and processing Test Files

Description	Parameter	Value	Units
Number of Range Bins	RngBin	256	Number
Number of Azimuth Bins	AzmBin	512	Number
Azimuth Filter Coefficients	AzmFilt	140	Number
Simulation Frequency	SimFreq	0.15	GHz
Pulse Width	PulseWidth	700.0	ns
Bandwidth	Bandwidth	0.1	GHz
Peak Amplitude	MaxAmpl	10.0	V
Centre Frequency	CentreFreq	0.141	GHz
Pulse Repetition Frequency	PRF	180.0	Hz
Target X (azimuth) Position	TargetX	0.0	m
Target Y (ground range) Position	TargetY	20000.0	m
Target Z (height) Position	TargetZ	0.0	m
Source Z (height) position	SourceZ	10 000.0	m
Source Velocity	SourceV	246.0	m/s
Range Offset	RngOffset	22300.0 (T2.asc) 22302.0 (Sim1.asc)	m
Transmission Power	TxPower	1.0	kW
Radar Cross Sectional Area	RCS	1.0	m^2

Losses (TxLoss+RxLoss+PropLoss)	Losses	0.0	dB
Antenna Gains	AntGain	7.0	dB
Amplification	Amplify	10 000 000.0	Number

The difference of the range offset value between the two input files *T2.asc* and *Sim1.asc* are caused by the coarse data capturing process used in SARSIM **[15]** but this problem is solved by the range offset value which realigns the source and target geometry.

The parameter values used are unrealistic but were chosen specifically for purpose of demonstration.

The file *pvmcomp.ini* contains these setup parameters read by the PVMCOMP software.

3.3.1 The pvmrngin.c file

The *pvmrngin.c* file is the PVM master which reads the range compression input file, runs *rng.c* to generate the matching reference waveform then generates PVM slaves to perform the range compression. *Pvmrngin.c* reads the file *pvmcomp.ini* for the setup parameters.

3.3.2 The rng.c file

The *rng.c* file is called by *pvmrngin.c* to generate the reference file *rng.asc*. *Rng.c* generates the expected range return pulse. For simplification, *rng.c* generates the transmitted pulse as it has similar frequency properties as the range returns and illustrates range compression equally well. Other factors affecting the return pulses such as antenna shape, beamwidth, STC and reflectivity have been excluded for simplicity.

Rng.c is given the transmitted waveform parameters from *pvmrngin.c*. To set the simulation framework, *rng.c* is given the simulation frequency and the number of range bins. The square waveform is described by the pulse width and amplitude and the chirp frequency from the chirp bandwidth.

The waveform is calculated by using a derivative of the equation:

$$Tx(t) := A\cdot\cos\left(\omega_0\cdot t + \frac{\mu\cdot t^2}{2}\right) \quad \text{where} \quad \mu := 2\cdot\pi\cdot\frac{B}{T}$$

B = chirp bandwidth,

T = pulse width

In this case, the signal is down-converted so that the equation is

$$Tx(t) := A\cdot\cos\left(\frac{\mu\cdot t^2}{2}\right)$$

The equation used in the simulation environment is

$$Tx(t) := A\cdot\cos\left[\frac{\mu\cdot\left(n - \frac{T}{2}\cdot simfreq\right)^2}{2}\right] \quad \text{where} \quad \mu := 2\cdot\pi\cdot\frac{\frac{B}{simfreq}}{T\cdot simfreq},$$

simfreq = simulation frequency, n = 0 to no. of range bins.

3.3.3 The rngcomp.c file

Each PVM slave is apportioned an azimuth vector of the input file. The slaves each run *rngcomp.c* to match the *rng.asc* file with the range return values at that azimuth, see Figure 8.

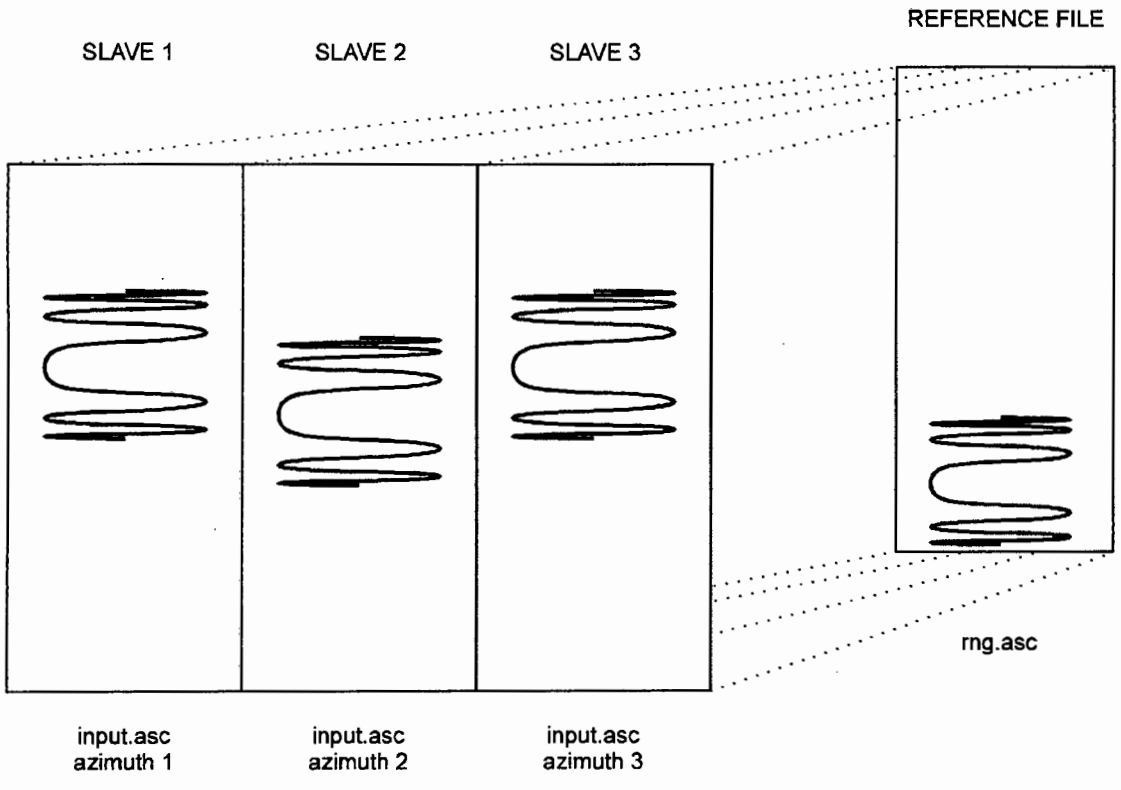


Figure 8: Range Compression Slaves

Rngcomp.c contains the range compression algorithm which scans range returns for chirp pulses having frequencies that match those expected. This is achieved by transforming the raw image data to the frequency domain, multiplying the frequency pattern with that of the *reference function (matched filter)*, then converting the result back to time domain[1]. The FFT process used is that from **[2]**.

The matching process matches the image from the start of the image vector to the end by assuming that the image before the start and after the end has the same pattern (infinite extent). To rule out this foldback operation from affecting the result, the reference and image vectors are

[1] See Appendix E **MATCHING THE CHIRP PULSE** which explains the theory of matching chirp pulses

expanded to double the RNGBIN length and this excess is padded with zeros. The Range Compression process returns the result having the original length which includes the resulting waveform in the case of *T2.asc* and *Sim1.asc* because the target is positioned within RNGBIN number of bins.

3.3.4 The rngspike.asc file

One of the range slaves generates a file *rngspike.asc* for use by the PVMCOMP Azimuth Compression software. The slave performs range compression by matching the reference range function to itself to provide a sample range compressed waveform.

3.3.5 Compiling the Range Compression Program

The executable programs can be compiled from the source code by typing **make** on the command line. This invokes the standard make utility included with the C compilers on the test machines and this utility processes the file **makefile**. This makefile defines the individual components of the executables and the order in which they are compiled. These are called dependencies. More information on the syntax of make files can be found by consulting the *man* pages on the individual systems. If invoked without any arguments the makefile is set up to create the compression executables, the corner turn executable and the slave programs. Typing **make help** will display the various arguments you may choose for selectively building various executables.

To execute the range compression program you must type:

```
rngmast inputfilename outputfilename
```

where **inputfilename** is the raw image data and **outputfilename** is the processed range compressed data.

Both input and output files are text files containing ASCII numbers written as floats of choice (nominally 6 digit) resolution. Each pair of numbers are separated from the next pair by a comma. A line break character may also be included. Each pair of numbers consists of the real and imaginary values per range bin, all per azimuth bin.

This ASCII format is chosen as it is the output of the radar simulators RADSIM **[8]** and SARSIM **[15]**. This method is very inefficient storage. Normally, IEEE floats would be utilised.

3.4 Program Methods and Results

The Range Compression Software is given the radar image as the range returns received versus azimuth.

The following figures[2] show the results of range compressing the input file *T2.asc* except where specified as compressing *sim1.asc*.

In the test examples used here the target is a stationary point target and the radar source travels past it in a straight line perpendicular to the target at a constant height.

The master divides the image into the range returns along each azimuth and supplies each to the slaves. The master also generates the reference range return for the matching of the chirp radar pulses.

The reference pulse is set up to be the transmitted pulse as it has the ideal chirp wave characteristics, as the radar return would be somewhat noisier and degenerated. The amplitude of the reference pulse is set to a constant square wave waveform, see Figure 9.

[2] All figures in this section show the actual values obtained from the test results

Although the pulsewidth equals 700 nanoseconds, it is used by the software as 105 units long by taking the simulation frequency of 0.15 GHz into account. The transmitted pulse centre is therefore at 52.5 units.

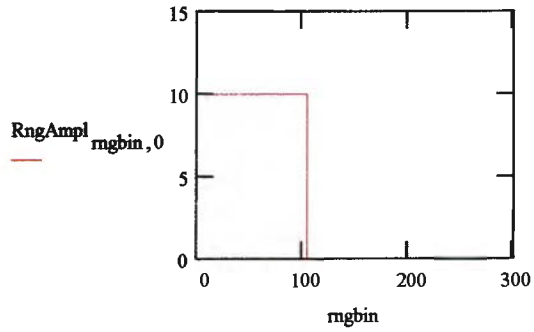


Figure 9: Transmitted Pulse Envelope

The real and imaginary transmitted chirp waveforms are shown in Figure 10. Note that the chirp pattern is symmetrical around the centre of the pulse.

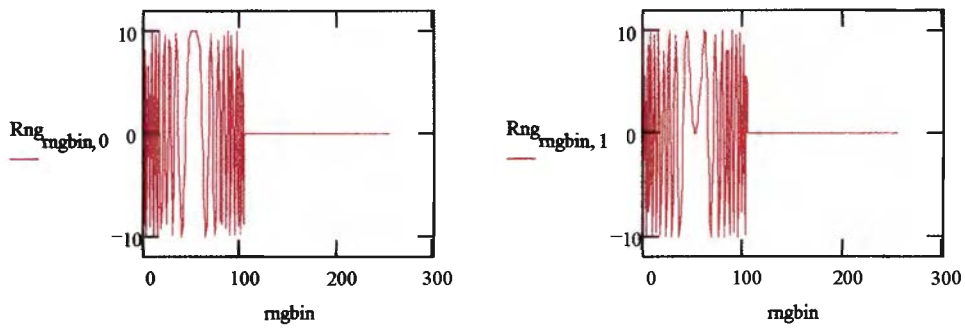


Figure 10: Reference Chirp (real & imag)

The return pulse at the exact azimuth of the point target, is shown in Figure 11. The return pulse centre is at 115.

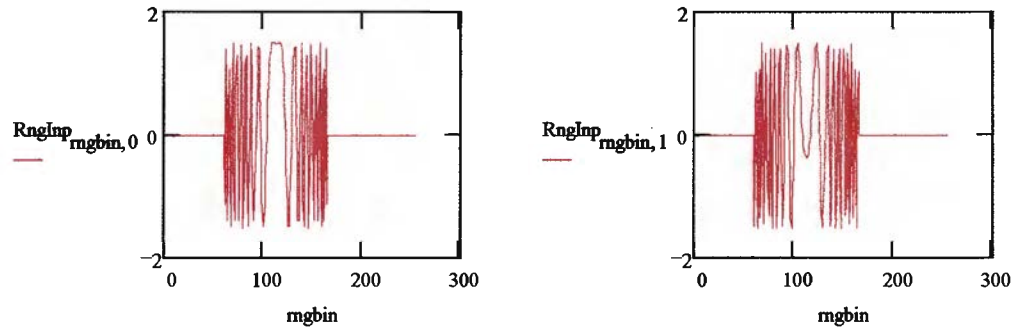


Figure 11: Input Chirp (real & imag)

The master shares out these range returns to the slaves to match with the reference pulse. An example of the corresponding real and imaginary waveforms are shown in Figure 12 which are matched by the slaves.

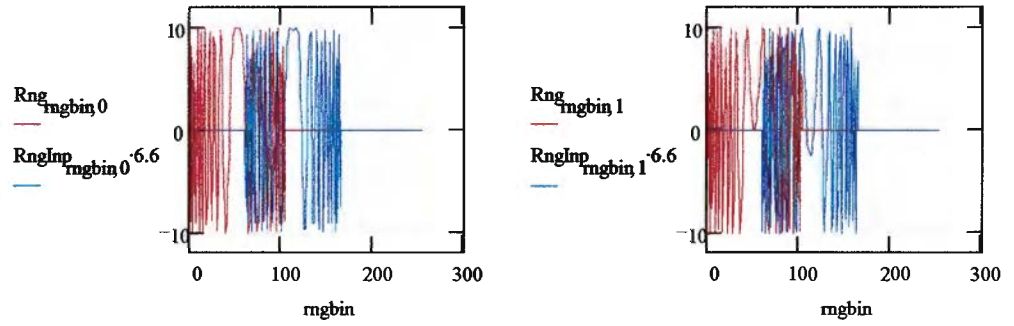


Figure 12: Reference and Input Chirps (real & imag)

The reference pulse is "windowed" using a Hamming window to reduce range sidelobes (at the expense of gain and resolution)[3]. The windowed reference function is shown in Figure 13.

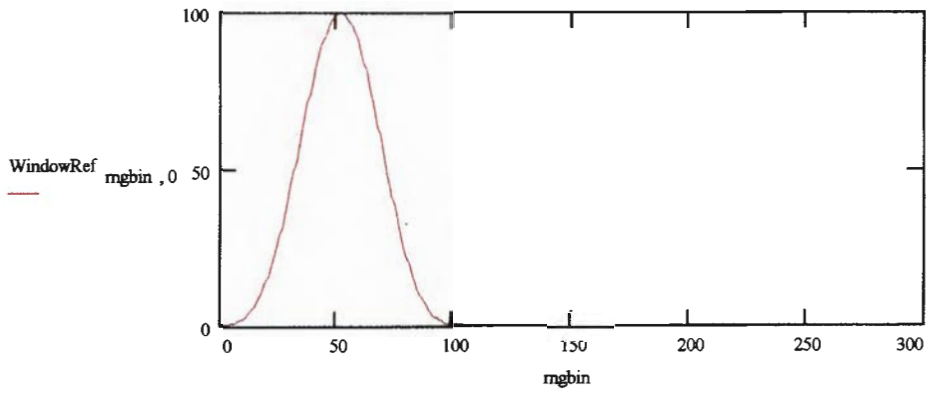


Figure 13: Windowed Reference Pulse (power)

The slaves match the pulses by converting the reference pulse and return pulse into the frequency domain. This is done by performing an FFT (Fast Fourier Transform) on the waveforms. The frequency description of the reference pulse is shown in Figure 14.

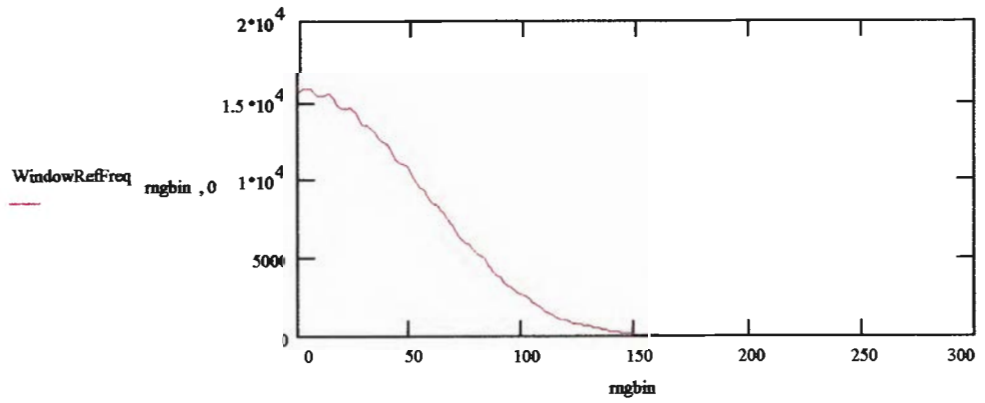


Figure 14: FFT of Reference Pulse (power)

[3] See Appendix E **MATCHING THE CHIRP PULSE** which explains the theory of matching chirp pulses

The frequency description of the input pulse is shown in Figure 15.

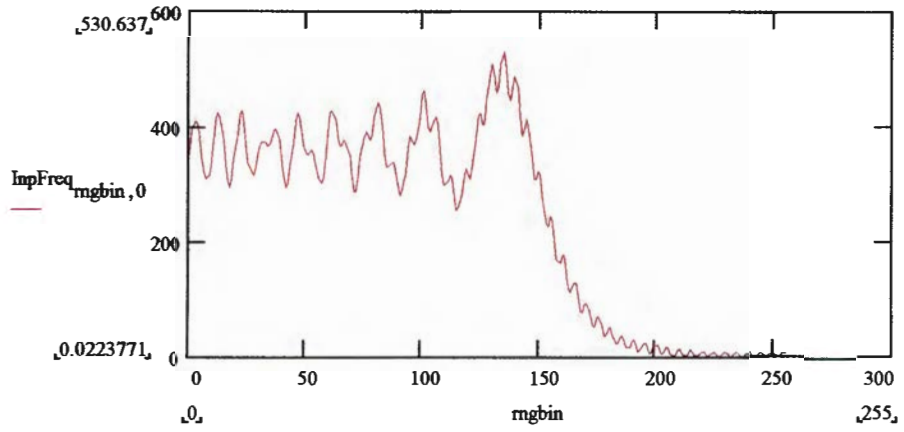


Figure 15: FFT of Input Pulse (power)

The frequency waveforms of the reference pulse and the input pulse are multiplied together to effectively correlate the two in terms of the time domain. The resulting waveform is shown in Figure 16.

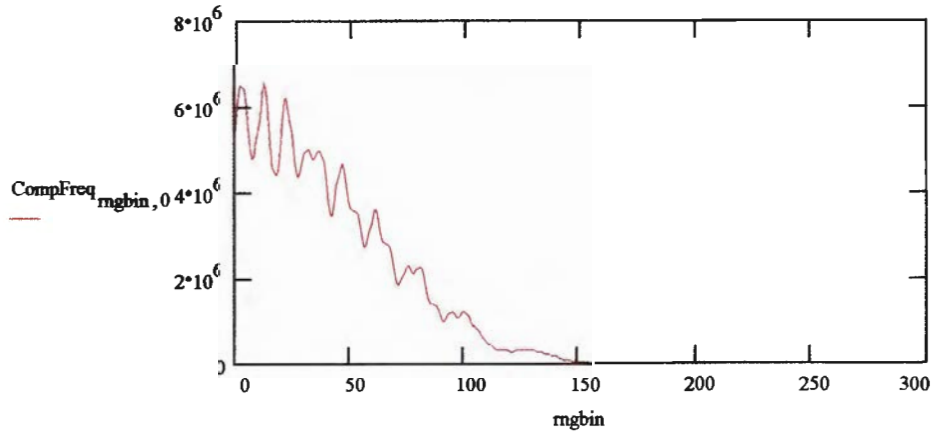


Figure 16: Multiplied FFT Pulses (power)

The resulting waveform is then transformed back to the time domain revealing the resulting compressed pulse, see Figure 17.

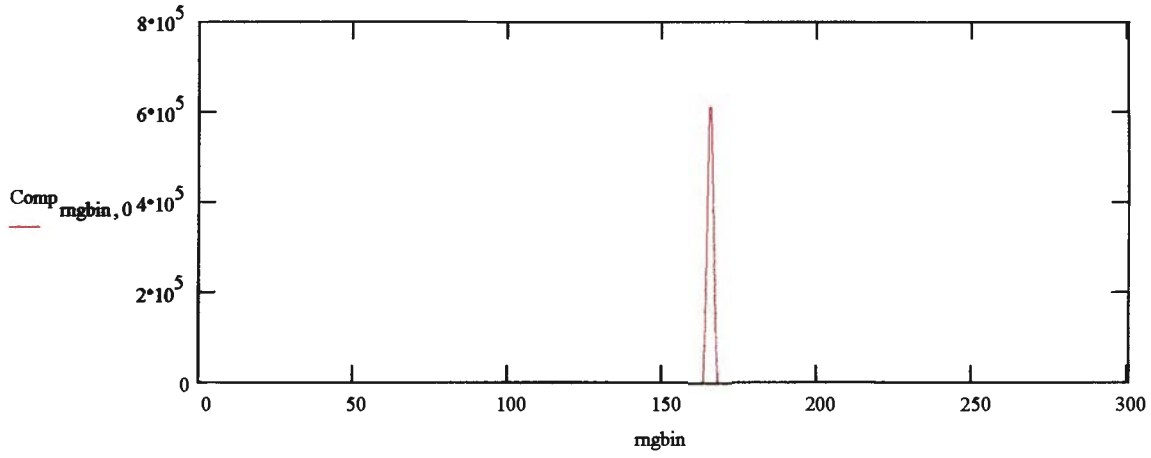


Figure 17: Range Compressed Pulse (power)

We expect the resulting waveforms to have the following characteristics:

1. The main lobe width should be approximately 3 units wide which is the simulated version of $\frac{2}{B} := 2 \cdot 10^{-8}$ where B is the chirp bandwidth.

2. The origin should be at approximately 167 units which is the difference of the two pulses (115 units - 53 units) plus the pulsewidth (105 units).

Figure 18 zooms in on the compressed pulse to show that our expectations are met satisfactorily.

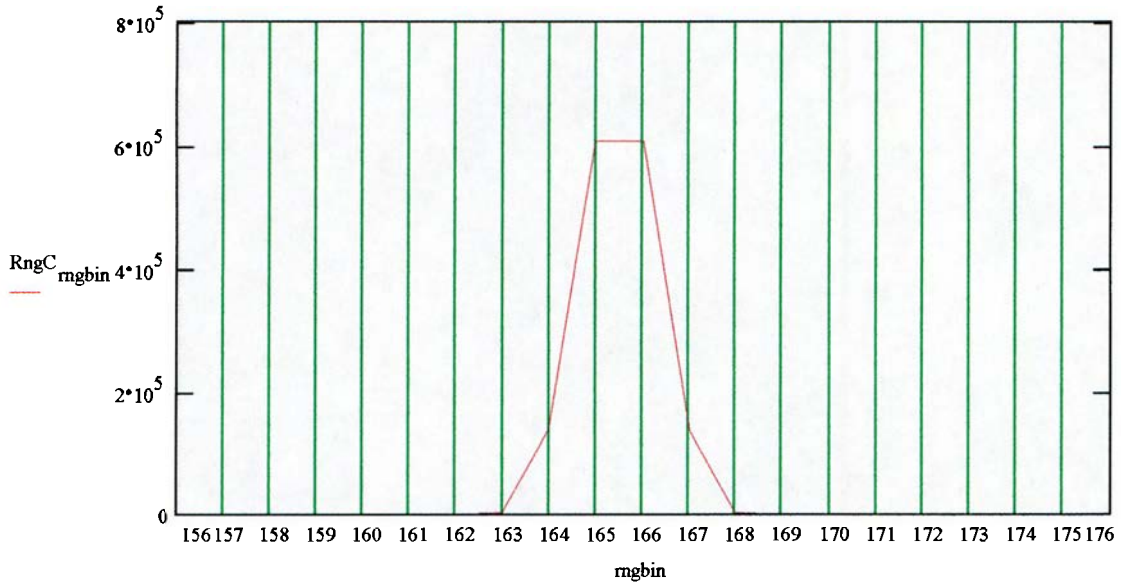


Figure 18: Range Compressed Pulse (power)

The *pulse compression ratio* is expected to be approximately 70 from either of 2 equations:

$$\frac{T}{t_0} := \frac{105}{1.5} \qquad T \cdot B := 700 \cdot 10^{-9} \cdot 100 \cdot 10^{6}$$

Figure 19 shows the envelopes of the input and resulting waveforms by combining the complex vectors of each signal. The figure shows this expected ratio to be true.

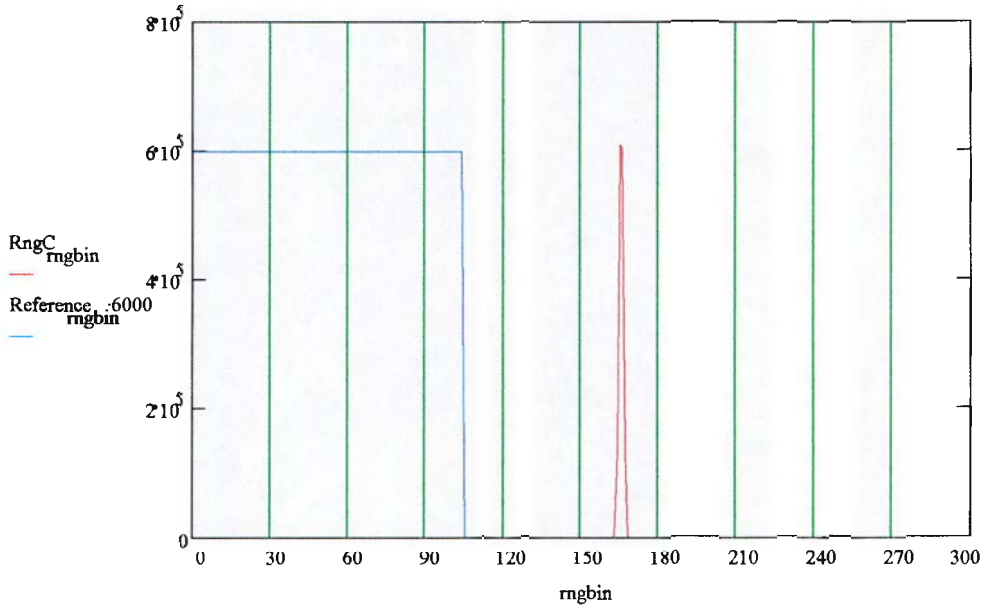
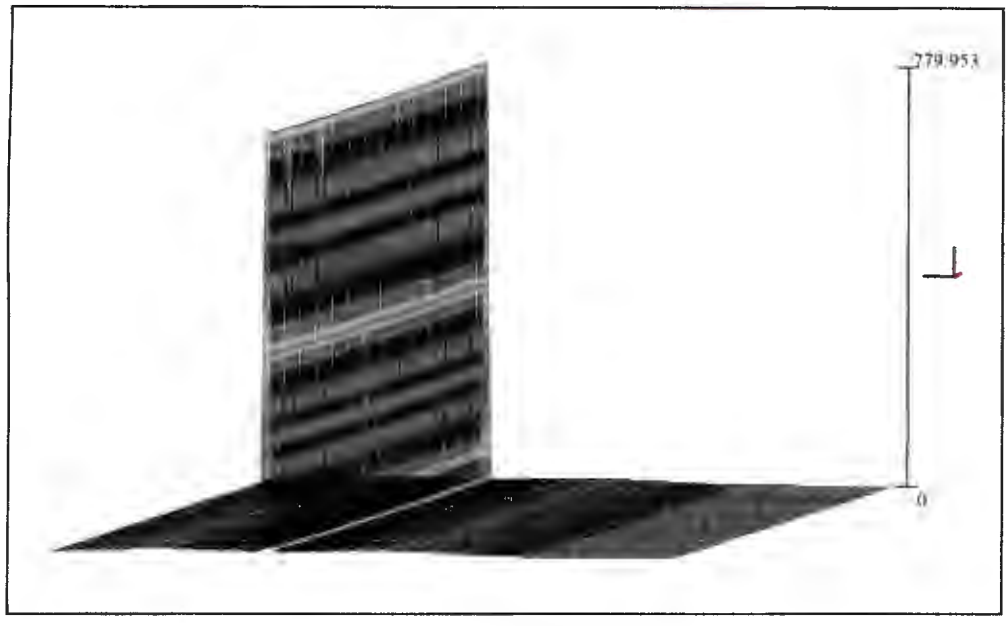


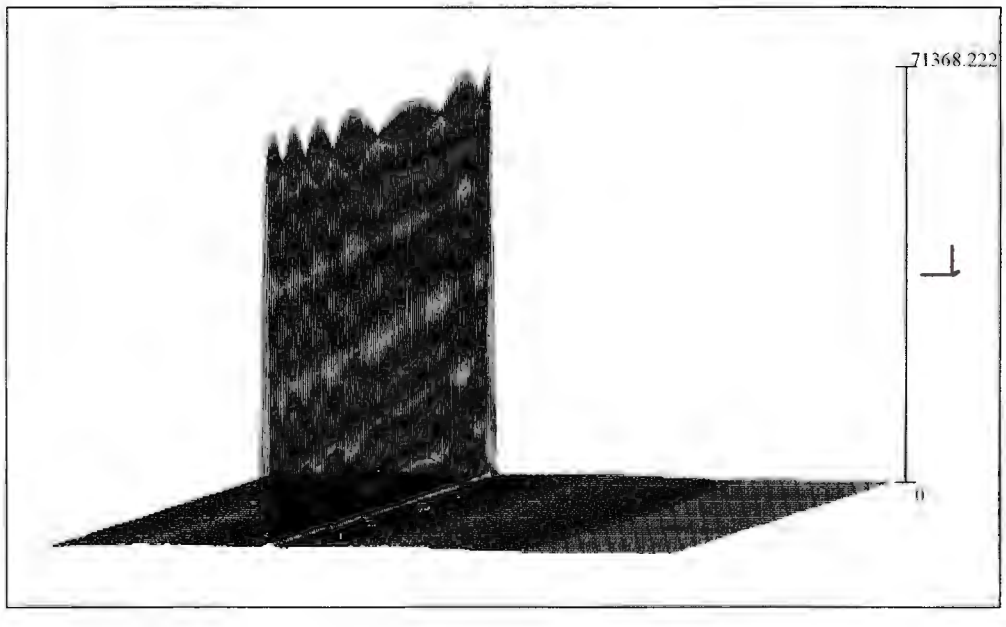
Figure 19: Envelopes of Range Compressed Chirp Pulse vs Transmitted Pulse

The return pulses return at different times depending on the range to the target at different azimuths. Each slave matches a different azimuth and, once the master combines the compression results, the result, as in Figure 20 and Figure 21, reveals a three dimensional arc around the target.



RngC2D

Figure 20: T2 Range Compression Arc (3D - range vs azimuth)



RngC2D

Figure 21: Sim1 Range Compression Arc (3D - range vs azimuth)

Seen from the top in Figure 22 the arc is quite visible.

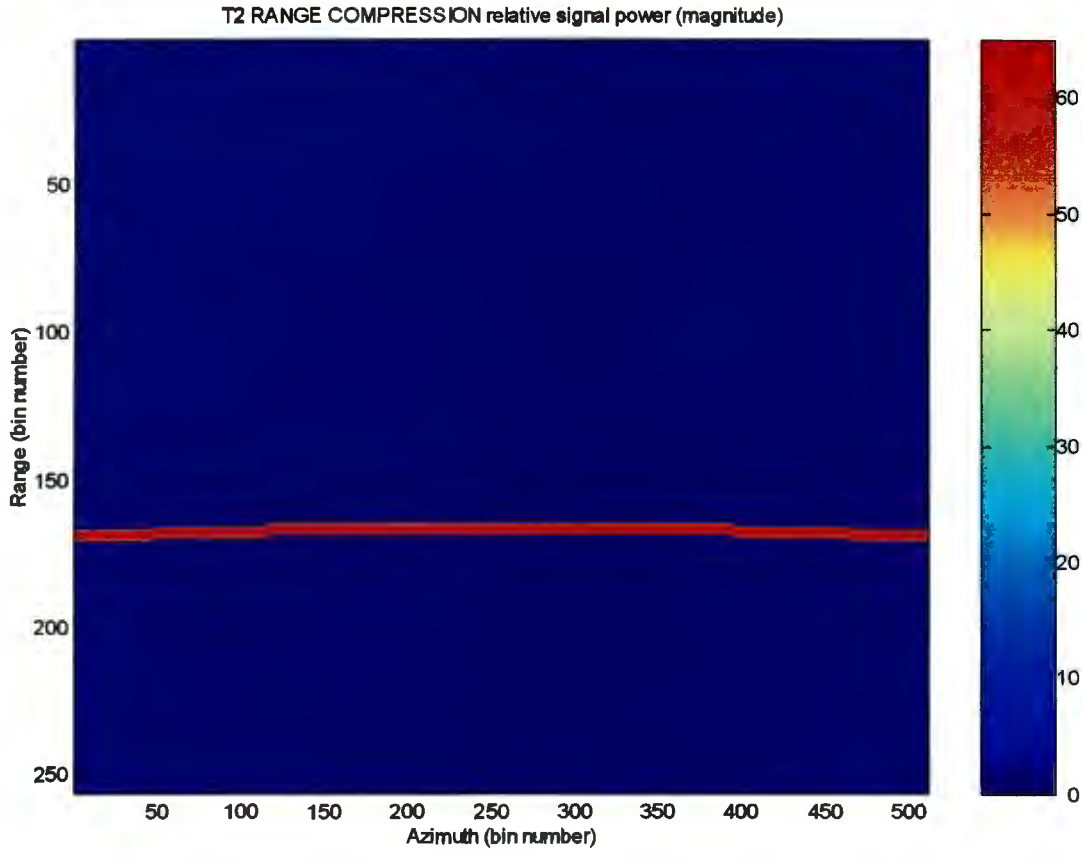


Figure 22: T2 Range Compression Arc (Top View - range vs azimuth)

Figure 23 shows the top view of the range compression of *sim1.asc*. The results are similar and, although not clearly visible, follow the same curve for *T2.asc* (see Figure 22) and *sim1.asc* (see Figure 23).

The position of the arc differs due to the capturing process of the three files. The three files were captured with different geometries to simulate different positions of the point target.

The position of the target in the azimuth direction has no effect on the range compression process although the position of the target in the range direction does have an effect on the azimuth compression process (see 3.3 **The input files**).

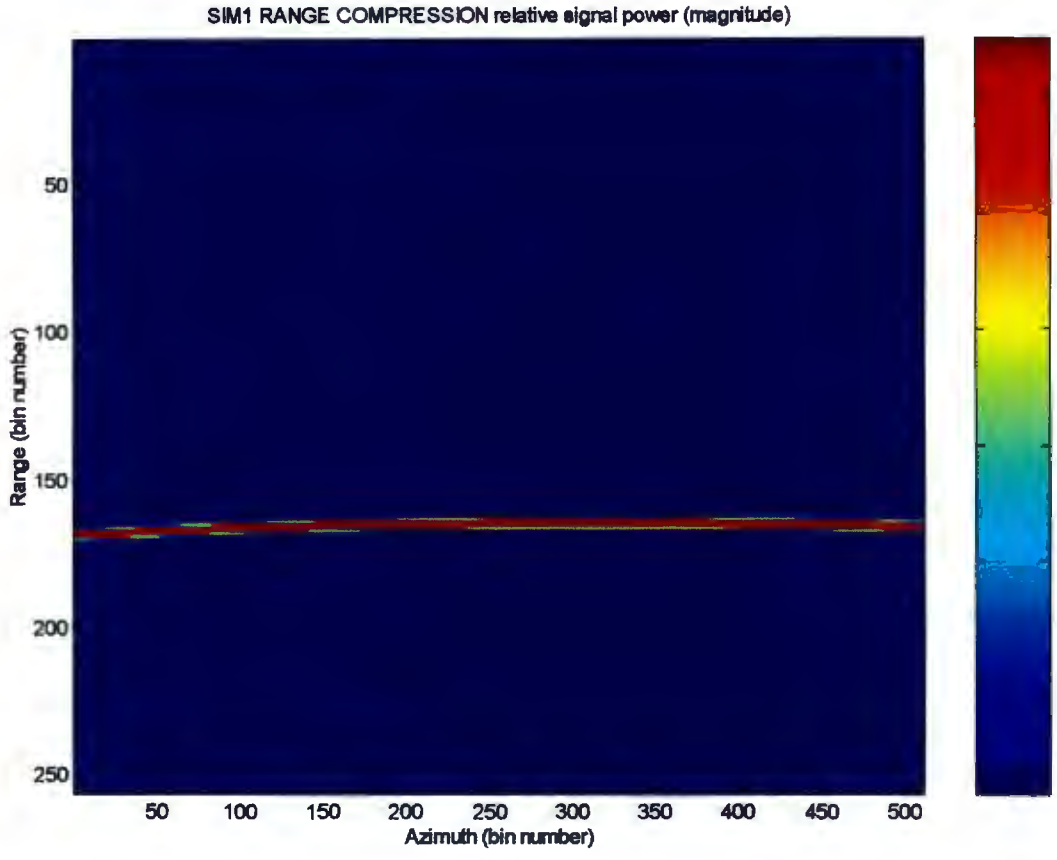


Figure 23: Sim1 Range Compression Arc (Top View - range vs azimuth)

4. AZIMUTH COMPRESSION

4.1 Why Perform Azimuth Compression?

Although azimuth information is sampled at very high resolution, the echo signal from a target is distributed over a large number of samples corresponding to the interval that the beam sweeps past the target. This interval is known as the *coherent integration time*. The problem then is to distinguish the echo signal in terms of its position within the beam.

The centre of the beam passing over the target can be identified by the phase of the echo as follows:

Refer to Figure 24. Consider a pulse transmitted when the target is nearest distance R away. When the pulse is returned, the aircraft has moved a distance x along a straight line perpendicular to the target. The target is now a distance R + dR away.

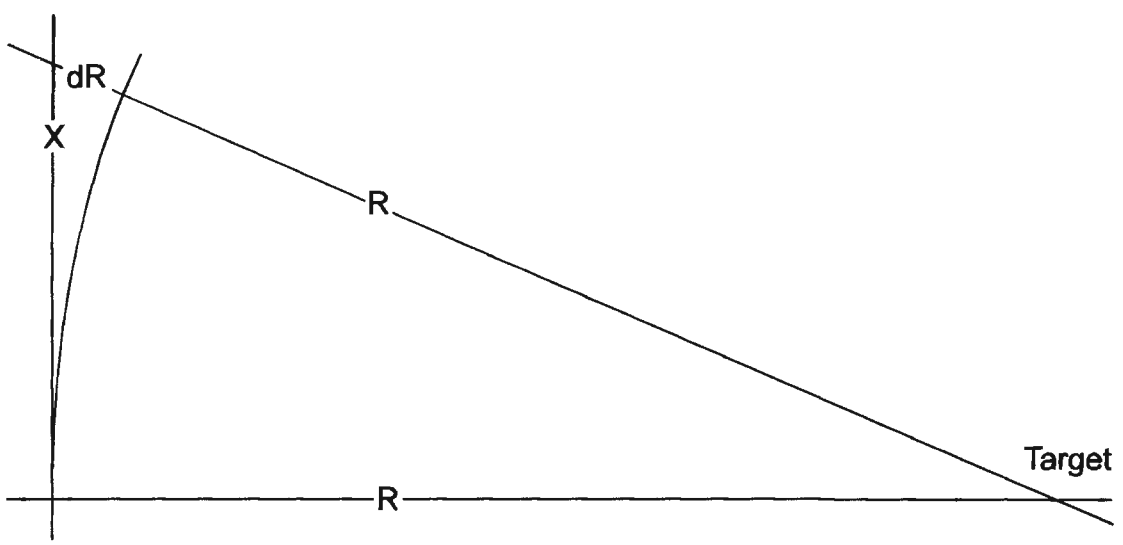


Figure 24: Boresight Line of Array

The phase error is then:

$$\Phi = 2\cdot\pi\cdot f\cdot t_d \qquad \Phi = 2\cdot\pi\cdot f\cdot\frac{2\cdot dR}{c} \qquad \Phi = 2\cdot\pi\cdot\frac{2\cdot dR}{\lambda}$$

where t_d is the two way time delay and f is the transmitted pulse frequency.

For the given nearest range distance R, the phase varies (approximately) quadratically with the azimuth position. This quadratic variation in phase can be interpreted as a frequency that varies linearly with azimuth position as demonstrated by the following:

Given that

$$(R - dR)^2 = R^2 - x^2$$

$$R^2 + 2\cdot R\cdot dR + dR^2 = R^2 + x^2 \qquad 2\cdot R\cdot dR\cdot\left(1 + \frac{dR^2}{2\cdot R}\right) = x^2$$

dR is approximately $dR = \frac{x^2}{2\cdot R}$ giving $\Phi = 2\cdot\pi\cdot\frac{x^2}{\lambda\cdot R}$

as $x^2 = V^2\cdot t^2$, $\Phi = 2\cdot\pi\cdot\frac{V^2\cdot t^2}{\lambda\cdot R}$

as $\omega = \frac{d\Phi}{dt}$, $\frac{d\Phi}{dt} = 2\cdot\pi\cdot\frac{V^2\cdot 2\cdot t}{\lambda\cdot R}$

and finally, as $\omega = 2\cdot\pi\cdot f(\Phi)$, we get the frequency shift as

$$f(\Phi) = \frac{2\cdot V^2\cdot t}{\lambda\cdot R}$$

where V is the velocity of the source and t is the time it took to travel the distance there and back.

The changing frequency of the signal in the azimuth direction resembles a linear chirp multiplied by the antenna pattern as represented by Figure 25.

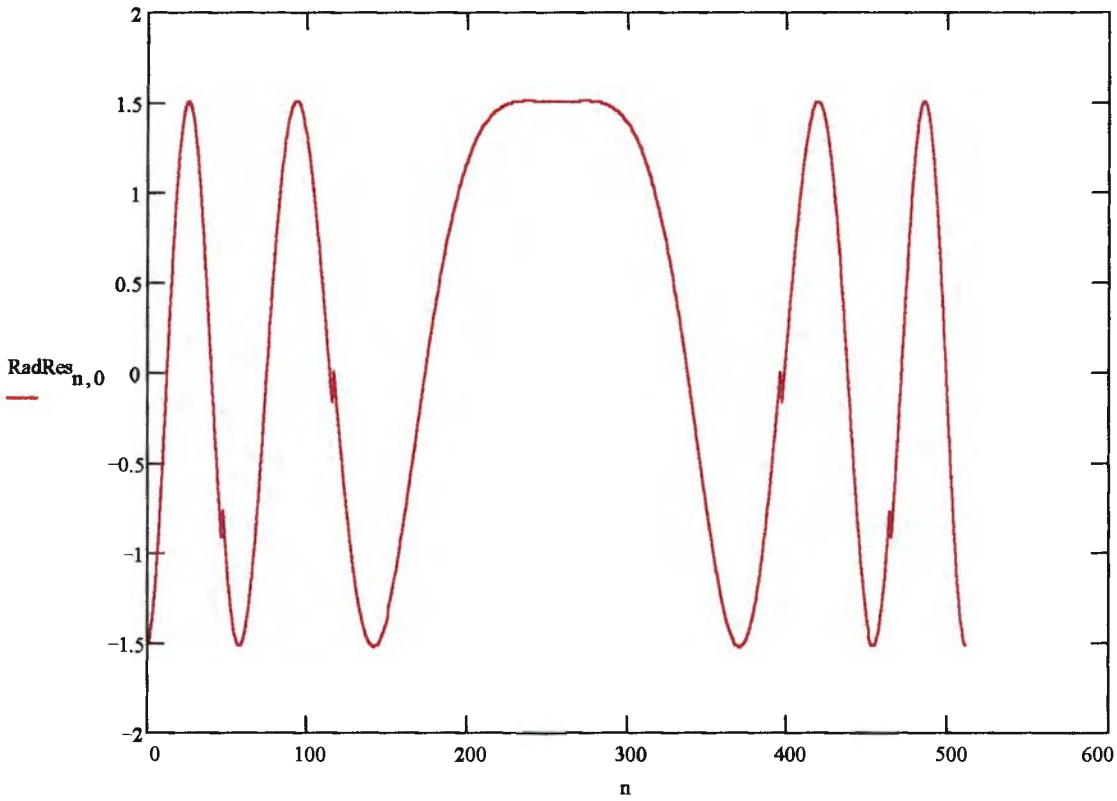


Figure 25: Azimuth Phase Shift in Azimuth Plane

Thus, the azimuth phase shift must be correlated with the point target response in the azimuth direction.

4.2 The Azimuth Compression Software

The Azimuth Compression Software consists of the *cnrturn.c*, *pvmazmin.c*, *azm.c* and *azmcomp.c* files as shown in Figure 26.

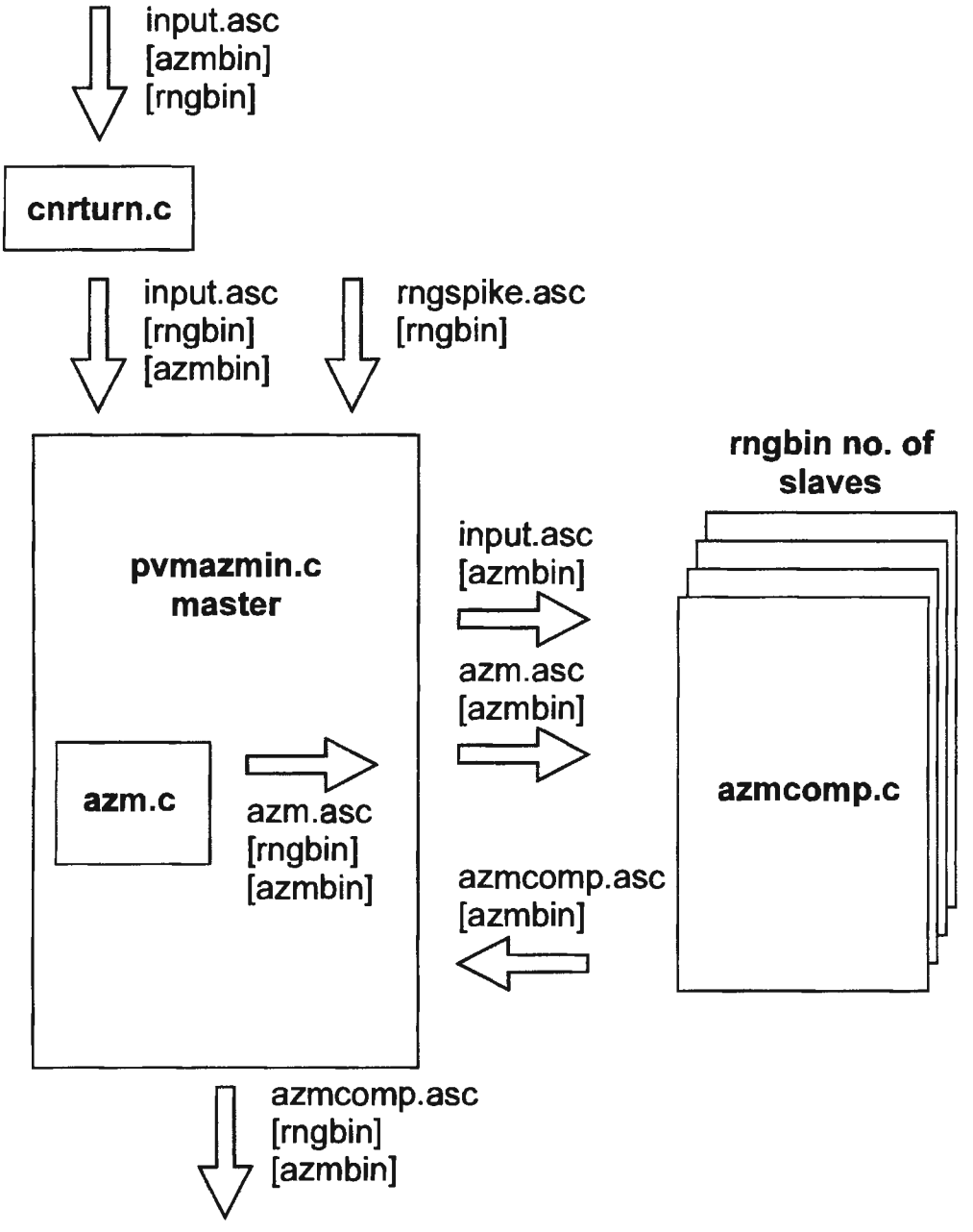


Figure 26: Azimuth Compression Algorithm

4.2.1 The input files

The input files used for testing the thesis software were range compressed versions of *T2.asc* and *sim1.asc* which are outputs from the radar simulators RADSIM **[8]** and SARSIM **[15]** respectively.

The assumptions and parameters used in generating these input files are explained in Chapter 3. **RANGE COMPRESSION** was first run on these files before the following corner turning and azimuth compression took place.

4.2.2 The cnrturn.c file - Matrix Corner Turning

Corner Turning is used to transpose the matrix of the input file from "range returns per azimuth" to "azimuth values per range bin". Files input to the thesis Azimuth Compression software require this latter format. Files of the first format, such as files output from the thesis Range Compression software, are converted by the Corner Turn function.

Corner Turning is traditionally a complicated method which shuffles the positions in memory that store the large amounts of SAR image data. In this case, the data is small enough to be shuffled while reading the input file.

4.2.3 The rngspike.asc file

The range compressed waveform used as a reference in *azm.c* is read from the file *rngspike.asc* which is generated by the thesis Range Compression software.

4.2.4 The pvmazmin.c file

The *pvmazmin.c* file is the pvm master which reads the azimuth compression input file, runs *azm.c* to generate the matching reference

image then generates pvm slaves to perform the azimuth compression. *Pvmazmin.c* reads the file *pvmcomp.ini* for the setup parameters.

4.2.5 The azm.c file

Azm.c calculates this 3-dimensional image of the expected range and azimuth profile.

The following describes the steps taken during calculation starting from the simple range compressed pulse to the resulting 3-D image.

To calculate the simple range compressed pulse, an example range compressed signal is obtained and assumed the same for each azimuth vector, see Figure 27. The range compressed waveform used as a reference is read from the file *rngspike.asc* which is generated by the thesis Range Compression software.

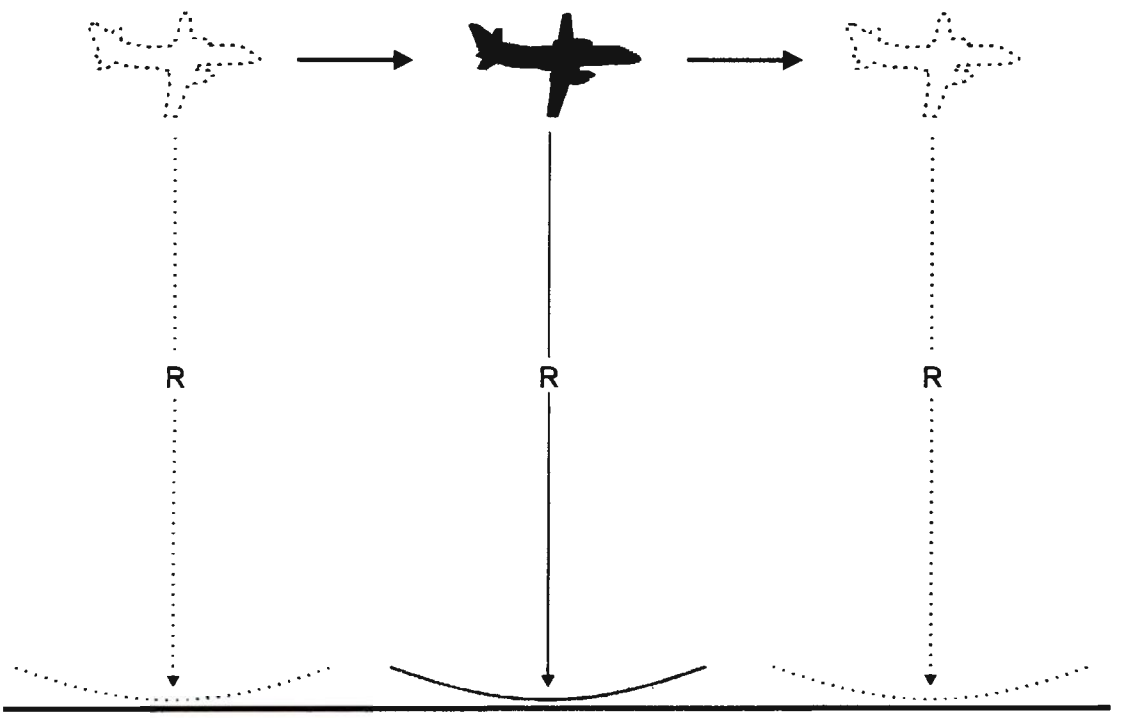


Figure 27: Simple Range Compression Signals

Azm.c then calculates the 3-dimensional range of the target to the source at each azimuth bin, taking the source velocity into account.

Figure 28 shows the range of the source to the target as the source travels past. The figure was generated using a smaller AZMBIN value which created a "zoomed-in" representation. The figure clearly displays the range of the target being closest as the source travels past.

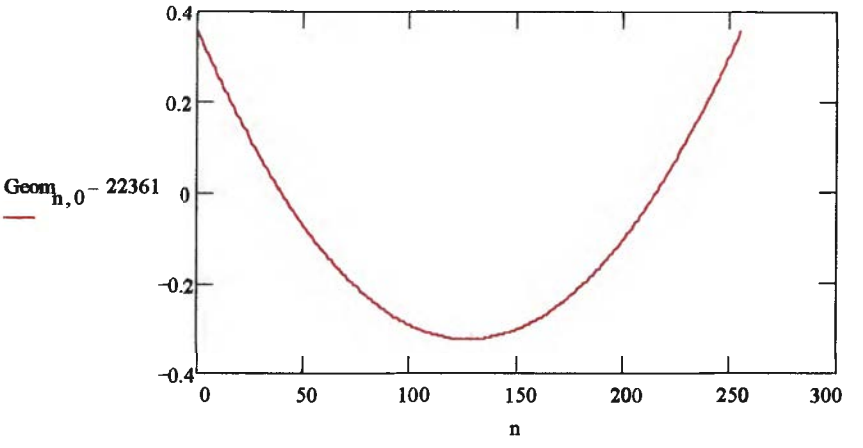


Figure 28: Range of Target to Aircraft vs Azimuth

Azm.c uses the range to calculate the signal gain given the transmitted power, the RCS, the radar losses, the antenna gain and an amplifying factor.

Gain is initially calculated as

$$GainPower := \frac{CLight}{(CentreFreq \cdot 1.0^{9})} \cdot \sqrt{\frac{TxPower \cdot 1000 \cdot RCS}{64 \cdot \pi^{3} \cdot 10^{\frac{Losses}{10}}}}$$

Gain is then recalculated at each azimuth, taking the geometry and antenna gain into account.

$$Gain := \frac{10^{\frac{AntGain}{10}} \cdot Amplify \cdot GainPower}{(Geom_{az})^{2}}$$

Figure 29 shows the return signal gain increasing as the source travels past the target. The figure was generated using a smaller AZMBIN value which created a "zoomed-in" representation. The figure clearly displays the return gain being greatest as the source travels past.

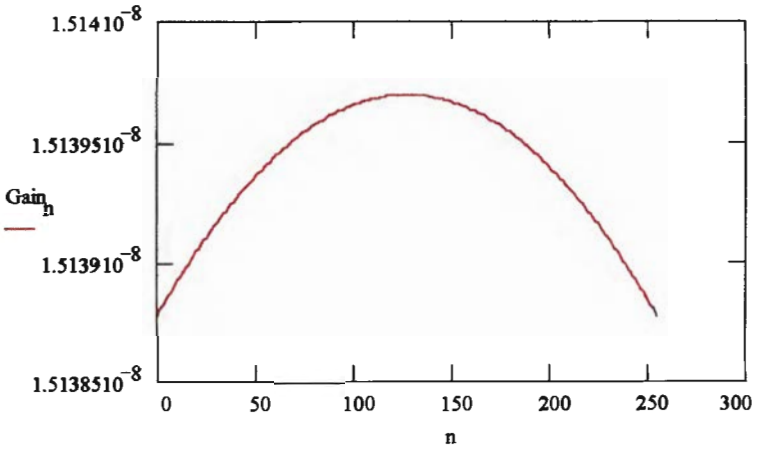


Figure 29: Radar Gain vs Azimuth

The timeshift is then used to calculate the range compressed waveform values at each range, see Figure 30.

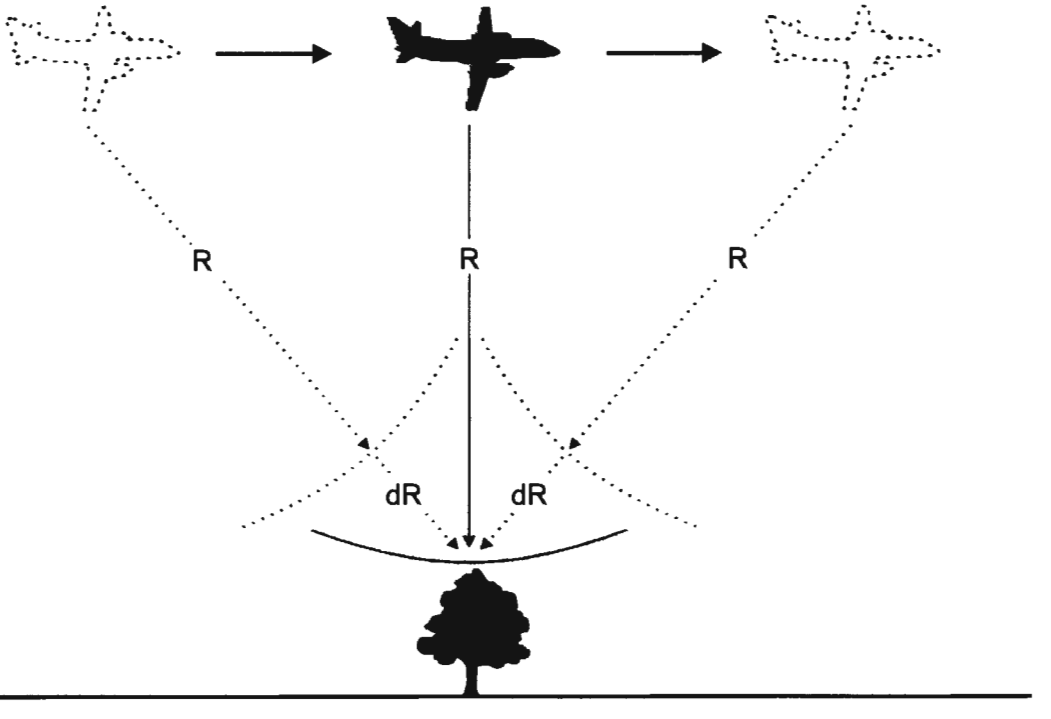


Figure 30: Range Compressed Pulses with Timeshift

At each azimuth, *azm.c* uses the range to calculate the timeshift which is the two way delay

$$t_d := \frac{2 \cdot r}{c}$$

The range value has been shortened by a range offset to bring the range returns within the simulation environment. See section 3.2.1 for the RNGOFFSET values used for the test files. The timeshift equation becomes:

$$t_d := \frac{2 \cdot (r - RngOffset)}{c}$$

The final step is to add the effect of the azimuth phase shift. As explained previously, the azimuth phase shift is seen as a chirp pulse in the radial direction.

Azm.c uses the timeshift to calculate the azimuth phase shift from the equation:

$$\Phi := 2 \cdot \pi \cdot f \cdot t_d$$

where t_d is the two way time shift and f is the transmitted pulse frequency.

The azimuth phase shift vector is multiplied with the "image" vector to give the resulting two dimensional azimuth reference image.

$$ReAz := Re \cdot \cos(\Phi) - Im \cdot \sin(\Phi) \qquad ImAz := Re \cdot \sin(\Phi) - Im \cdot \cos(\Phi)$$

4.2.6 The azmcomp.c file

Each pvm slave is apportioned an range vector of the input file and that of the reference file *azm.asc*. The slaves each run *azmcomp.c* to match the azimuth values within those vectors, see **Figure 8**.

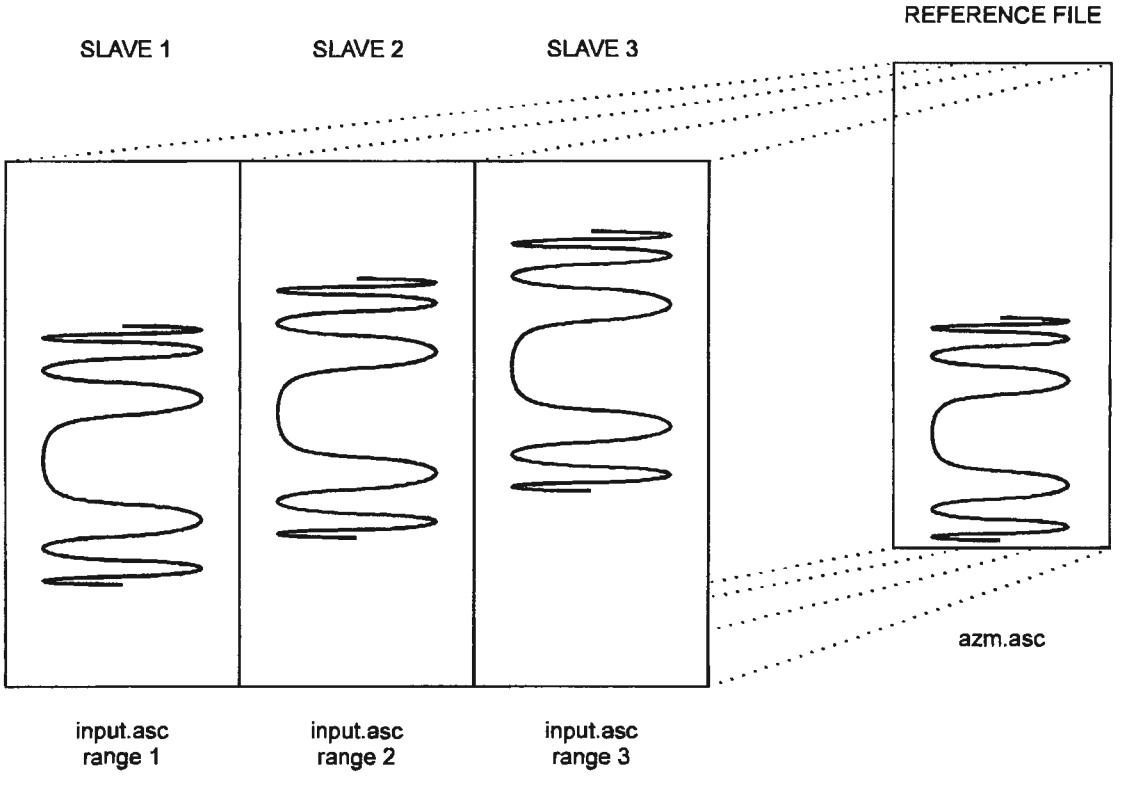


Figure 31: Azimuth Compression Slaves

Azmcomp.c contains the azimuth compression algorithm which scans azimuth values for chirp pulses having frequencies that match those expected. This is achieved by transforming the raw image data to the frequency domain, multiplying the frequency pattern with that of the *reference function (matching filter)*, then converting the result back to time domain[4]. The FFT process is that from **[2]**.

[4] See Appendix E **MATCHING THE CHIRP PULSE** which explains the theory of matching chirp pulses

The matching process matches the image from the start of the image vector to the end by assuming that the image before the start and after the end has the same pattern (infinite extent). To rule out this foldback operation from affecting the result, the reference and image vectors are expanded to double the AZMBIN length and this excess is padded with zeros. The Azimuth Compression process returns the result having double the original length which includes the resulting waveform in the case of *T2.asc* and *Sim1.asc* because the target is positioned at or after AZMBIN number of bins.

Due to range curvature, the edges of the signal in any specific range bin are moved into further range bins. By matching all azimuth values in a single range bin, we are omitting part of the signal. To focus the energy back to the original range bin, the compression program limits the window size to those azimuth values that contain the signal at the centre of the curve.

4.2.7 Compiling the Azimuth Compression Program

Use the make utility as described in the section 2.2.6 **Compiling the Range Compression Program** to create the executable **azmmast**. The make file is set up to create both compression programs and the corner turn executable as well as the slave programs if invoked without any arguments.

To execute the azimuth compression program you must type:

```
azmmast inputfilename outputfilename
```

where **inputfilename** is the corner-turned, range compressed data in ASCII format and **outputfilename** is the processed azimuth compressed data.

4.3 Program Methods and Results

The Azimuth Compression Software is given the radar image as the azimuth values expected at the same range vectors. In the test examples used here the target is a stationary point target and the radar source travels past it in a straight line perpendicular to the target at a constant height.

The following figures[5] show the results of range compressing the input file *T2.asc* except where specified as compressing *sim1.asc*.

The master divides the image into the azimuth values for each range bin and supplies each to the slaves. The master also generates the two dimensional reference azimuth values for the matching of the azimuth chirp.

The reference waveform varies according to the range and so, in turn, does the chirp compression factor.

[5] All figures in this section show the actual values obtained from the test results

At range bin 163, the reference waveform *azm.asc* is shown in Figure 32 as it resembles a chirp waveform and it matches the input file *T2.asc* as shown in Figure 33.

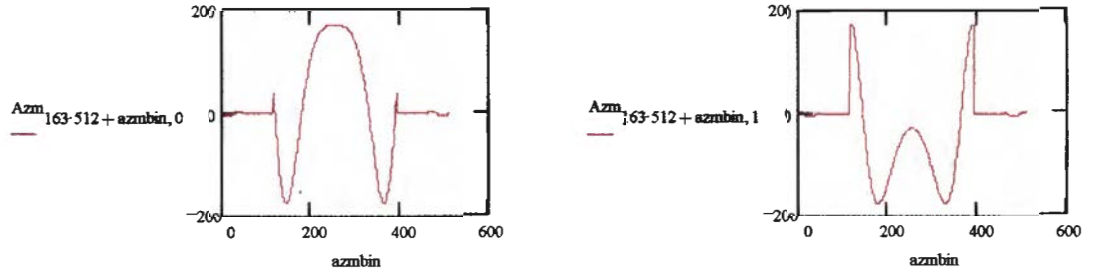


Figure 32: Reference Chirp (real & imag)

The range compressed input of file *T2.asc* has that pattern at range bin 163, see Figure 33.

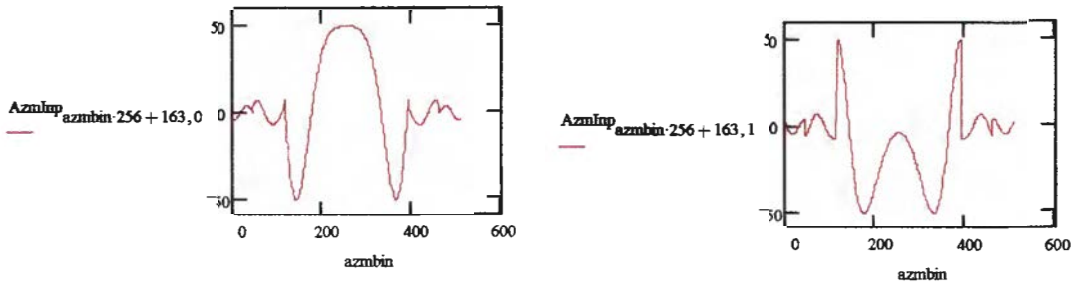


Figure 33: Input Chirp (real & imag)

The master shares out these azimuth waveforms to the slaves to match with the reference waveforms. An example of the corresponding real and imaginary waveforms are shown in Figure 34 which are matched by the slaves. The centres of these waveforms in range bin 163 are both at 256 for the reference and input waveforms of *T2.asc* respectively.

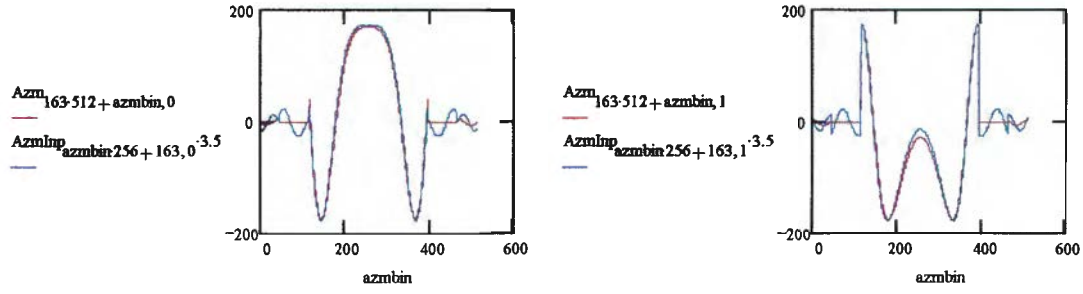


Figure 34: Reference and Input Chirps (real & imag)

The reference pulse is "windowed" using a Hamming window to reduce range sidelobes (at the expense of gain and resolution)[6]. The windowed reference function is shown in Figure 35.

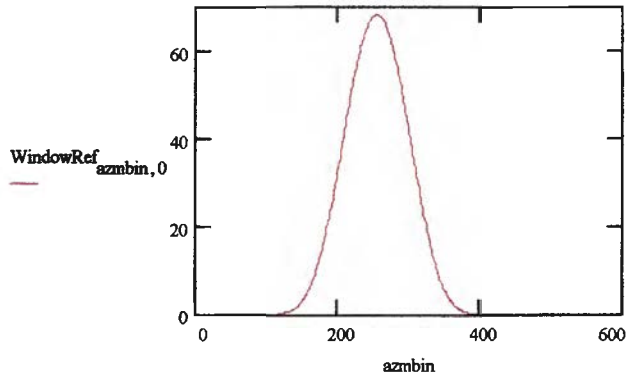


Figure 35: Windowed Reference Waveform (power)

The slaves match the waveforms by converting the reference waveform and input waveform into the frequency domain. This is done by performing an FFT (Fast Fourier Transform) on the waveforms. The frequency description of the reference waveform is shown in Figure 36.

[6] See Appendix E **MATCHING THE CHIRP PULSE** which explains the theory of matching chirp pulses

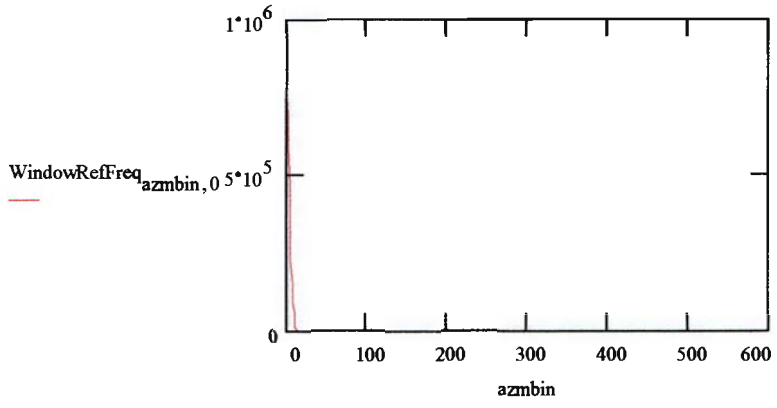


Figure 36: FFT of Reference Waveform (power)

The frequency description of the input waveform is shown in Figure 37.

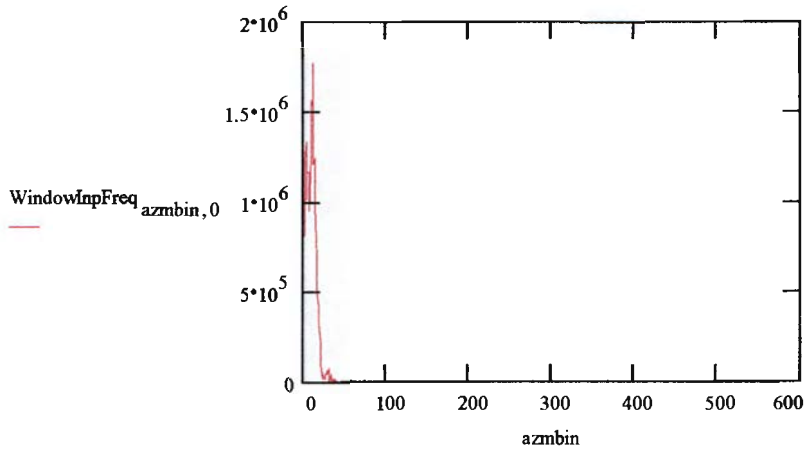


Figure 37: FFT of Input Waveform (power)

The frequency waveforms of the reference waveform and the input waveform are multiplied together to effectively correlate the two in terms of the time domain. The resulting waveform is shown in Figure 38.

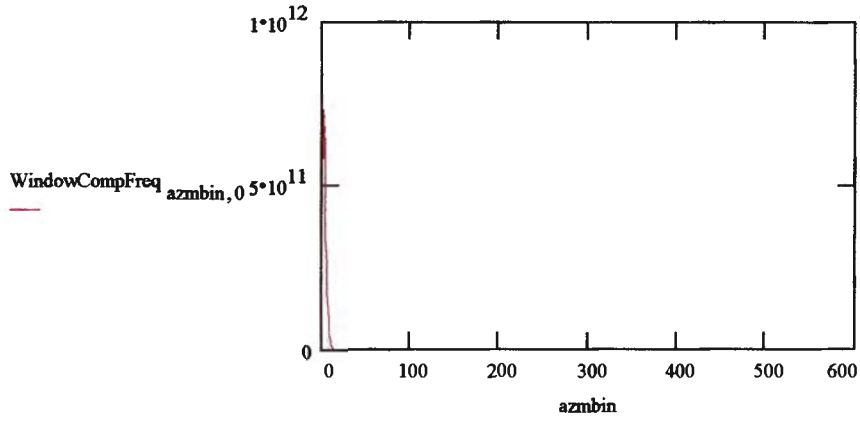


Figure 38: Multiplied FFT Waveforms (power)

The resulting waveform is then transformed back to the time domain revealing the resulting compressed pulse, see Figure 39.

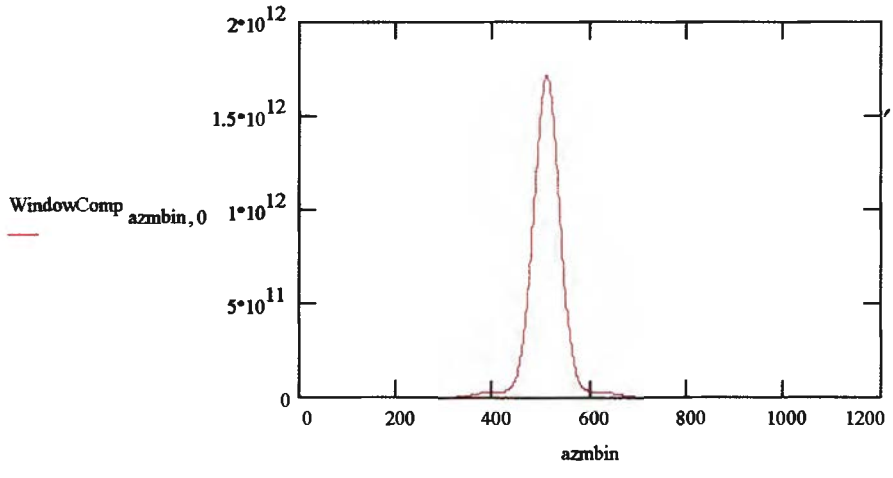


Figure 39: Azimuth Compressed Pulse (power)

We expect the resulting waveforms to have the following characteristics:

1. The main lobe width should be approximately 64 units wide calculated from $W := \frac{2 \cdot v \cdot \sin\left(\frac{\theta}{2}\right)}{\lambda}$ where v is the source velocity of 246 m/s, λ is the wavelength and θ is the angle of approximately 32° as the source's view looking down in a triangle whose base line is made up of the 512 azimuth points.

2. The origin should be at approximately 512 units which is the difference of the two pulses (0 units - 0 units) plus the pulsewidth (512 units).

Figure 40 zooms in on the compressed pulse to show that our expectations are met satisfactorily.

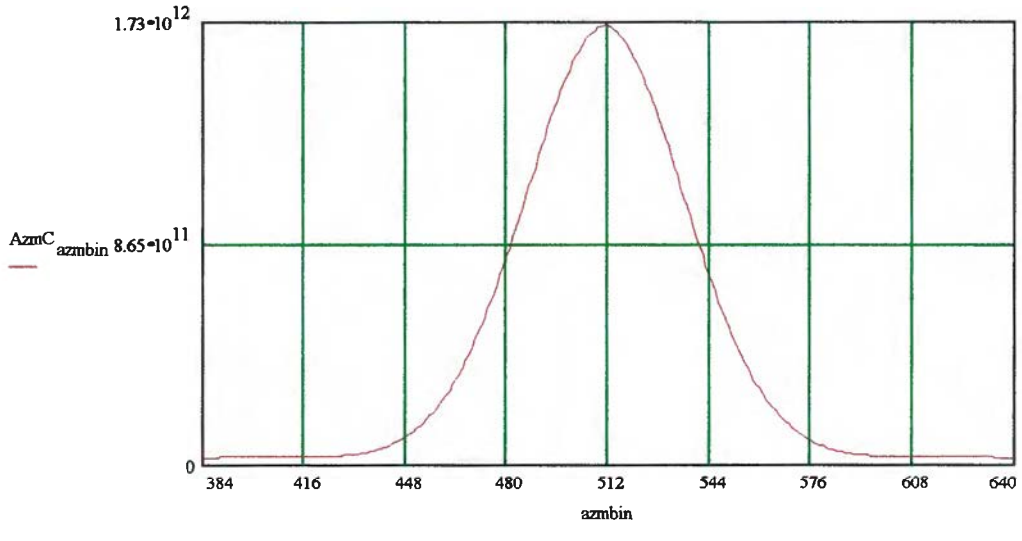


Figure 40: Azimuth Compressed Pulse (zoomed-in)

The *pulse compression ratio* is expected to be approximately 4 from the equation: $\frac{T}{t_0} := \frac{140}{32}$

Figure 41 shows the envelopes of the input and resulting waveforms by combining the complex vectors of each signal. The figure shows this expected ratio to be true.

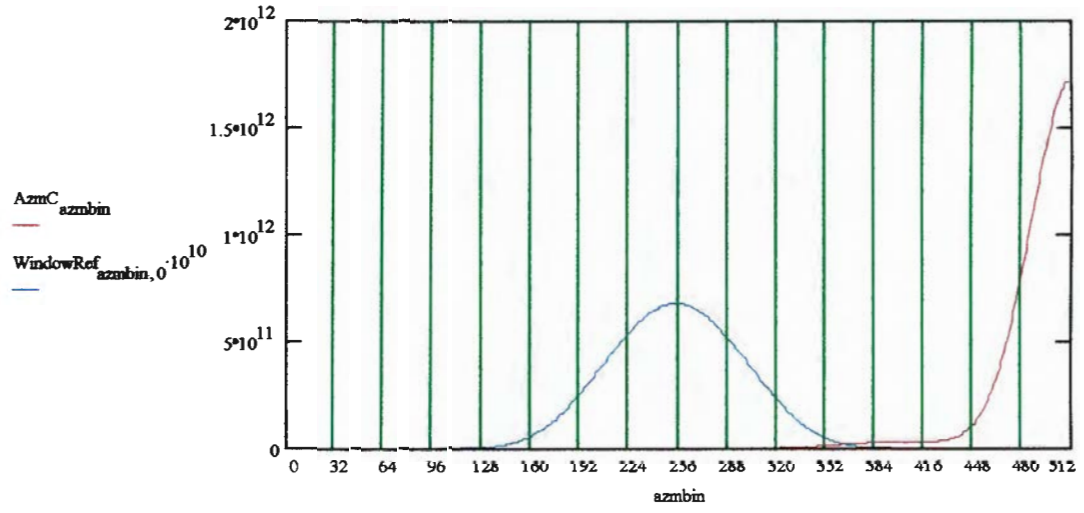


Figure 41: Envelopes of Azimuth Compressed Chirp Pulse vs Windowed Reference Pulse (power)

The azimuth compression removes the azimuth phase shift. Each slave matches a different range bin and, once the master combines the compression results, the result, as in Figure 42 and Figure 43, reveals a three dimensional spot representing the target position.

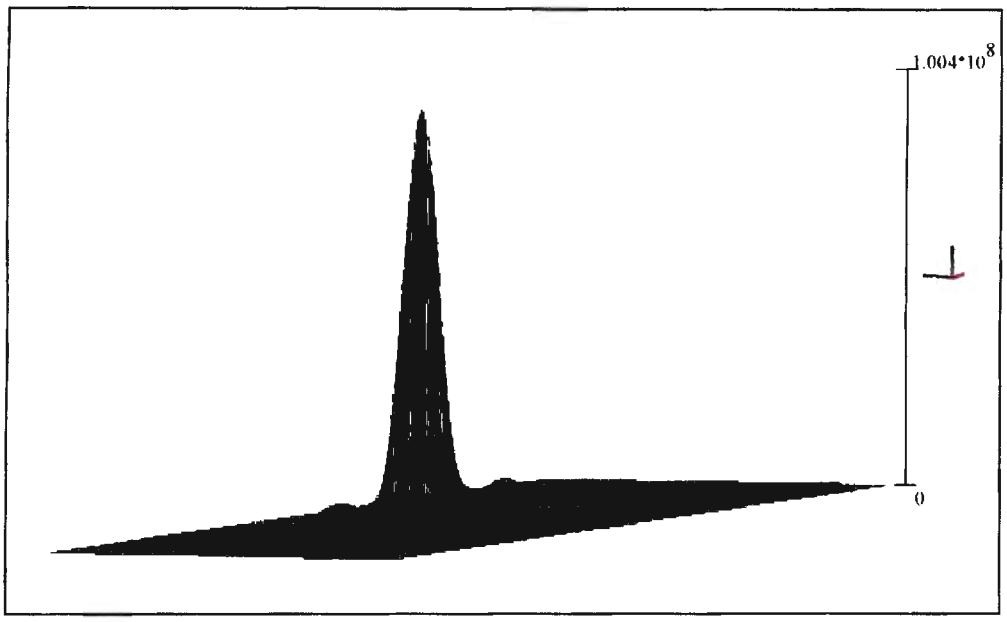


Figure 42: T2 Azimuth Compression Spot (3D - range vs azimuth)

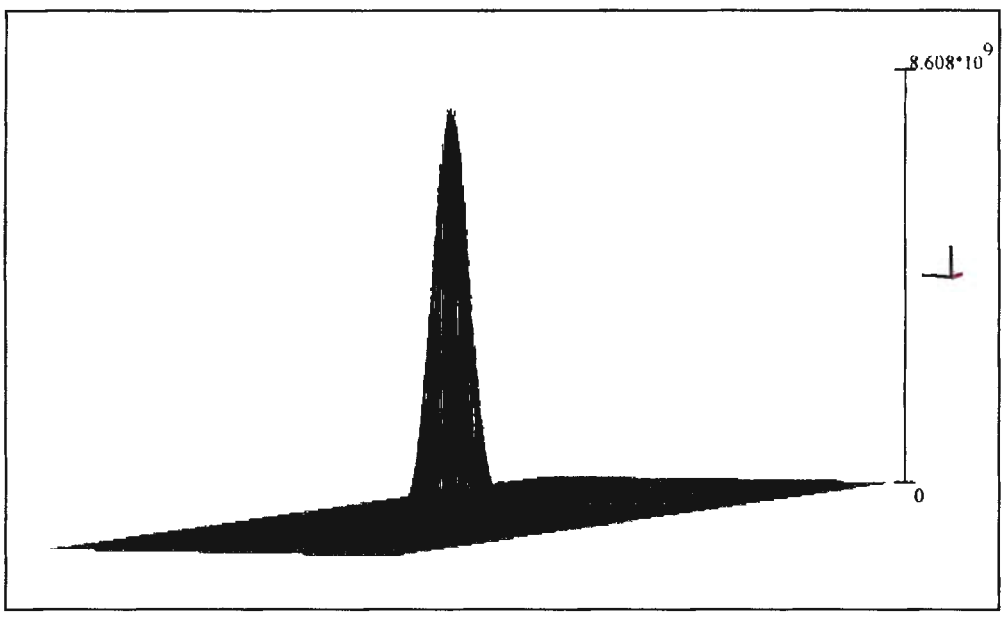


Figure 43: Sim1 Azimuth Compression Spot (3D - range vs azimuth)

Seen from the top in Figure 44, the spot is quite visible.

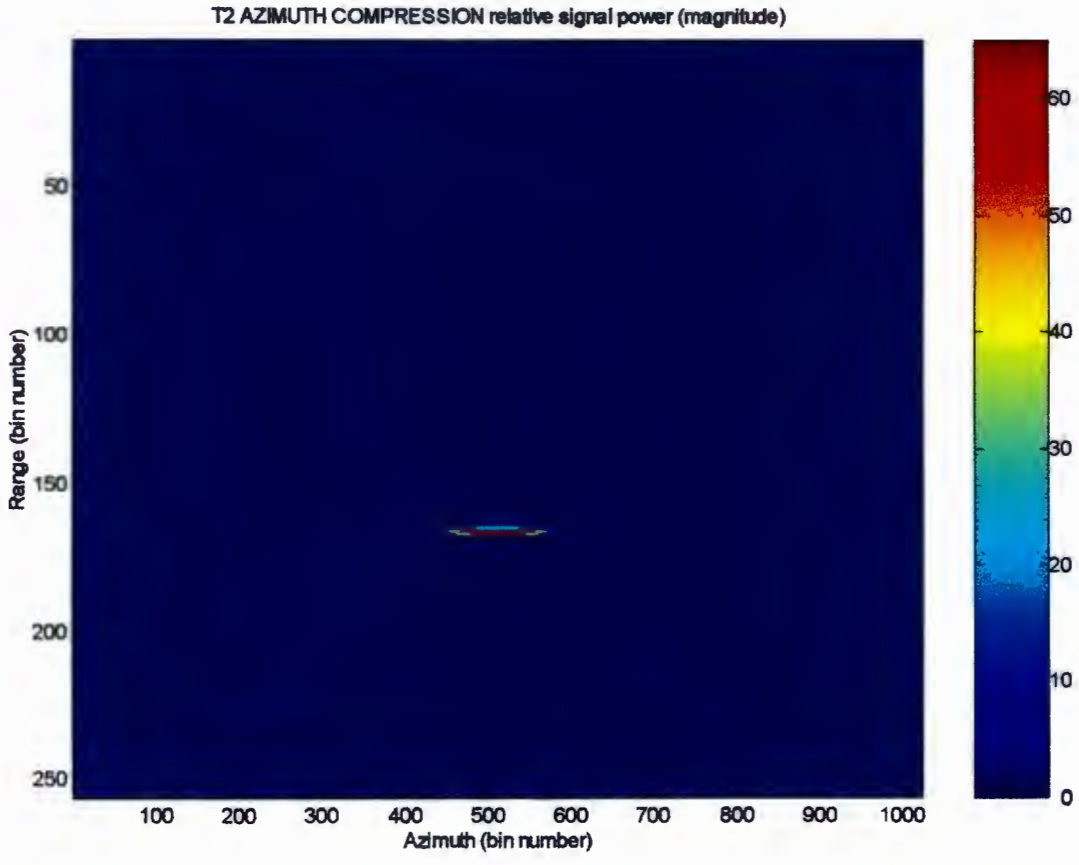


Figure 44: T2 Azimuth Compression Spot (Top View - range vs azimuth)

The azimuth compression results are similar for *T2.asc* (see Figure 44) and *sim1.asc* (see Figure 45).

The position of the spot differs due to the capturing process of the two files. The two files were captured with different geometries to simulate different positions of the point target. The position of the *T2.asc* target was in the centre of the azimuth geometry which is where it is shown. The position of the *sim1.asc* target was slightly to the right of centre and is also proven to be correctly placed by the compression program.

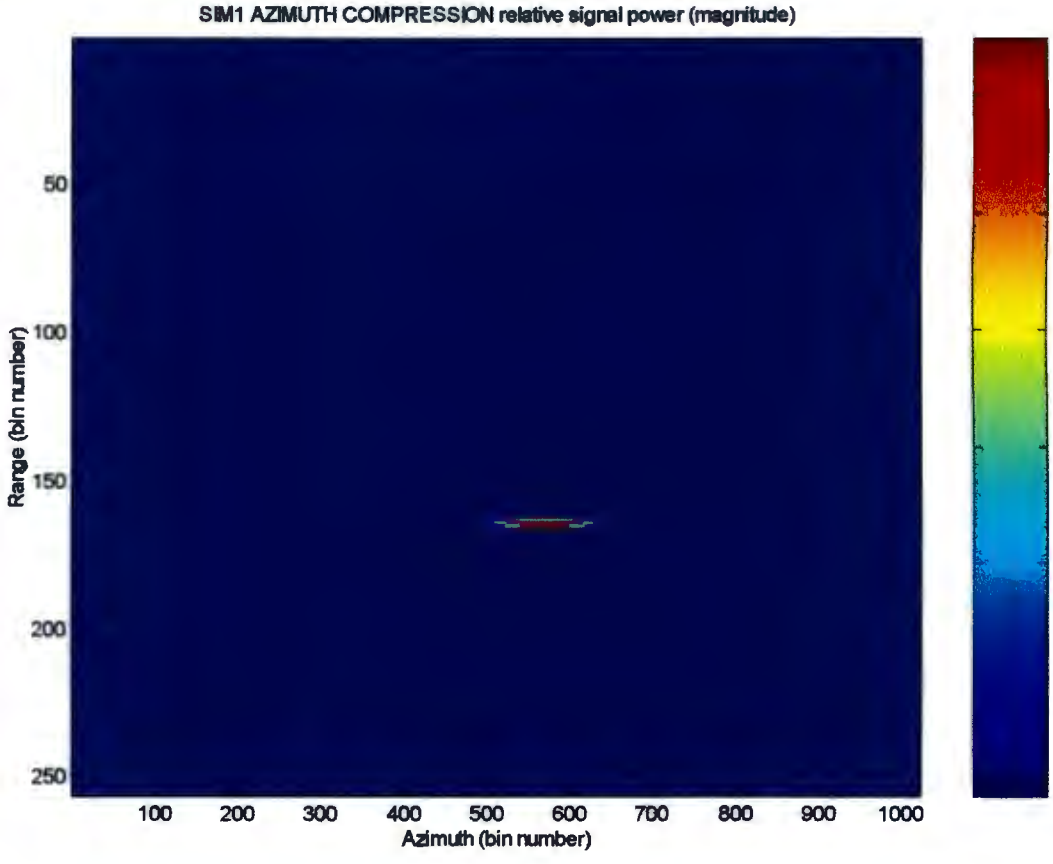


Figure 45: Sim1 Azimuth Compression Spot (Top View - range vs azimuth)

The position of the target in the azimuth direction has no effect on the range compression process (see Chapter 3: **RANGE COMPRESSION**) although the position of the target in the range direction does have an effect on the azimuth compression process. This is solved by the value of RNGOFFSET specified before compression by the user in the *pvmcomp.ini* setup file.

5. TESTING PVM

5.1 The PVM Environment

5.1.1 The Setup

PVM version 3.3.1 was installed on four heterogeneous machines on a non-switched, moderately loaded Ethernet network and then the program was tested with different machine configurations.

The four machines are shown in Table 4.

Table 4: Architectures tested by PVMCOMP

Machine	Operating System	Network IO Interface	Memory (MB)
Tatung Pentium 75MHz	Linux v1.3.20	UTP	32
Alphastation 200	Digital Unix v3.2c	UTP	32
HP 9000 / 806	HP-UX v10.20	UTP	64
HP 9000 / 817	HP-UX v9.0	UTP	96

5.1.2 Installations

Installations of PVM onto the machines worked correctly on the first attempt. The only post-installation work involved setting the permissions on the $PVM_ROOT/bin directory to allow writing of slave executables by all users and setting the environment variables such as PVMROOT as described in the installation document.

Pentium (Linux 1.3.20)

1. Some problems were experienced with the remote login access onto the Linux version. After repeated attempts to set up the .rhosts and inetd.conf files, it was decided to use the Linux box as the PVM master.

2. During compilation of the C code the GNU C compiler did not generate warnings regarding array addressing that were generated by the other architectures. This did not influence the operation of the code and can be attributed to the default warning level of the compiler.

3. A local PVM installation was done into a user directory on this machine.

Alphastation (Digital UNIX 3.2c)

1. The Digital UNIX Alphastation machine did not have GNU C installed and the default DEC C compiler was used.

2. The PVM installation was done into a public directory on this machine necessitating changes to the Makefile to use /usr/local/pvm3/bin/ALPHA.

3. The math.h header file needed PI defined and so it was added in globals.h.

HP 9000/817 (HP-UX v9.0)

1. The HP UNIX machine did not have GNU C installed and the default HP Non-ANSI C compiler was used.

2. The PVM installation was done into a public directory on this machine necessitating changes to the Makefile to use /usr/local/pvm3/bin/HPPA.

3. The Non-ANSI C compiler forced function declarations to be in the old style with function parameters declared on a separate line after the function declaration. It also did not allow use of the void keyword in function declarations. This only affected the slave routines as no PVM master code was run on this machine.

4. The Non-ANSI C compiler did not allow function prototypes and these were removed.

5. The math.h header file needed PI defined and so it was added in globals.h.

6. The Non-ANSI C compiler also does not allow initialization of string arrays. These were replaced with the relevant strcpy() functions.

HP 9000/806 (HP-UX 10.20)

1. The HP UNIX machine did not have GNU C installed and the default HP Non-ANSI C compiler was used once again.

2. The PVM installation was done into a public directory on this machine necessitating changes to the Makefile to use /usr/local/pvm3/bin/HPPA.

3. The math.h header file needed PI defined and so it was added in globals.h.

5.1.3 Running PVM

The master code was written to specify the specific host machine when spawning slaves. The actual slave/host configuration for each run was read from a configuration file **config.ini**. The PVM spawn command to start slaves on the Alpha machine was:

```
numt=pvm_spawn(SLAVENAME, (char**)0, PvmTaskArch,
    "Alpha", NumSlavesAvailable, tids);
```

The SLAVENAME argument above is the zero-terminated character string describing the slave executable compiled on all slave machines. The program must be compiled separately for every architecture in the virtual machine typically using the following command:

```
cc -L~/pvm3/lib/ARCH file.c -lpvm3 -o file
```

where **file** is the file to be compiled and **ARCH** is the architecture name of the computer.

After compiling, the executable was placed in the directory ~/pvm3/bin/**ARCH** which is where PVM looks by default.

The tests were performed in host-dependent mode where the user specifies exactly which host computer in the virtual machine the slave is to run on.

Calling "pvm hostfile" on the command line starts PVM running on all specified machines automatically. This can be verified by entering the PVM console and typing the **conf** command.

Here is the host file called linux_dual used below in the example:

```
pentium

alpha dx=/usr/local/pvm3/lib/pvmd so=sm

hppa1 dx=/usr/local/pvm3/lib/pvmd so=sm
```

Here is a typical output from the PVM console after starting PVM with the above hostfile:

```
pentium:/home/swuyts/pvmwork# pvm linux_dual

pvm> conf

3 hosts, 3 data formats

                     HOST     DTID     ARCH   SPEED

                  pentium    40000    LINUX    1000

                    alpha    80000    ALPHA    1000

                    hppa1    c0000     HPPA    1000

pvm>
```

Calling pvm from the command line without specifying a hostfile runs PVM only on the local host.

5.1.4 The Program and Tests

The Range Compression and Azimuth Compression programs are run sequentially with Azimuth Compression input data made up of the results from the Range Compression program.

A compression program consists of a PVM master which sets up the data, spawns PVM slave tasks, handles message buffers to and from these tasks, and manages the data to and from the slaves.

Using the test input files, the Range Compression program needs 512 slaves in total and the Azimuth Compression program needs 256 slaves in total. The master reads the *config.ini* file to see how many slaves to run at a time per remote machine. It then performs as many runs of this amount necessary to achieve the total number of slaves.

The ASCII input file used in all cases was the *Sim1.asc* file. Either alternative input file would take the same processing time as they are all the same length.

The tests shown in Table 5 were performed on the hosts shown in Table 4 by running PVMCOMP on the PVM virtual machine.

Table 5: Setups of Tests Performed

No	Description
1	Each machine, in turn, runs a number of slaves.
2	Combinations of machines run a number of slaves.
3	Slaves recalculate compression N times.
4	Master resends message N times.
5	Master sends large message.
6	Time entire compression procedure.

Tests 1 to 5 measure the time in seconds from when the master enrolls into PVM until it receives the replies from the slaves. It does not measure the time taken to generate the reference data, read setup files and save the output file. This is measured in Test 6. These functions are omitted from the timing as they overwhelm the contrast between the time taken by processing versus the PVM handling time.

5.1.4.1 Test 1

Test 1 consists of a number of smaller tests, each testing the spawning of slaves on specific machines. They measure the processing time for different numbers of slaves per run.

5.1.4.2 Results of Test 1

See Appendix D for the full results of the following tests.

Table 6 shows the running time of the range compression and azimuth compression programs for each specified number of slaves per run on each machine operating in isolation.

Table 6: Running Time of Test 1 (rngcomp / azmcomp) in seconds

Test 1	**No of slaves per run**				
Slave Machine	**16**	**32**	**64**	**128**	**256**
Pentium	49/35	50/34	53/36	58/42	-
Alphastation	74/44	83/52	-	-	-
HP 9000 / 806	79/59	-	-	-	-
HP 9000 / 817	-	-	-	-	-

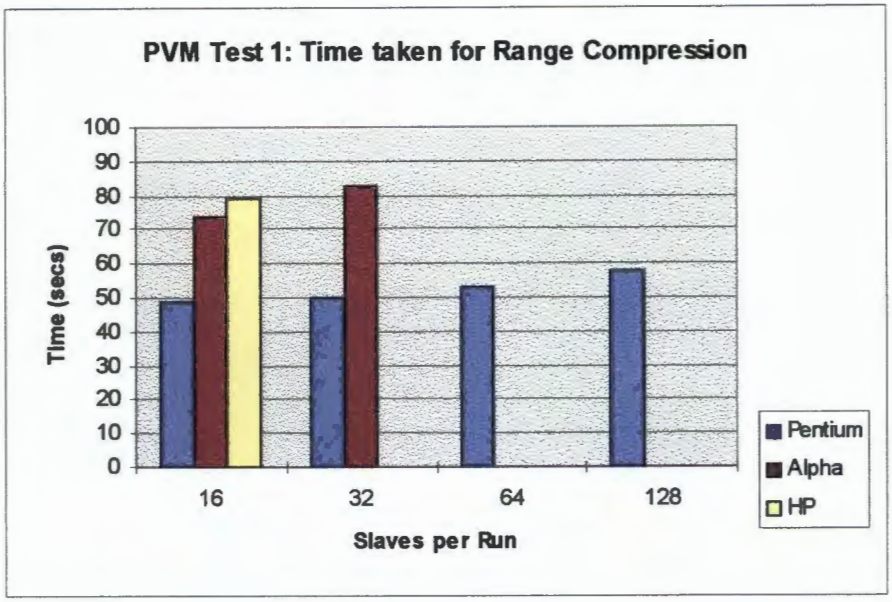


Figure 46: Time taken for Range Compression for the number of slaves per run per machine

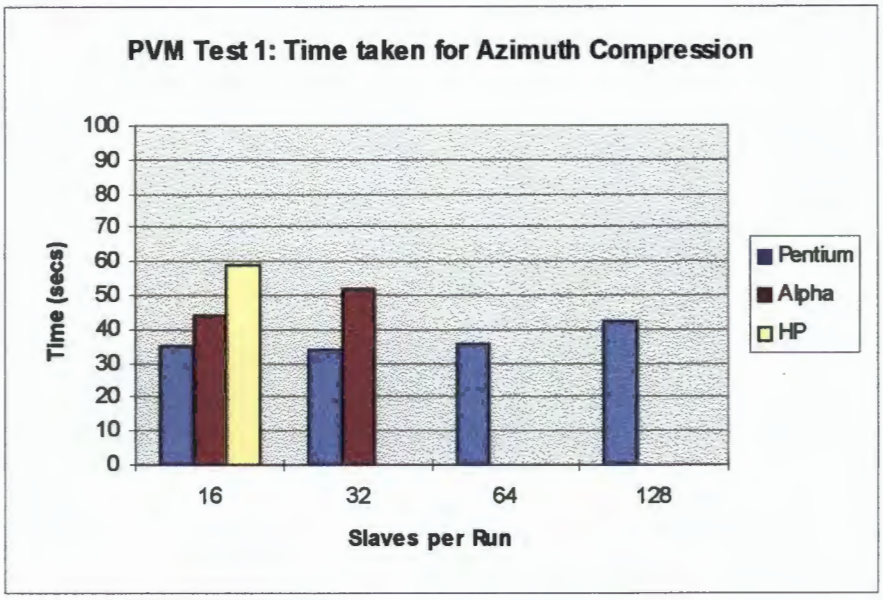


Figure 47: Time taken for Azimuth Compression for the number of slaves per run per machine

1. The maximum number of slaves we could run is 145, see [7]. When attempting to run 256 slaves on the Linux Pentium the master spawned the first 145 slaves then, for the rest, gave the error message "trouble spawning slaves, aborting" and gave the relevant tid numbers to be killed. The other hosts generate this message for even fewer slaves. The hosts used were running other normal network applications and thus were limited to the number of sockets, process slots and memory available for spawning slaves. This was the case for the Alphastation limiting it to 64 slaves and the two HP's to 16 slaves.

2. The problem of a short supply of resources was not always apparent. When the ALPHA box had a short supply of sockets at 64, it logged error messages to the log file but still returned the data messages to the master from the slaves that had managed to get a socket. The data output files were corrupt but the information looked presentable at first glance although the data vectors were in the wrong order.

3. The slaves were monitored on the remote machines by running "top -s1" from the UNIX command line. Linux requires the command "top -d1".

4. An early version of code compiled on the HP 9000 / 806 was trying to save data to a non-existent directory. The master program processed up to 17 out of 32 runs of 16 slaves each before attempting to save the buffered data and reporting the error. This is as a result of the write-back cache on the Alpha machine. No attempt was made to flush the I/O stream to disk as this would increase the overheads of writing the data. The directory name was corrected.

5. Using the slower HP 9000 / 817 was abandoned on account of it being too slow. After 5 minutes it had not yet completed one batch of 16 slaves out of the 32 runs needed for Range Compression. Any

attempt to use this machine would have slowed the entire virtual machine to a crawl. This is an important lesson.

5.1.4.3 Conclusions of Test 1

1. The azimuth processing is faster than range processing because, although the azimuth slaves have twice the data to process, they have half the runs to perform compared to range processing.
2. The times vary slightly for the machines where the range compression performs better and the azimuth compression doesn't and vice versa. This is due to the network fluctuations from other applications. Conclusions are drawn from the general trend.
3. Both azimuth and range compression execute slower as more slaves are used. The graph in Figure 47 shows that the trend is exactly the same for both even though the actual computation and message sizes are completely different.
4. As soon as we send data and start slaves on a remote machine we see a slowdown in the processing for that run. This is due to network related overheads and remote PVM overheads. We see this difference in the times between the local-only slaves on the Pentium and the remote slave startup on the Alpha.
5. The difference between remote slaves on the Alpha and HP is small. This is because both perform the same sequence of operations involving remote slave startup and network I/O.
6. The azimuth compression should take half the time of the range compression. This is due to the number of runs to be executed. The Alpha machine shows this relationship well because it finishes the floating point calculations quickly but the HP machine does comparatively poorly on the azimuth calculation due to its slower floating point performance.

5.1.4.4 Test 2

Test 2 measures the time taken by the program when shared over a combination of machines running a number of slaves. This is a true virtual machine.

5.1.4.5 Results of Test 2

Table 7 shows the running time of the range compression and azimuth compression programs when shared over the three machines at a specified number of slaves per run. The number of slaves per machine were selected optimally based on the results of test 1.

Table 7: Running Time of Test 2 (rngcomp) in seconds

Test 2	**Slave Machines**			**Processing Time**
No of Slaves per Run	**Pentium**	**Alphastation**	**HP 9000 / 806**	**Time (s) of RngComp**
64	32	16	16	49
64	16	32	16	50
64	44	16	4	51

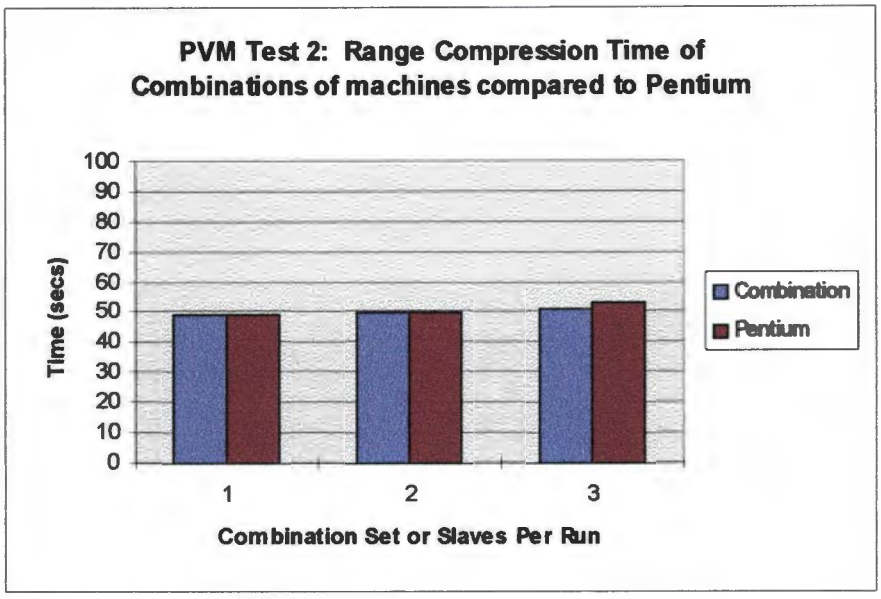


Figure 48: Time taken for Range Compression by set combinations of machines vs slaves per run of Pentium

5.1.4.6 Conclusions of Test 2

1. The time taken over three machines is longer than the time for the run on a local machine. This is important as it shows the current task granulation is incorrect for this configuration of the virtual machine. We would need to increase the work per slave and add more machines to the virtual machine. The time gained by using only two additional machines is neutralised by the extra time needed for the PVM and network processing.
2. Test 2 shows that, if correctly load balanced, optimum results can be achieved regardless of loads on certain machines in the virtual machine. This is essentially using any spare processing on available machines.

5.1.4.7 Test 3

Test 3 tests the program time when the slaves repeatedly (N times) recalculate the compression procedure, for different values of N. This simulates a more complex computation task for each slave.

5.1.4.8 Results of Test 3

Table 8 shows the running time of the range compression program when shared over the three machines at a specified number of slaves per run and for a specified number of iterations of processing.

Table 8: Running Time of Test 3 (rngcomp) in seconds

Test 3	**Iterations**	**Slave Machines**			**Processing Time**
No of Slaves per Run	**Value of N**	**Pentium**	**Alphastation**	**HP 9000 / 806**	**Time (s) of RngComp**
16	5	16			93
16	10	16			149
64	5	32	16	16	90
64	10	32	16	16	143
64	13	32	16	16	-

See Figure 49 and Figure 50 for the results of Tests 3 to 5 for Range and Azimuth Compression respectively.

A maximum of only 12 iterations could be performed due to limitations on the resource allocation. The range compression process opens, writes, and closes 8 files within the function that was iterated and thus quickly reaches the maximum allowable file handles for the process.

5.1.4.9 Conclusions of Test 3

1. Each slave should be as frugal as possible with system resources as this can block certain machines in the virtual machine. Such blocking can effectively stop all other processing.

5.1.4.10 Test 4

Test 4 tests the program time when the master repeatedly (N times) resends the compression data to the slaves, for different values of N. This simulates large data volumes in small packets.

5.1.4.11 Results of Test 4

Table 9 shows the running time of the range compression program when shared over the three machines at a specified number of slaves per run and for a specified number of transmissions.

Table 9: Running Time of Test 4 (rngcomp) in seconds

Test 4	Iterations	Slave Machines			Processing Time
No of Slaves per Run	Value of N	Pentium	Alphastation	HP 9000 / 806	Time (s) of RngComp
16	5	16			71
16	10	16			94
64	5	32	16	16	68
64	10	32	16	16	94

See Figure 49 and Figure 50 for the results of Tests 3 to 5 for Range and Azimuth Compression respectively.

5.1.4.12 Conclusions of Test 4

1. Similar times for the larger data volumes here show that they can quickly overload the network. This means that a certain maximum data through-put should be taken into consideration in designing PVM applications. This will be directly related to network type and it must be noted that the lowest bandwidth section of the network involved will limit the rest of the network. Attempts should be made to interleave processing and I/O on each machine to obtain the maximum performance. This could be a difficult task unless it can be automated.

5.1.4.13 Test 5

Test 5 tests the program time when the master sends the compression data which is N time larger than the original, for different values of N. This simulates fewer but larger packets of data.

5.1.4.14 Results of Test 5

Table 10 shows the running time of the range compression program when shared over the three machines at a specified number of slaves per run and for a specified number of expanding the message size.

Table 10: Running Time of Test 5 (rngcomp) in seconds

Test 5	**Iterations**	**Slave Machines**			**Processing Time**
No of Slaves per Run	**Value of N**	**Pentium**	**Alphastation**	**HP 9000 / 806**	**Time (s) of RngComp**
16	5	16			67
16	10	16			-
64	5	32	16	16	63

See Figure 49 and Figure 50 for the results of Tests 3 to 5 for Range and Azimuth Compression respectively.

1. PVM could not send messages that were 10 times the original message size which each contain 10 integers and 512x4x10 floats in the case of range compression and 10 integers and 256x4x10 floats in the case of azimuth compression.

5.1.4.15 Conclusions of Test 5

1. The nature of the Ethernet network is that it handles heavy loads near its theoretical limit very badly. Here we see that increasing the packet size by 5 times allows better performance by a pool of three machines than the performance of one machine alone. This is ideal and is the whole thrust of PVM. However, increasing the size by 10 times freezes the network I/O system as a result of the network being unable to send the massive packet. Care should be taken in constructing messages not to reach or exceed this limit.

5.1.4.16 Test 3 to Test 5 Results

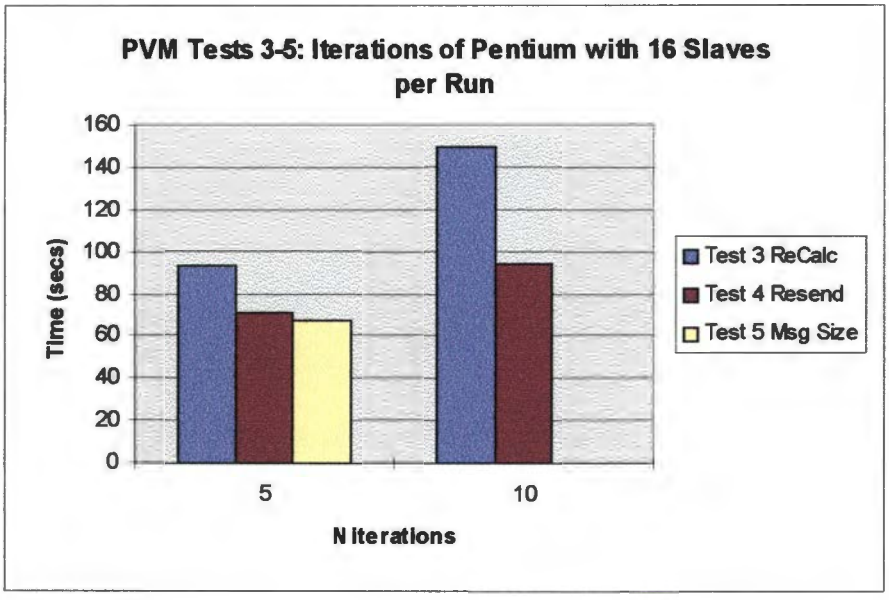


Figure 49: Iterations of calculation, sending and message size on Pentium

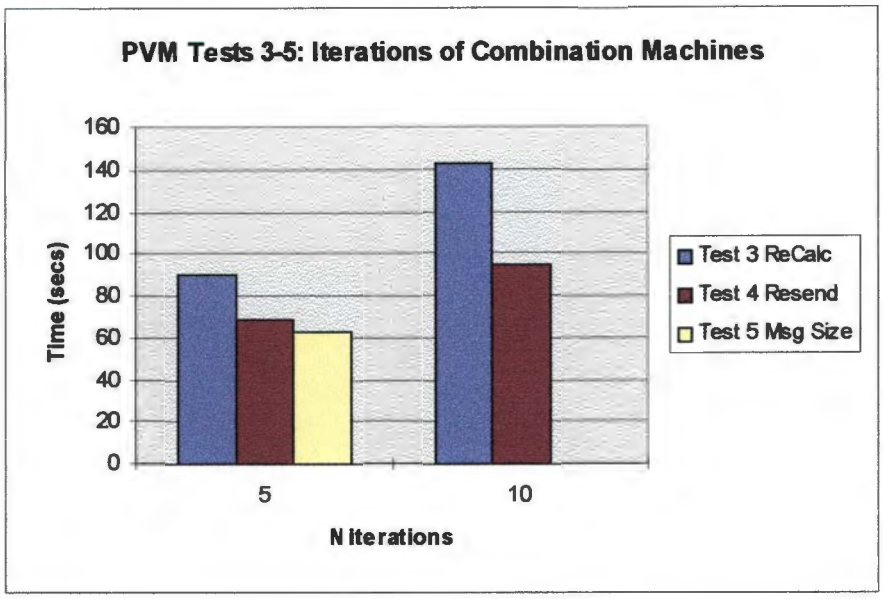


Figure 50: Iterations of calculation, sending and message size on Combinations of Machines

5.1.4.17 Test 6

Test 6 tests the program time including the disk I/O that is required for loading and saving of data files and the reference function computation which adds much computation. These results can be compared to the test 1 result which excludes these operations.

5.1.4.18 Results of Test 6

Table 11 shows the running time of the range and azimuth compression programs when measuring the time of each entire program including the time taken to generate the reference data, read setup files and save the output file.

Table 11: Running Time of Test 6 (rngcomp/azmcomp) in seconds

Test 6	No of slaves per run
Slave Machine	**16**
Pentium	81/143

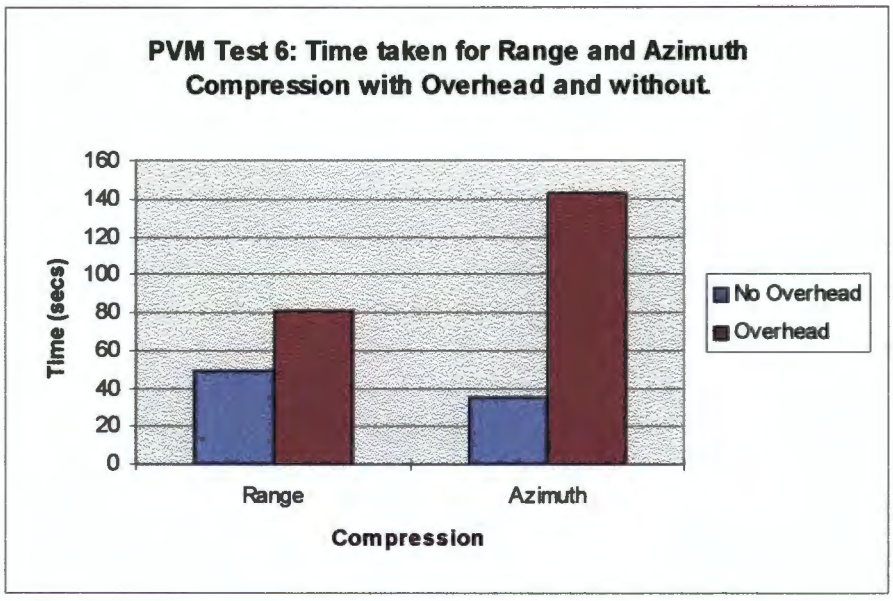


Figure 51: Time taken for Range and Azimuth Compression on Pentium with and without timing the I/O and reference function calculation

5.1.4.19 Conclusions of Test 6

1. We can see that computing the reference functions (which happens once every run only) adds 32 seconds to a range compression run and 108 to an azimuth compression run. Of this run most time is used in the computation of the reference function. The azimuth run computes a 256x512 2-D image and the range compression a 256x1 1-D vector.
2. No speed benefit is assumed by setting up each slave to calculate its own applicable reference function.

5.1.4.20 General Conclusions

1. The less slaves, the quicker the run. This indicates that there are certain resource limits that must be avoided. A maximum number of slaves per host must be determined and enforced. We have seen network, process slots, memory and file handles all limit the performance in the tests above.
2. Setting up of slaves takes longer than the actual processing of the slave. This is related to this computational problem and shows that slaves must perform a minimum amount of computing before they become cost-effective.
3. Processing the entire program on the Linux machine alone (effectively a standard multi-processing situation) shows linear performance until resource limits are reached. In this case when we exceeded 145 slaves. This is true no matter how complex the computational task is.
4. In tests 3 to 5 that increase the amount of computation, resend the messages many times and increase the message length the total execution time is does not grow in a linear fashion. This shows that we are benefiting from using multiple machines. The baseline test of a single range and azimuth compression does not load the virtual machine enough to gain these benefits.

6. CONCLUSIONS

6.1 Conclusions of PVM

Many factors should be taken into account to make the most use of PVM. These factors include the task granularity, the total number of messages conveyed, the number of slaves, system resources, the method of parallelism, the network considerations, load balancing and the time spent on specific functions. These factors are discussed below and conclusions are drawn from the test results as to optimising and using PVM for use in SAR image processing.

6.1.1 Task Granularity

Task granularity is the ratio of the number of bytes received by a process to the number of floating point operations it does i.e. incoming message size vs. work done. Increasing the granularity will speed up the application but the trade-off is a reduction in the available parallelism.

Using the test results to plot the granularity curve only allow us a coarse representation as shown in Figure 52. This curve has been smoothed by the least-squares fit.

The values used to calculate granularity here are the proportions that either the incoming message size changes or that the number of floating points in a single computation changes. The results from test 3 which tests the effect of increasing the computation size by 5 and 10 iterations are used. The results from test 5 which tests the effect of changing the incoming message size to 5 times larger is also used.

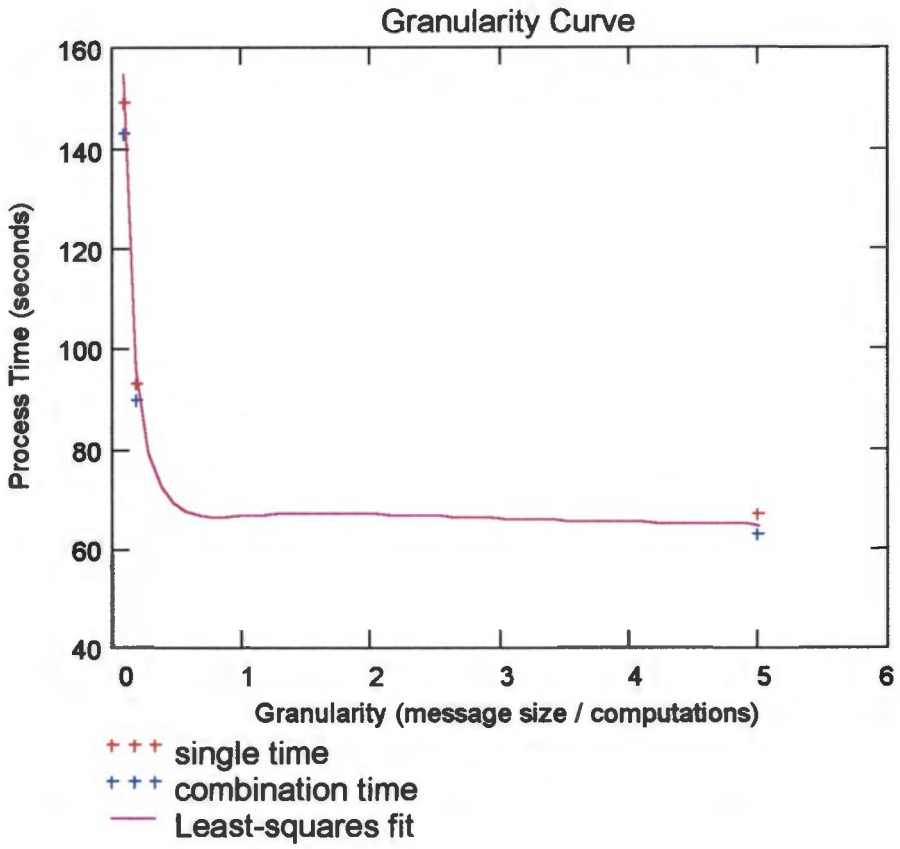


Figure 52: Process Time vs Granularity for Different Configurations

Figure 52 agrees with the above statement that increased granularity speeds up the application. As can be seen, the curve is steep for low granularity when the message size is small compared to the amount of the work done by the slave task. In this region of the curve, the system is not taking true advantage of the PVM environment as the system emulates a single process.

The curve levels out for high granularity when the message size is large compared to the amount of work done. Once the curve levels out, increasing the message size or decreasing the work done is limited in the effect on speed. A maximum message size is dictated by the network environment. A minimum amount of work may be done before the system becomes ineffective.

Note that the curves are different for the single machine vs a combination of machines. The combination of machines makes the most advantage of the PVM environment.

The curves are calculated from results of tests 3 to 5. These tests increase the amount of computation by resending the messages many times and increasing the message length thereby proving that the total execution time does not grow in a linear fashion. This shows that we are benefiting from using multiple machines.

In tests 1 and 2 however, the time taken over three machines is the same or longer than the time for the run on a local machine. This is important as it shows the current task granulation is incorrect for this configuration of the virtual machine. We would need to increase the work per slave and add more machines to the virtual machine. The time gained by using only two additional machines is neutralised by the extra time needed for the PVM and network processing overhead.

Thus, the use of PVM is only effective once the message size is 5 times that of the 256 x 512 messages used in tests 1 and 2 (effectively that used in test 5) and when the computation per task is minimised to that of the same magnitude required in tests 1 and 2.

6.1.2 The total number of messages conveyed

Sending a small number of large messages takes less time than sending a large number of small messages. This statement is a generalisation and the problem is really application specific. Some applications can overlap sending small messages with other computation.

This generalised statement is true of the tests performed here but care must be taken in adopting the idea. These tests were limited by the network for both sending large messages and sending many messages.

Test 5 shows that the nature of the Ethernet network is that is handles heavy loads near its theoretical limit very badly. Here we see that increasing the packet size by 5 times allows better performance by a pool of three machines than the performance of one machine alone. This is ideal and is the whole thrust of PVM. However, increasing the size by 10 times freezes the network I/O system as a result of the network being unable to send the massive packet. Care should be taken in constructing messages not to reach or exceed this limit.

Test 4 shows that large data volumes can quickly overload the network. This means that a certain maximum data through-put should be taken into consideration in designing PVM applications. This will be directly related to network type and it must be noted that the lowest bandwidth section of the network involved will limit the rest of the network. This could be a slow network card, busy network segment, overloaded router or a load on the system preventing the I/O processes from getting CPU time. A load on the system could be caused by an unrelated CPU loads such as a backup running, hardware errors, memory contention or heavy X-windows usage.

Attempts should be made to interleave processing and I/O on each machine to obtain the maximum performance. This could be a difficult task unless it can be automated.

6.1.3 The Number of Slaves

Another factor to be considered is the number of slaves used. The graphs # and # show that both azimuth and range compression execute slower as more slaves are used. The trend is exactly the same for both even though the actual computation and message sizes are completely different.

The less slaves, the quicker the run indicates that there are certain resource limits that must be avoided. A maximum number of slaves per

host must be determined and enforced. We have seen network, process slots, memory and file handles all limit the performance in the tests above.

This idea should be balanced by the fact that, in tests 1 and 2, the setting up of slaves takes longer than the actual processing of the slave. This is related to this computational problem and shows that slaves must perform a minimum amount of computing before they become cost-effective.

6.1.4 System Resources

Each slave should be as frugal as possible with system resources as this can block certain machines in the virtual machine. Such blocking can effectively stop all other processing.

Processing the entire program on the Linux machine alone (effectively a standard single CPU multi-processing situation) shows linear performance until resource limits are reached. In this case when we exceeded 145 slaves. This is true no matter how complex the computational task is.

6.1.5 Functional parallelism vs data parallelism

Compare functional parallelism where different machines do different tasks based on their specific capabilities to data parallelism where the data is distributed to all of the tasks with similar operations in the virtual machine.

In these tests, the data parallelism method was adopted but the results show that a certain amount of functional parallelism may have been used. In the case of azimuth compression, the Alpha machine proved better than the HP machine. This is due to its more modern floating point design.

According to the number of runs to be executed, the azimuth compression should take half the time of the range compression. The Alpha machine shows this relationship well because it finishes the floating point

calculations quickly but the HP machine does comparatively poorly on the azimuth calculation due to its slower floating point performance.

6.1.6 Network considerations

Different computers have different processing power. Machines with the same initial processing power may be disadvantaged due to other multitasking processes currently executing.

Network fluctuations from other applications affected the time results of the tests. The times vary slightly for the machines where the range compression performs better and the azimuth compression doesn't and vice versa.

Even in a dedicated networked environment, with no external use, raw performance is hard to measure since operating system activity, window and filesystem overheads, and administrative network traffic can influence results. The processing power available may change dynamically based on machine loads.

As soon as we send data and start slaves on a remote machine we see a slowdown in the processing for that run. This is due to network related overheads and remote PVM overheads. We see this difference in the times between the local-only slaves on the Pentium and the remote slave startup on the Alpha.

However, the difference between remote slaves on the Alpha and HP is small. This is because both perform the same sequence of operations involving remote slave startup and network I/O.

To combat these problems, load balancing should be applied.

6.1.7 Load Balancing

Load balancing enhances performance by ensuring that each task is doing its fair share of work.

Static load balancing was used in the tests. Many of the above-mentioned problems that were encountered would have been avoided through the use of dynamic load balancing.

If correctly load balanced, optimum results can be achieved regardless of loads on certain machines in the virtual machine. This is essentially using any spare processing on available machines.

Calculating the time spent on specific functions should be considered when balancing loads.

6.1.8 The Time Spent on Specific Functions

We can calculate the times taken for computation and message transmission for the single machine and for the combination of machines. See Table 12 and Table 13 for the times of these respective configurations.

Table 12: Times of Computation per Configuration (seconds)

Test 3	Time for 5 Iterations (sec)	Time for 10 Iterations (sec)	Time per Iteration (sec)
Single Machine	93	149	11.2
Combination of Machines	90	143	10.6

Table 13: Times of Message Transmission per Configuration (seconds)

Test 4	Time for 5 Iterations (sec)	Time for 10 Iterations (sec)	Time per Iteration (sec)
Single Machine	71	94	4.6
Combination of Machines	68	94	5.2

Figure 53 shows the times spent on specific functions for the 2 configurations.

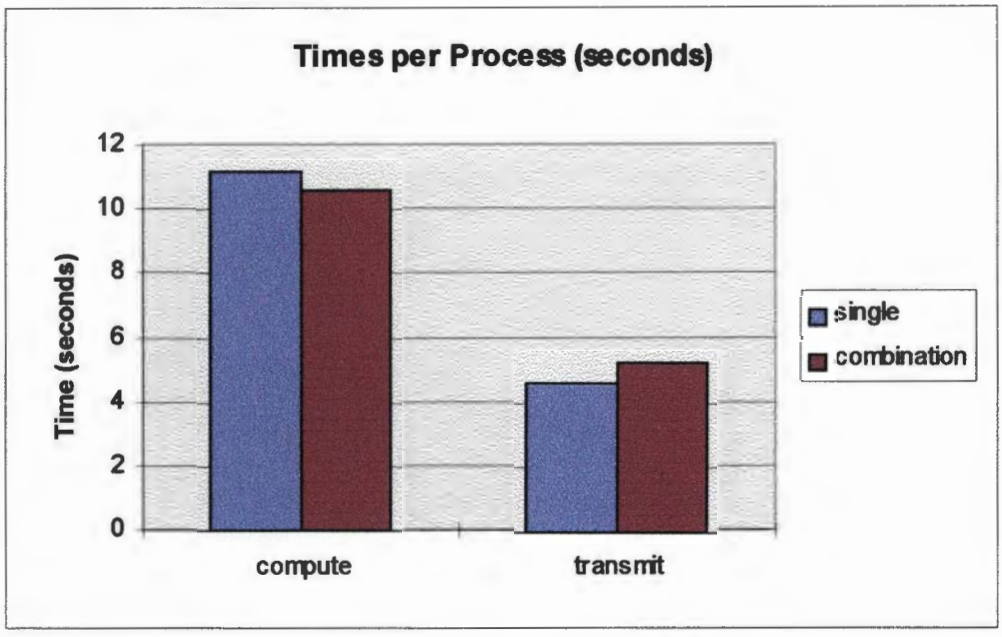


Figure 53: Times Calculated of Specific Functions (seconds)

The single machine is quicker to transmit as there are no network and machine encumbrances. The combination of machines are quicker to compute as it shares the workload over many machines.

We can use these values to gauge the approximate overhead time taken by setting up the slaves, compiling the data and other functions in the processing time.

Table 14 and Table 15 show the overhead times for various tests where test 1 tests the initial setting up of the environment, test 3 tests the computation times and test 4 tests the message transmission times.

Table 14: Times of Overhead Functions on Single Machine (seconds)

	Total Time (sec)	No. Of Computations	No. Of Transmissions	Time of Overhead (sec)
Test 1	49	1	2	28.6
Test 3: 5 Iterations	93	5	2	27.8
Test 3: 10 Iterations	149	10	2	27.8
Test 4: 5 Iterations	71	1	6	32.2
Test 4: 10 Iterations	94	1	11	32.2

Table 15: Times of Overhead Functions on Combination of Machines (seconds)

	Total Time (sec)	No. Of Computations	No. Of Transmissions	Time of Overhead (sec)
Test 1	49	1	2	28
Test 3: 5 Iterations	93	5	2	26.6
Test 3: 10 Iterations	149	10	2	26.6
Test 4: 5 Iterations	71	1	6	26.2
Test 4: 10 Iterations	94	1	11	26.2

As can be seen from these tables, the overhead time is unaffected by the number of iterations performed by a process.

Figure 54 translates these figures into a graph which clearly shows the times spent by these configurations on surplus functions.

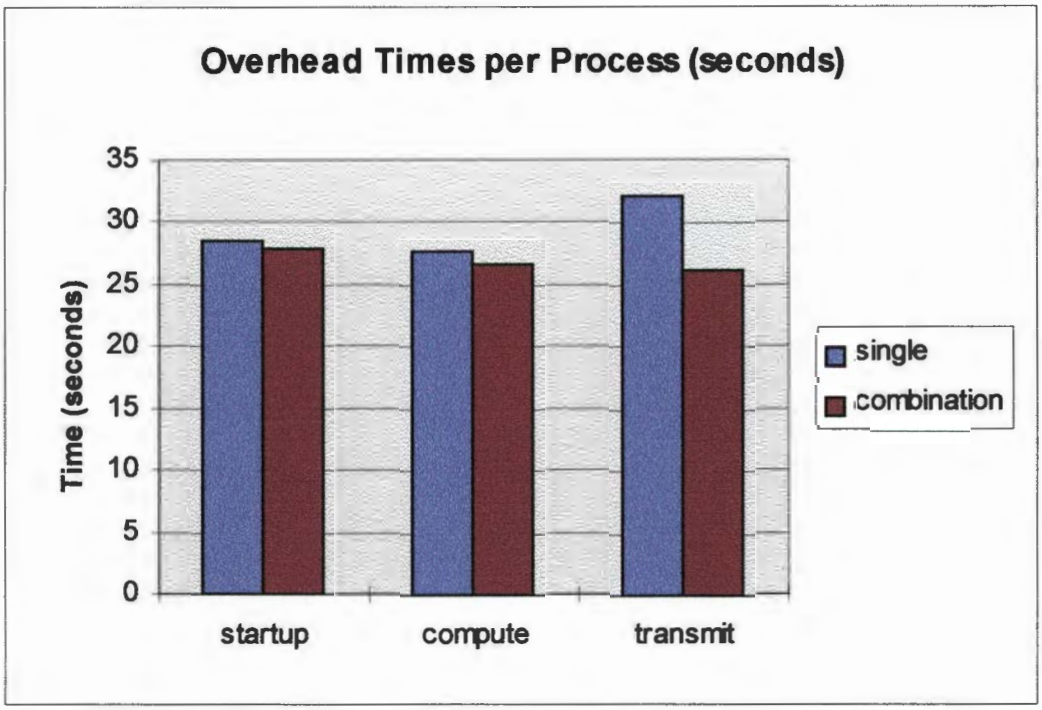


Figure 54: Times Calculated of Overhead when performing Tests on Specific Functions (seconds)

In order from quickest to slowest, the single machine spends the least overhead time on computing, then startup then transmission. The combination of machines spends the least overhead time on transmission, then computing then startup.

Of all these functions, the single machine takes the shortest overhead time performing the iterative computations as it already has the data loaded into memory. This will change to take more time as the data size grows larger. It takes the most overhead time sending messages to itself. This is due to the fact that it has to perform both the sending and receiving operations.

Of all these functions, the combination of machines spends the most overhead time at startup but, once set up, it performs the message

sending and computations quickly as, again, it shares the workload over many machines.

Together Figure 53 and Figure 54 show the following:

1. The single machine spends a lot of overhead time setting up the transmission of messages but, once going, performs them quickly. By comparison, the combination of machines has little message overhead and yet still performs them rapidly once set up.

2. The startup overhead time is much the same for both configurations as the same startup operations occur and all in the master (single) machine.

3. The computing times are better in the combination of machines and should prove even better as larger computing is required.

These show the benefit of using PVM.

6.1.9 In Closing

PVM definitely improves the time taken to process a SAR image once all these factors, especially task granularity, are taken into consideration.

6.2 Future Work with PVM

The following are suggestions for future work planned using PVM for SAR processing:

1. Tests should be performed with real SAR images to test the speed reduction capability.
2. Tests should be performed using larger networks, groups of networks and different types of networks e.g. FDDI or ATM.
3. Tests should be performed using more machines in a heterogeneous or homogeneous environment.
4. Tests should also be done by splitting the reference function computation over a PVM network. In the case of range compression, each slave would be computing the same reference function so no speed advantage would be seen. In the case of azimuth compression, each slave would be computing its own distinct reference function although parts of the computation would have the same parameter values. If this test is performed on azimuth compression the results may show that PVM is quicker especially in the case of large images.
5. Tests should experiment with using all facilities available in PVM e.g. fault tolerance hooks, signals and tracing.

6.3 Conclusions of SAR IMAGE COMPRESSION:

Sections 3.3 and 4.3 show that the compression accurately pin-pointed the point target location and produced results that agree with the theory.

The compression produced the same results from two different radar simulation programs.

6.4 Future Work with SAR IMAGE COMPRESSION

The following are suggestions for the refinement of the SAR image compression software:

1. The compression programs should be refined to compensate for other effects that influence SAR images. These effects include range curvature, azimuth migration and motion.
2. The program should also be tried with real SAR images to test the compression performance.
3. The compression set-up data should use more realistic parameter values as these have not been used for the sake of demonstration.

The aim of this thesis was to demonstrate that the use of PVM improves the time taken to process a SAR image. This has been shown to be true.

7. APPENDIX A: RADSIM VERSION 4 TEST PARAMETERS

```
! Simulation begins

$GENERAL  A
1       ! [ ] Number of Pulse Repetition Intervals to Simulate

180.00          ! [Hz] Pulse Repetition Frequency

256     ! [ ] Number of range samples to simulate

0.15    ! [GHz] Simulation frequency


$TARGET   A

0.00    ! [m] Target x direction offset from mid path

20000.00        ! [m] Ground range at closest approach

0.00    ! [m] Target height above ground level

5.00    ! [m^2] Target radar cross section


$GEOMETRY G1 A

NULL  ! Use which motion data ?

10000.00        ! [m] Aircraft altitude

246.00          ! [m/s] Aircraft velocity
```

! $WRITE G1, ASC, OVR, G1.ASC

$PULSEGEN T1 A1 ! Generate pulse waveform data and put into array A1 and T1

A ! Use which GENERAL data ?

100.0 ! [ns] Rise time

100.0 ! [ns] Fall time

600.0 ! [ns] Pulse width

10.0 ! [V] Peak amplitude

CHIRP ! CHIRP or MONO

0.1 ! [GHz] Chirp Bandwidth (arb non-zero for MONO)

$WRITE A1, ASC, APP, A1.ASC

$WRITE T1, ASC, APP, T1.ASC

$FFT F1 T1

$WRITE F1, ASC, APP, F1.ASC

```
$MATCHFILT F2  ! Output

F1      ! Input

0.08    ! Windowing Factor


$WRITE F2, ASC, APP, F2.ASC


$RETURNL T2 G1 A1

1.0     ! [kW] Radar Transmitted Power

0.141   ! [GHz] Radar centre frequency

7.0     ! [dB] Antenna gains

0.0     ! [dB] Losses ( TxLoss + RxLoss + PropLoss )

1.0     ! [m^2] Radar cross sectional area

22300.0         ! [m] Range offset for geometry setup

CHIRP           ! Chirp or Mono

0.1     ! [GHz] Chirp Bandwidth (arb non-zero for MONO)

NULL    ! SIN or NULL antenna type

60.0    ! [deg] Elevation beamwidth

7.16    ! [deg] Azimuth beamwidth
```

-40.0 ! [deg] Elevation angle (down is negative)

0.0 ! [deg] Fixed squint angle (forward is positive)

$AMPLIFY T2 T2 10000000.0

$WRITE T2, ASC, APP, T2.ASC

$FFT F3 T2

$WRITE F3, ASC, APP, F3.ASC

$MULTIPLY F4 F3 F2

$WRITE F4, ASC, APP, F4.ASC

$FFT T3 F4

$WRITE T3, ASC, APP, T3.ASC

$A2D D1 T3

240 ! [V] LSB value

8 ! [bits] number of bits of A2D precision (4, 8 or 16)

0 ! [bin] Starting bin (inclusive)

0 ! [bins] How many bins to process after starting bin

(0 means process until end of range line)

```
1       ! [samples/bin] skip how many samples for one output bin ?


$WRITE D1, ASC, APP, D1RAD.ASC

$ENDLOOP

! End of simulation
```

8. APPENDIX B: SARSIM VERSION 060597 TEST PARAMETERS

```
1 ! Radar Simulator SARSIM version B2B (c) 1997 R.L.
2
3
4 $PLATFORM  Earth
5 STATIONARY
6 0                     ! (m) X-Position
7 0                     ! (m) Y-Position
8 0                     ! (m) Z-Position
9 0                     ! (deg) X-axis Rotation
10 0                    ! (deg) Y-axis Rotation
11 0                    ! (deg) Z-axis Rotation
12 STDDEV
13 0                    ! (m) X-Position standard deviation
14 0                    ! (m) Y-Position standard deviation
15 0                    ! (m) Z-Position standard deviation
16 0                    ! (deg) X-axis Rotation standard deviation
17 0                    ! (deg) Y-axis Rotation standard deviation
18 0                    ! (deg) Z-axis Rotation standard deviation
19
20 $PLATFORM  PLANE
21 TRAJECTORY
22 INLINE
23 2                    ! Number of samples for Position X (m) vs Time (s)
24 0, -122.5  1, 122.5
25 1                    ! Number of samples for Position Y (m) vs Time (s)
26 0, 20000
27 1                    ! Number of samples for Position Z (m) vs Time (s)
28 0, 10000
29 NOT ALIGNED
30 0                    ! (deg) X-axis Rotation
31 0                    ! (deg) Y-axis Rotation
32 0                    ! (deg) Z-axis Rotation
33 STDDEV
34 0                    ! (m) X-Position standard deviation
35 0                    ! (m) Y-Position standard deviation
36 0                    ! (m) Z Position standard deviation
37 0                    ! (deg) X-axis Rotation standard deviation
38 0                    ! (deg) Y-axis Rotation standard deviation
39 0                    ! (deg) Z-axis Rotation standard deviation
40
41
42 $TARGET EARTH  ! Platform
43 0, 0, 0  ! Position X,Y,Z (m)
44 0, 0, 0  ! Position standard deviation X,Y,Z (m)
45 1, 0  ! Radar cross section (m*m), RCS std. dev. (m*m)
46 ISOTROPIC
47
48
49 $RADAR
50 RADAR1                 ! Name of radar
51 PLANE                 ! Platform name of radar
52 CHIRP
53 0.1                   ! (GHz) Chirp bandwidth
54 700                   ! (ns)  zero to zero Pulse width
55 RECT                  ! Rectangular envelope
56 CONSTANT              ! Constant PRI
57 180                   ! (Hz)  Pulse repetition frequency
58 SINGLE                ! Constant frequency
59 0.141                 ! (GHz)  Center frequency
60 1                     ! (KW)  Power output
61 0                     ! (dB)  Total system losses
62 0                     ! (K)   Noise temperature
63 ISOTROPIC             ! Isotropic transmitter antenna
64 SAME                  ! Same as transmitter antenna
65 FIXED                 ! Fixed antenna direction
66 0                     ! (deg) Beam-direction - Elevation
```

```
1 0                     ! (deg) Beam-direction - Azimuth
2 RECT                  ! Rectangular MF window
3 NO STC                ! no STC
4
5
6 $SIMULATION
7 RADAR1                ! Name of radar
8 22000                 ! (m) Slant range start
9 23000                 ! (m) Slant range end
10 -122.5               ! (s) Azimuth range start
11 122.5                ! (s) Azimuth range end
12 8                    ! A/D bit accuracy
13 0.15                 ! (GHz) Sample frequency
14 1.6B257E-07          ! (mV) Least Significant Bit value
15 ASCII                ! Save file format
16 RAW                  ! Processing
17 SIM1.ASC             ! File Name of output file
18                      ! Size : 512 (Azimuth) x 256 (Slant Range) points
```

9. APPENDIX C: CODE FOR PVM COMPRESSION SOFTWARE

```c
/*-------------------------------------------------------------------------*/
/* PvmAzm generates the 2-D azimuth compression reference parameters by calculating the geometry,  */
/* phase shift and other effects on the SAR image.                         */
/*-------------------------------------------------------------------------*/
#include "globals.h"


int PVMAzm(void)
  {
  int i, j, n;
  float Phase, TimeShift, Re, Im, Delay, Theta;                  /* chirp pulse parameters */
  float x,y,z;                                                   /* geometrical parameters */
  float Gain, GainPower;                                         /* gain parameters */
  float Geom[MAXAZM], ReRng[MAXRNG], ImRng[MAXRNG];
  float ReAz, ImAz;
  FILE *Afile, *Zfile;



  /* GET RANGE COMPRESSION PULSE */
  Afile = fopen("/home/swuyts/pvmwork/rngspike.asc","rt");
  if (!Afile)
    {
    printf("Cannot open rngspike input file\n");
    return(0);
    }
  for ( i=0 ; i<RngBin; i++)
    if (!feof(Afile))
      fscanf(Afile,"%f %f", &ReRng[i], &ImRng[i]);
    else
      {
      printf("Not enough data in rngspike.asc!\n");
      fclose(Afile);
      exit(1);
      }
  fclose(Afile);



  /* SAVE RETURN OUTPUT */
  Zfile = fopen("/home/swuyts/pvmwork/azm.asc","wt");
  if (!Zfile)
    {
    printf("Cannot open return output file\n");
    return(0);
    }


  /* SET UP GEOMETRY */
  x = -0.5* (AzmBin-1)* SourceV/PRF;
  y = TargetY;
  z = SourceZ - TargetZ;
  for ( j=0 ; j<AzmBin; j++)
    {
    Geom[j] = sqrt(x*x + y*y + z*z);
    x+=SourceV/PRF;
    }


  /* INITIAL CALCULATIONS */
  GainPower = CLight / (CentreFreq * 1.0e+09) *
                       sqrt(TxPower * 1000.0 * RCS /
                      (64 * PI * PI * PI * pow(10, Losses/10.0)));

  /* AT EACH RANGE BIN . */
  for (i=0 ; i<RngBin; i++)
    {
    /* . CALCULATE AZIMUTH SHIFT ..*/
    for ( j=0 ; j<AzmBin; j++)
      {

      /* ..CALCULATE GAIN */
      Gain = pow(10, AntGain/10.0) * Amplify * GainPower /
                (Geom[j]*Geom[j]);

      /* ..CALCULATE AZIMUTH PHASE SHIFT */
      TimeShift = 2 * (Geom[j] - RngOffset) / CLight;
      Phase = -2 * PI * CentreFreq * 1.0e9 * TimeShift;


      /* ..CALCULATE RETURN DELAY */
      Delay = (int) (TimeShift * SimFreq * 1.0e+9 + 0.5);

      n = i - (int)Delay;
      if ((n < 0)||(n > 250))
        Re = Im = 0.0;
      else
        {
        /* ..CHIRP PULSE */
        Re = ReRng[n] * Gain;
        Im = ImRng[n] * Gain;
        }

      /* ..RESULT */
      ReAz = (Re*cos(Phase) - Im*sin(Phase));
      ImAz = (Re*sin(Phase) + Im*cos(Phase));

      /* ..PRINT RESULT */
      fprintf(Zfile,"%6.6f  %6.6f\n", ReAz, ImAz);

      }
    }

  fclose(Zfile);

  return(1);
  }


/*=========================================================================*/
```

```
1  /*-------------------------------------------------------------------------------------------------------------*/
2  /*                                                                                                             */
3  /*-------------------------------------------------------------------------------------------------------------*/
4  /* Author: Shirley Wuyts                                                                                       */
5  /* Date:  August 1997                                                                                          */
6  /* Project: PVMCOMP                                                                                            */
7  /* Task: AzmComp.c                                                                                             */
8  /*                                                                                                             */
9  /* Description: This function acts as a PVM slave task for performing azimuth compression on a one             */
10 /* dimensional image vector.                                                                                   */
11 /*                                                                                                             */
12 /* The slave reads the setup file then sends/receives compression data from the PVM master.                    */
13 /* Each slave is given a single range vector of azimuth values to compare with the reference data.             */
14 /*-------------------------------------------------------------------------------------------------------------*/
15
16 /*-------------------------------------------------------------------------------------------------------------*/
17 /* Include Files                                                                                               */
18 /*-------------------------------------------------------------------------------------------------------------*/
19 #include "globals.h"
20
21 /*-------------------------------------------------------------------------------------------------------------*/
22 /* Exported Variable Definitions                                                                               */
23 /*-------------------------------------------------------------------------------------------------------------*/
24 int RngBin, AzmBin, AzmFilt;                                        /* range and azimuth dimensions */
25
26 /*-------------------------------------------------------------------------------------------------------------*/
27 /* Module Macro Definitions                                                                                    */
28 /*-------------------------------------------------------------------------------------------------------------*/
29 #define MAXVARS 3
30
31 /*-------------------------------------------------------------------------------------------------------------*/
32 /* Module Variable Definitions                                                                                 */
33 /*-------------------------------------------------------------------------------------------------------------*/
34 char *vars[MAXVARS] = {"RNGBIN", "AZMBIN", "AZMFILT"};
35 float reWin[MAXAZM*2], imWin[MAXAZM*2];                                  /* window vars */
36 float reRef[MAXAZM*2], imRef[MAXAZM*2];                                /* reference vars */
37 float reInp[MAXAZM*2], imInp[MAXAZM*2];                                  /* input vars */
38 float reOut[MAXAZM*2], imOut[MAXAZM*2];                            /* matched output vars */
39 char  destination[64];
40 int me;                                                                  /* slave ID */
41
42 /*-------------------------------------------------------------------------------------------------------------*/
43 /* Function Definitions                                                                                        */
44 /*-------------------------------------------------------------------------------------------------------------*/
45 int GetInitFile (void);
46 void Comp(void);
47 void AzmCompOutput(float *rePrn, float *imPrn);
48 void fft(float *re, float *im, int bin, int dir);
49
```



```
1
2  /*-------------------------------------------------------------------------------------------------------------*/
3  /* AzmComp calls the function GetInitFile for initialising the program parameters.                             */
4  /* AzmComp receives the azimuth data in a single range vector and the reference data in that range vector      */
5  /* from the PVM master.                                                                                        */
6  /* AzmComp calls the function Comp to perform azimuth compression on these two vectors.                        */
7  /*-------------------------------------------------------------------------------------------------------------*/
8
9
10 int main()
11   {
12   int i,j;
13
14   int mytid;                                                                        /* my task id */
15   int master, msgtype;                                                             /* comms stuff */
16
17
18
19   /* INITIALISE ARRAYS AND ZERO PAD */
20   for (i=0; i<(AzmBin*2); i++)
21     {
22     reWin[i] = imWin[i] = 0.0;
23     reRef[i] = imRef[i] = 0.0;
24     reInp[i] = imInp[i] = 0.0;
25     reOut[i] = imOut[i] = 0.0;
26     }
27
28
29
30   /* SET UP INITIALISATION FILE - READ FROM INI FILE */
31   if (!GetInitFile())
32     {
33     printf("LINUX slave : (azmcomp) problem with Initialising File");
34     return(0);
35     }
36
37
38
39   /* ENROLL SLAVE IN PVM */
40   mytid = pvm_mytid();                                                              /* my task id */
41
42
43
44   /* RECEIVE DATA FROM MASTER */
45   msgtype = 0;
46   pvm_recv( -1, msgtype );                                                  /* receive msg type 0 */
47   pvm_upkint(&me, 1, 1);                                                          /* slave ID */
48   pvm_upkfloat(reRef, AzmBin, 1);                                        /* real reference vector */
49   pvm_upkfloat(imRef, AzmBin, 1);                                        /* imag reference vector */
50   pvm_upkfloat(reInp, AzmBin, 1);                                          /* real input vector */
51   pvm_upkfloat(imInp, AzmBin, 1);                                          /* imag input vector */
52
53
54
55   /* DO AZIMUTH COMPRESSION CALCULATIONS */
56   Comp();
57
58
59
60   /* SEND RESULT TO MASTER */
61   pvm_initsend( PvmDataDefault );                                          /* set up send buffer */
62   pvm_pkint(&me, 1, 1);                                                     /* send my slave id */
63   pvm_pkfloat(reOut, (AzmBin*2), 1);                                        /* send real result */
64   pvm_pkfloat(imOut, (AzmBin*2), 1);                                        /* send imag result */
65   msgtype = 5;
66   master = pvm_parent();
```

```
  pvm_send( master, msgtype );                                  /* send msg type 5 to master */


  /* TASK FINISHED. EXIT PVM BEFORE STOPPING */
  pvm_exit();
  return(1);
  }

/*-------------------------------------------------------------------------------------------------------*/
/* Comp achieves azimuth compression by windowing the reference function, converting both the windowed   */
/* reference function and the input function to the frequency domain (by performing Fast Fourier Transforms)*/
/* , multiplying the two, then converting back to the time domain.                                       */
/* To perform the fast fourier transforms, Comp calls the function fft.                                  */
/*-------------------------------------------------------------------------------------------------------*/

void Comp(void)
  {
  int j,isign;
  float BW;
  int LeftFilt, RightFilt;


  if (me == RngBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/azminp.asc");
    AzmCompOutput((float *)reInp,(float *)imInp);
    sprintf(destination,"/home/swuyts/pvmwork/azmref.asc");
    AzmCompOutput((float *)reRef,(float *)imRef);
    }


  /* WINDOW REFERENCE FUNCTION */
  BW = (float)(2 * AzmFilt);
  LeftFilt = AzmBin/2 - AzmBin +1;
  RightFilt = AzmBin/2 + AzmFilt;
  for (j=0; j<(AzmBin*2); j++)
    {
    if ((j >= LeftFilt) && (j < RightFilt))
      {
      reWin[j]=(0.54+0.46*cos(2*PI*((float)j-BW/2.0-(float)LeftFilt)/BW));
      imWin[j]= reWin[j]*(-1);
      }
    reRef[j] *= reWin[j];
    imRef[j] *= imWin[j];
    }

  if (me == RngBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/azmwin.asc");
    AzmCompOutput((float *)reWin,(float *)imWin);
    sprintf(destination,"/home/swuyts/pvmwork/azmrefw.asc");
    AzmCompOutput((float *)reRef,(float *)imRef);
    }


  /* PERFORM FOURIER TRANSFORM ON REFERENCE DATA */
  isign=1;
  fft(reRef,imRef,(AzmBin*2),isign);

  if (me == RngBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/azmreff.asc");
    AzmCompOutput((float *)reRef,(float *)imRef);
    }

```



```

  /* PERFORM FOURIER TRANSFORM ON INPUT DATA */
  isign=1;
  fft(reInp,imInp,(AzmBin*2),isign);

  if (me == RngBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/azminpf.asc");
    AzmCompOutput((float *)reInp,(float *)imInp);
    }


  /* MULTIPLY BY REFERENCE FUNCTION */
  for(j=0; j<(AzmBin*2); j++)
    {
    reOut[j]=reRef[j]*reInp[j]-imRef[j]*imInp[j];
         imOut[j]=imRef[j]*reInp[j]+reRef[j]*imInp[j];
    }

  if (me == RngBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/azmcompf.asc");
    AzmCompOutput((float *)reOut,(float *)imOut);
    }


  /* PERFORM IFFT ON ROW OF OUTPUT DATA */
  isign=-1;
  fft(reOut,imOut,(AzmBin*2),isign);

  }

```

```
1
2 /*-------------------------------------------------------------------------------------------------*/
3 /* GetInitFile reads the setup file pvmcomp.ini for initialising the Azimuth Compression parameters.  */
4 /*-------------------------------------------------------------------------------------------------*/
5
6 int GetInitFile (void)
7   {
8   char inpfile[]="/home/swuyts/pvmwork/pvmcomp.ini";
9   FILE *Ifile;
10
11  char name[20];
12  int i = 0;
13
14
15  /* READ INITIALISATION FILE */
16  Ifile = fopen(inpfile,"rt");
17  if (!Ifile)
18    {
19    printf("LINUX slave : (azmcomp) Cannot open %s\n", inpfile);
20    return(0);
21    }
22
23
24  fscanf(Ifile,"%s %d", &name, &RngBin);
25  if ((strcmp(vars[i++], name))!=0)
26    {fclose(Ifile); return 0;}
27
28  fscanf(Ifile,"%s %d", &name, &AzmBin);
29  if ((strcmp(vars[i++], name))!=0)
30    {fclose(Ifile); return 0;}
31
32  fscanf(Ifile,"%s %d", &name, &AzmFilt);
33  if ((strcmp(vars[i++], name))!=0)
34    {fclose(Ifile); return 0;}
35
36
37  fclose(Ifile);
38  return(1);
39  }
40
41
```



```
1
2 /*-------------------------------------------------------------------------------------------------*/
3 /* AzmCompOutput prints the output of the azimuth compression processes into the file "destination" named  */
4 /* when called by the Comp routine.                                                                */
5 /*-------------------------------------------------------------------------------------------------*/
6
7 void AzmCompOutput( float *rePrn, float *imPrn)
8   {
9   int j;
10  FILE *outf;
11
12  outf = fopen(destination,"wt");
13  if (!outf)
14    {
15    printf("LINUX slave : (azmcomp) Cannot open file %s for writing!\n",destination);
16    exit(1);
17    }
18
19  for (j=0; j<(AzmBin*2); j++)
20        fprintf(outf,"%12.6f %12.6f\n",*(rePrn + j),*(imPrn + j));
21
22  fclose(outf);
23  }
24
25 /*=================================================================================================*/
```

```c
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*-------------------------------------------------------------------------*/
/* Author: Shirley Wuyts                                                   */
/* Date:  August 1997                                                      */
/* Project: PVMCOMP                                                        */
/* Task: CnrTurn.c                                                         */
/*                                                                         */
/* Description: CnrTurn reads the setup values and input data then transposes the input data matrix of    */
/* [AZMBIN][RNGBIN] to [RNGBIN][AZMBIN] and saves to a file.               */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/* Include Files                                                           */
/*-------------------------------------------------------------------------*/
#include "globals.h"

/*-------------------------------------------------------------------------*/
/* Exported Variable Definitions                                           */
/*-------------------------------------------------------------------------*/
int RngBin, AzmBin,                                 /* range and azimuth dimensions */

/*-------------------------------------------------------------------------*/
/* Command Line Variable Definitions                                       */
/*-------------------------------------------------------------------------*/
char InputFile[128], OutputFile[128];

/*-------------------------------------------------------------------------*/
/* Module Macro Definitions                                                */
/*-------------------------------------------------------------------------*/
#define MAXVARS 2

/*-------------------------------------------------------------------------*/
/* Module Variable Definitions                                             */
/*-------------------------------------------------------------------------*/
char *vars[MAXVARS] = {"RNGBIN", "AZMBIN"};

/*-------------------------------------------------------------------------*/
/* Function Definitions                                                    */
/*-------------------------------------------------------------------------*/
int GetInitFile (void);
```



```c

/*-------------------------------------------------------------------------*/
/* CnrTurn reads the setup values and input data then transposes the input data matrix of [AZMBIN][RNGBIN]  */
/* to [RNGBIN][AZMBIN] and saves to a file.                                */
/*-------------------------------------------------------------------------*/

int main(int argc, char *argv[])
  {
  int i,j;
  float reInp[MAXRNG][MAXAZM], imInp[MAXRNG][MAXAZM];                    /* input vars */
  FILE *Cfile, *Gfile;

  printf("CNRTURN execution started!\n");

  if (argc != 3)
    {
    printf("Usage error!\n");
    printf("Expecting input file and output file on command line\n");
    return(0);
    }
  else
    {
    strncpy(InputFile,argv[1],128);
    strncpy(OutputFile,argv[2],128);
    printf("Command line read:\n");
    printf("\tInput : %s\n",InputFile);
    printf("\tOutput: %s\n",OutputFile);
    }


  /* READ SETUP FILE */
  printf("READ FROM CNRTURN SETUP FILE pvmcomp.ini\n");
  if (!GetInitFile())
    {
    printf("pvmcomp.ini parameters not in correct order\n");
    return(0);
    }


  /* READ INPUT FILE */
  printf("READ FROM CNRTURN INPUT FILE %s\n",InputFile);
  Cfile = fopen(InputFile,"rt");
  if (!Cfile)
    {
    printf("Cannot open cnrturn input file %s\n",InputFile),
    return(0);
    }


  /* PERFORM CORNER TURN */
  for ( j=0; j<AzmBin ; j++ )
    for ( i=0; i<RngBin ; i++ )
      if (!feof(Cfile))
        fscanf(Cfile,"%f %f", &reInp[i][j], &imInp[i][j]);
      else
        {
        printf("Not enough data in input file %s\n",InputFile);
        fclose(Cfile);
        exit(1);
        }

  fclose(Cfile);
```

```
/* SAVE CNRTURN OUTPUT FILE */
printf("SAVE TO CNRTURN OUTPUT FILE %s\n",OutputFile);
Gfile = fopen(OutputFile,"wt");
if (!Gfile)
  {
  printf("Cannot open cnrturn output file %s\n",OutputFile);
  return(0);
  }

for ( i=0; i<RngBin ; i++ )
  for ( j=0; j<AzmBin ; j++ )
    fprintf(Gfile,"%6.0f  %6.0f\n", reInp[i][j], imInp[i][j]);

fclose(Gfile);
return(1);
}
```

```

/*-------------------------------------------------------------------------------------------------*/
/* GetInitFile reads the setup file pvmcomp.ini for initialising the Presumming parameters        */
/*-------------------------------------------------------------------------------------------------*/

int GetInitFile (void)
  {
  char inpfile[]="/home/swuyts/pvmwork/pvmcomp.ini";
  FILE *Ifile;

  char name[20];
  int i = 0;


  /* READ INITIALISATION FILE */
  Ifile = fopen(inpfile,"rt");
  if (!Ifile)
    {
    printf("Cannot open %s\n", inpfile);
    return(0);
    }

  fscanf(Ifile,"%s %d", name, &RngBin);
  if ((strcmp(vars[i++], name))!=0)
    {fclose(Ifile); return 0;}

  fscanf(Ifile,"%s %d", name, &AzmBin);
  if ((strcmp(vars[i++], name))!=0)
    {fclose(Ifile); return 0;}

  fclose(Ifile);
  return(1);
  }

/*=================================================================================================*/
```

```
1 pentium 32
2 alpha 16
3 hppa1 16
```

```
/*-------------------------------------------------------------------------------------------------------------*/
/* fft performs a Fast Fourier Transform algorithm                                                             */
/*-------------------------------------------------------------------------------------------------------------*/
#include "globals.h"

/*-------------------------------------------------------------------------------------------------------------*/
/* Module Variable Definitions                                                                                 */
/*-------------------------------------------------------------------------------------------------------------*/

static float cosarray[28] = {
/* cos(-2pi/N) for N = 2,4,8,....,16384 */
-1.00000000000000, 0.00000000000000, 0.70710678118655,
 0.92387953251129, 0.98078528040323, 0.99518472667220,
 0.99879545620517, 0.99969881869620, 0.99992470183914,
 0.99998117528260, 0.99999529380958, 0.99999882345170,
 0.99999970586288, 0.99999992646572,
/* cos(2pi/N) for N = 2,4,8,....,16384 */
-1.00000000000000, 0.00000000000000, 0.70710678118655,
 0.92387953251129, 0.98078528040323, 0.99518472667220,
 0.99879545620517, 0.99969881869620, 0.99992470183914,
 0.99998117528260, 0.99999529380958, 0.99999882345170,
 0.99999970586288, 0.99999992646572
};

static float sinarray[28] = {
/* sin(-2pi/N) for N = 2,4,8,....,16384 */
 0.00000000000000,-1.00000000000000,-0.70710678118655,
-0.38268343236509,-0.19509032201613,-0.09801714032956,
-0.04906767432742,-0.02454122852291,-0.01227153828572,
-0.00613588464915,-0.00306795676297,-0.00153398018628,
-0.00076699031874,-0.00038349518757,
/* sin(2pi/N) for N = 2,4,8,....,16384 */
 0.00000000000000, 1.00000000000000, 0.70710678118655,
 0.38268343236509, 0.19509032201613, 0.09801714032956,
 0.04906767432742, 0.02454122852291, 0.01227153828572,
 0.00613588464915, 0.00306795676297, 0.00153398018628,
 0.00076699031874, 0.00038349518757
};

/*-------------------------------------------------------------------------------------------------------------*/
/* Function Definitions                                                                                        */
/*-------------------------------------------------------------------------------------------------------------*/
static unsigned int shuffleindex(unsigned int i, int wordlength);
static void shuffle2arr(float *a, float *b, int bitlength);
void fft(float *re, float *im, int bin, int dir);


/*-------------------------------------------------------------------------------------------------------------*/
/* fft performs a Fast Fourier Transform algorithm.  "re" and "im" point to the start of the real and          */
/* imaginary arrays of numbers.  "pwr" holds the array sizes as a power of 2 while "dir" indicates whether    */
/* an FFT (dir positive >-1) or an inverse FFT (dir negative <= 0) must be performed.                          */
/* fft returns the transformed information at "re" and "im" (real and imaginary).                              */
/*-------------------------------------------------------------------------------------------------------------*/
void fft(float *re, float *im, int bin, int dir)
{
   int pwr;
   int pwrhelp;
   int n;
   int section;
   int anglecounter;
   int flydistance;
   int flycount;
   int index1;
   int index2;
   float tempr,tempi;
   float c,s;
   float scale;
   float temp;
   float qr,qi;


   pwr = (int)(log10((float)bin)/log10(2.0));

   shuffle2arr(re,im,pwr);  /* shuffle before (I)FFT */

   pwrhelp = pwr;

   /* determine size of arrays */
   n = 1;
   do
   {
      n = n * 2;
      pwrhelp--;
   } while (pwrhelp > 0);
     .
   if (dir >= 1) anglecounter = 0;  /* FFT */
   else anglecounter = 14;          /* IFFT */

   section = 1;
   while (section < n)
   {
      flydistance = 2 * section;
      c = cosarray[anglecounter];
      s = sinarray[anglecounter];
      qr = 1.0;
      qi = 0.0;

      for (flycount = 0; flycount <= section - 1; flycount++)
      {
         index1 = flycount;
         do
         {
            index2 = index1 + section;

            /* perform 2 point DFT */
            tempr = 1.0 * qr * re[index2] - 1.0 * qi * im[index2];
            tempi = 1.0 * qr * im[index2] + 1.0 * qi * re[index2];

            re[index2] = re[index1] - tempr;
            re[index1] = re[index1] + tempr;
            im[index2] = im[index1] - tempi;
            im[index1] = im[index1] + tempi;

            index1 = index1 + flydistance;
         } while (index1 <= (n-1));

/*                 k                                                */
/* calculate new q  = cos(ak) + j * sin(ak) = qr + j * qi  */
/*           -2*pi                                                  */
/* with: a=--------                                              *  */
/*            n                                                   */

         temp = qr;
         qr = qr * c - qi * s;
         qi = qi * c + temp * s;
      }

      section = section * 2;
      anglecounter++;
   }

   if (dir <= 0)  /* normalise for inverse FFT only */
```

```
   {
      scale = 1.0 / n;

      for (index1 = 0; index1 <= n - 1; index1++)
      {
         re[index1] = scale * re[index1];
         im[index1] = scale * im[index1];
      }
   }
}
/*---------------------------------------------------------------------------------------------------------*/
/* shuffleindex finds the shuffle index of array elements. The array length must be a power of 2.         */
/* The power is stored in "wordlength".                                                                    */
/* shuffleindex returns "i" the destination index for shuffling.                                           */
/*---------------------------------------------------------------------------------------------------------*/
static unsigned int shuffleindex(unsigned int i, int wordlength)
{
   unsigned int newindex;
   unsigned char bitnr;

   newindex = 0;

   for (bitnr = 0; bitnr <= wordlength - 1; bitnr++)
   {
      newindex = newindex << 1;
      if ((i & 1) != 0)
         newindex = newindex + 1;
      i = i >> 1;
   }

   return newindex;
}

/*---------------------------------------------------------------------------------------------------------*/
/* shuffle2arr shuffles both arrays "a" and "b".  This function is called before performing the actual fft */
/* so the array elements are in the correct order after fft.                                               */
/*---------------------------------------------------------------------------------------------------------*/
static void shuffle2arr(float *a, float *b, int bitlength)
{
   unsigned int indexold, indexnew;
   float temp;
   unsigned int n;
   int bitlengthtemp;

   bitlengthtemp = bitlength;  /* save for later use */

   /* find array length */
   n = 1;
   do
   {
      n = n * 2;
      bitlength = bitlength - 1;
   } while (bitlength > 0);

   /* shuffle all elements */
   for (indexold = 0; indexold <= n - 1; indexold++)
   {
      /* find index to exchange elements */
      indexnew = shuffleindex(indexold, bitlengthtemp);
      if (indexnew > indexold)
      {
         /* exchange elements of a[] */
         temp = a[indexold];
         a[indexold] = a[indexnew];
         a[indexnew] = temp;

         /* exchange elements of b[] */
         temp = b[indexold];
         b[indexold] = b[indexnew];
         b[indexnew] = temp;
      }
   }
}

/*---------------------------------------------------------------------------------------------------------*/
```

```
 1 /*-------------------------------------------------------------------------------------------------------*/
 2 /*                                                                                                       */
 3 /*-------------------------------------------------------------------------------------------------------*/
 4 /* Author: Shirley Wuyts                                                                                 */
 5 /* Date:  August 1997                                                                                    */
 6 /* Project: PVMCOMP                                                                                      */
 7 /* Task: Globals.h                                                                                       */
 8 /*                                                                                                       */
 9 /* Description: This header file is used by the PVMCOMP C files : azm.c, azmcomp.c, pvmazmin.c, rng.c,   */
10 /*              rngcomp.c, pvmrngin.c                                                                    */
11 /*-------------------------------------------------------------------------------------------------------*/
12
13 /*-------------------------------------------------------------------------------------------------------*/
14 /* Include Files                                                                                         */
15 /*-------------------------------------------------------------------------------------------------------*/
16 #include <stdlib.h>
17 #include <stdio.h>
18 #include <math.h>
19 #include <string.h>
20 #include <sys/time.h>
21 #include <unistd.h>
22 #include "/home/swuyts/pvm3/include/pvm3.h"
```

```
 1
 2 /*-------------------------------------------------------------------------------------------------------*/
 3 /* Exported Macro Definitions                                                                            */
 4 /*-------------------------------------------------------------------------------------------------------*/
 5 #define CLight 3.0e8
 6 #define MAXRNG 256
 7 #define MAXAZM 512
 8 #define MAXSLAVES 64
 9 #define START OK 0
10 #define START FAILED 1
11 #define SEND OK 0
12 #define READ OK 0
13
14 /*-------------------------------------------------------------------------------------------------------*/
15 /* Structure Variable Declarations                                                                       */
16 /*-------------------------------------------------------------------------------------------------------*/
17 struct MachineType
18   (
19   char HostName[10];
20   int NumSlavesAvailable;
21   int Status;
22   );
23
24 /*-------------------------------------------------------------------------------------------------------*/
25 /* Exported Variable Declarations                                                                        */
26 /*-------------------------------------------------------------------------------------------------------*/
27 extern int RngBin;
28 extern int AzmBin;
29 extern int AzmFilt;
30 extern float SimFreq;
31 extern float PulseWidth;
32 extern float Bandwidth;
33 extern float MaxAmpl;
34 extern float CentreFreq;
35 extern float PRF;
36 extern float TargetX;
37 extern float TargetY;
38 extern float TargetZ;
39 extern float SourceZ;
40 extern float SourceV;
41 extern float RngOffset;
42 extern float TxPower;
43 extern float RCS;
44 extern float Losses;
45 extern float AntGain;
46 extern float Amplify;
47
48 /*=======================================================================================================*/
```

```
# pvm makefile for LINUX

all: pvmrc pvmac

pvmrc: rngmast rngslave presum
pvmac: azmmast azmslave cnrturn

rngmast: pvmrngin.o rng.o
        gcc -o rngmast -L/home/swuyts/pvm3/lib/LINUX pvmrngin.o rng.o -lpvm3 -lm

presum: presum.o
        gcc -o presum -L/home/swuyts/pvm3/lib/LINUX presum.o -lpvm3 -lm

rngslave: rngcomp.o fft.o
        gcc -o rngslave -L/home/swuyts/pvm3/lib/LINUX rngcomp.o fft.o -lpvm3 -lm
        mv rngslave /home/swuyts/pvm3/bin/LINUX/.

azmmast: pvmazmin.o azm.o
        gcc -o azmmast -L/home/swuyts/pvm3/lib/LINUX pvmazmin.o azm.o -lpvm3 -lm

cnrturn: cnrturn.o
        gcc -o cnrturn -L/home/swuyts/pvm3/lib/LINUX cnrturn.o -lpvm3 -lm

azmslave: azmcomp.o fft.o
        gcc -o azmslave -L/home/swuyts/pvm3/lib/LINUX azmcomp.o fft.o -lpvm3 -lm
        mv azmslave /home/swuyts/pvm3/bin/LINUX/.

clean:
        rm -f *.o

help:
        #
        # Select one of the following make commands:
        #
        # Make all      - build both processors
        # Make pvmrc    - build Range processor and slave
        # Make rngmast  - build Range processor only
        # Make pvmac    - build Azimuth processor only
        # Make azmmast  - build Azimuth processor and slave
        # Make rngslave - build Range slave only
        # Make azmmast  - build Azimuth slave only
        # Make clean    - delete old working files
        # Make help     - display this screen
        #

# ---------------- components -----------------

ampl.o: ampl.c
        gcc -c ampl.c

azm.o: azm.c
        gcc -c azm.c

azmcomp.o: azmcomp.c
        gcc -I/home/swuyts/pvm3/include -c azmcomp.c

fft.o: fft.c
        gcc -c fft.c

geom.o: geom.c
        gcc -c geom.c

pvmazmin.o: pvmazmin.c
        gcc -I/home/swuyts/pvm3/include -c pvmazmin.c

pvmrngin.o: pvmrngin.c
        gcc -I/home/swuyts/pvm3/include -c pvmrngin.c

rng.o: rng.c
        gcc -c rng.c

rngcomp.o: rngcomp.c
        gcc -I/home/swuyts/pvm3/include -c rngcomp.c
```

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                                                                                                             */
/*-------------------------------------------------------------------------------------------------------------*/
/* Author: Shirley Wuyts                                                                                       */
/* Date:  August 1997                                                                                          */
/* Project: PVMCOMP                                                                                            */
/* Task: PvmAzmIn.c                                                                                            */
/*                                                                                                             */
/* Description: This function acts as a PVM master task for performing azimuth compression on a two           */
/* dimensional image vector.                                                                                   */
/*                                                                                                             */
/* The master reads the setup file then reads the complex image data and enrolls in the PVM environment.      */
/* The master generates multiple slaves running in the PVM environment, each to perform azimuth compression   */
/* on a single range vector.                                                                                   */
/*                                                                                                             */
/* A time stamp is generated at the start of the run and again at the end and a time difference is computed    */
/* and printed.                                                                                                */
/*-------------------------------------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------------------------------------*/
/* Include Files                                                                                               */
/*-------------------------------------------------------------------------------------------------------------*/
#include "globals.h"

/*-------------------------------------------------------------------------------------------------------------*/
/* Exported Variable Definitions                                                                               */
/*-------------------------------------------------------------------------------------------------------------*/
int RngBin, AzmBin, AzmFilt;                                        /* range and azimuth dimensions */
float Bandwidth, SimFreq, MaxAmpl, PulseWidth, CentreFreq, PRF;           /* chirp pulse parameters */
float TargetX, TargetY, TargetZ, SourceZ, SourceV, RngOffset;             /* geometrical parameters */
float TxPower, RCS, Losses, AntGain, Amplify;                                   /* gain parameters */
struct timeval StartTime, EndTime;                                     /* time in secs, microsecs */
struct MachineType Machine[4];                          /* list of machines in PVM virtual machine */
int NumMachines;
int NumRuns, SlavesPerRun;                    /* number of runs needed to run SlavesNeeded slaves */
int tids[MAXSLAVES];                                                    /* slave task ids */
FILE *logfile;

int DataOffset;                                                                 /* slave ID */

float reRef[MAXRNG][MAXAZM], imRef[MAXRNG][MAXAZM];                        /* reference vars */
float reInp[MAXRNG][MAXAZM], imInp[MAXRNG][MAXAZM];                           /* input vars */
float reOut[MAXRNG][MAXAZM*2], imOut[MAXRNG][MAXAZM*2];                /* matched output vars */

/*-------------------------------------------------------------------------------------------------------------*/
/* Command Line Variable Definitions                                                                           */
/*-------------------------------------------------------------------------------------------------------------*/
char InputFile[128], OutputFile[128];

/*-------------------------------------------------------------------------------------------------------------*/
/* Module Macro Definitions                                                                                    */
/*-------------------------------------------------------------------------------------------------------------*/
#define MAXVARS 20
#define SLAVENAME "azmslave"

/*-------------------------------------------------------------------------------------------------------------*/
/* Module Variable Definitions                                                                                 */
/*-------------------------------------------------------------------------------------------------------------*/
char *vars[MAXVARS] = {"RNGBIN", "AZMBIN", "AZMFILT", "SIMFREQ",
                       "PULSEWIDTH", "BANDWIDTH", "MAXAMPL",
                       "CENTREFREQ", "PRF", "TARGETX", "TARGETY",
                       "TARGETZ", "SOURCEZ", "SOURCEV", "RNGOFFSET",
                       "TXPOWER", "RCS", "LOSSES", "ANTGAIN", "AMPLIFY"};

/*-------------------------------------------------------------------------------------------------------------*/
/* Function Definitions                                                                                        */
/*-------------------------------------------------------------------------------------------------------------*/
```



```
/*-------------------------------------------------------------------------------------------------------------*/
int GetConfigFile (void);
int GetInitFile (void);
int PVMAzm(void);
int StartSlaves(int HostNumber);
int SendData(int HostNumber, int run);
int ReadData(void);
void PvmAzmOutput( float *rePrn, float *imPrn);

```

```
1
2  /*-------------------------------------------------------------------------------------------------*/
3  /* PvmAzmIn calls the function GetInitFile for initialising the program parameters.                */
4  /* PvmAzmIn calls the function PVMAzm to generate the 2-D azimuth compression reference parameters.*/
5  /* PvmAzmIn reads the the input file, inpazm.asc.                                                  */
6  /* PvmAzmIn spawns pvmcomp slaves and sends/receives the compression data to/from them individually.  Each */
7  /* slave is given a single range vector of azimuth values to compare with the corresponding range vector of */
8  /* the reference data.                                                                             */
9  /* The time measured between spawning the slaves and finishing compression is recorded.            */
10 /*-------------------------------------------------------------------------------------------------*/
11 int main(int argc, char *argv[])
12   {
13   int i,j,run,host;
14   int masterid;                                                          /* my task id */
15
16
17   FILE *txf,*rxf;
18
19   printf("PVMAZMIN execution starting!\n");
20
21   if (argc != 3)
22     {
23     printf("Usage error:\n");
24     printf("Expecting input file and output file on command line\n");
25     return(0);
26     }
27   else
28     {
29     strncpy(InputFile,argv[1],128);
30     strncpy(OutputFile,argv[2],128);
31     printf("Command line read!\n");
32     printf("\tInput : %s\n",InputFile);
33     printf("\tOutput: %s\n",OutputFile);
34     }
35
36
37   /* READ SETUP FILE */
38   printf("READ FROM AZMMAST SETUP FILE pvmcomp.ini\n");
39   if (!GetInitFile())
40     {
41     printf("pvmcomp.ini parameters not in correct order\n");
42     return(0);
43     }
44
45   /* PVM TEST LOG FILE */
46   logfile = fopen("/home/swuyts/pvmwork/logazm.asc","wt");
47   if (logfile == NULL) { printf("Cannot open log file, logazm.asc!\n"); exit(2); }
48
49
50
51   /* READ CONFIG FILE */
52   printf("READ FROM VIRTUAL MACHINE CONFIG FILE config.ini\n");
53   if (!GetConfigFile())
54     {
55     printf("problem with Initialising File config.ini\n");
56     return(0);
57     }
58
59
60   /* GENERATE AZIMUTH REFERENCE DATA */
61   printf("GENERATE AZIMUTH REFERENCE DATA - GENERATE azm.asc FILE\n");
62   if (!PVMAzm())
63     {
64     printf("problem with PVMAzm");
65     return(0);
66     }
```



```
1
2
3   /* READ AZIMUTH REFERENCE DATA */
4   printf("READ FROM AZIMUTH REFERENCE FILE azm.asc\n");
5   txf = fopen("/home/swuyts/pvmwork/azm.asc","rt");
6   if (!txf)
7     {
8     printf("Cannot open azimuth reference file azm.asc\n");
9     return(0);
10    }
11  for ( i=0; i<RngBin ; i++ )
12    for ( j=0; j<AzmBin ; j++ )
13      if (!feof(txf))
14        fscanf(txf,"%f %f", &reRef[i][j], &imRef[i][j]);
15      else
16        {
17        printf("Not enough data in azm.asc\n");
18        fclose(txf);
19        exit(1);
20        }
21  fclose(txf);
22
23
24
25  /* READ AZMMAST INPUT FILE */
26  printf("READ FROM AZMMAST INPUT FILE %s\n",InputFile);
27  rxf = fopen(InputFile,"rt");
28  if (!rxf)
29    {
30    printf("Cannot open azmmast input file %s\n",InputFile);
31    return(0);
32    }
33  for ( i=0; i<RngBin ; i++ )
34    for ( j=0; j<AzmBin ; j++ )
35      if (!feof(rxf))
36        fscanf(rxf,"%f %f", &reInp[i][j], &imInp[i][j]);
37      else
38        {
39        printf("Not enough data in %s\n",InputFile);
40        fclose(rxf);
41        exit(1);
42        }
43  fclose(rxf);
44
45
```

```
1
2   /* OBTAIN START TIME */
3   gettimeofday(&StartTime,NULL);
4
5
6   /* ENROLL MASTER IN PVM */
7   printf("ENROLL MASTER IN PVM\t");
8   masterid = pvm_mytid();
9   printf("masterid = %d\n\n",masterid);
10
11
12  /* LOOP OF NumRuns SLAVES. */
13  DataOffset = 0;
14  for (run=0; run<NumRuns; run++)
15    {
16    printf("==========================================\n");
17    printf("STARTING RUN %d OF %d\n",run+1,NumRuns);
18
19    /* .PROCESS EACH HOST */
20    for (host=0; host<NumMachines; host++)
21      {
22      /* ..START SLAVES ON EACH HOST */
23      printf("  %s starting %d slaves",Machine[host].HostName,
24        Machine[host].NumSlavesAvailable);
25      StartSlaves(host);
26
27      /* ..SEND DATA TO EACH HOST */
28      printf("  sending data to %s slaves",Machine[host].HostName);
29      SendData(host,run);
30
31      } /* NEXT HOST */
32
33    /* ..RECEIVE DATA FROM SLAVES */
34    printf("\n  reading data from all slaves");
35    ReadData();
36
37    } /* NEXT SET OF NumRuns SLAVES*/
38
39  printf("\n==========================================\n");
40
41  /* OBTAIN END TIME */
42  gettimeofday(&EndTime,NULL);
43
44
45  /* PRINT TIMING RESULTS */
46  printf("\nTotal Run time : %d secs\n",EndTime.tv_sec - StartTime.tv_sec);
47
48  fprintf(logfile,"Azimuth Compression Run time : %d secs\n",EndTime.tv_sec - StartTime.tv_sec);
49  fclose(logfile);
50
51  /* SAVE AZMMAST OUTPUT FILE */
52  printf("SAVE TO AZMMAST OUTPUT FILE %s\n",OutputFile);
53  PvmAzmOutput ((float *)reOut,(float *)imOut);
54
55
56  return (1);
57  }
58
59
```



```
1
2   /*-------------------------------------------------------------------------*/
3   /* StartSlaves starts a set of slaves on a remote machine and if not       */
4   /* successful will automatically clean up the remote machine.  The parameter passed is */
5   /* offset in the Machine array that describes the host                     */
6   /*-------------------------------------------------------------------------*/
7   int StartSlaves(int HostNumber)
8     {
9     int j,numt;
10
11    /* .SPAWNING OF SLAVES */
12    numt=pvm_spawn(SLAVENAME, (char**)0, PvmTaskHost,
13      Machine[HostNumber].HostName, Machine[HostNumber].NumSlavesAvailable,
14      &tids);
15
16    if (numt < Machine[HostNumber].NumSlavesAvailable)          /* if spawning problems */
17      {
18      printf(", FAILED!\n");
19      printf("** Trouble spawning slaves on %s. Ignoring host. Error codes are:\n",
20        Machine[HostNumber].HostName);
21
22      for (j=numt ; j<Machine[HostNumber].NumSlavesAvailable ; j++)
23        printf("TID %d %d\n",j,tids[j]);
24
25      for (j=0 ; j<numt ; j++)
26        pvm_kill(tids[j]);                                      /* kill healthy slaves */
27
28      return(START_FAILED);
29      }
30
31    printf(", successful\n");
32    return START_OK;
33  }
34
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* SendData sends the data required by each slave                          */
4 /*-------------------------------------------------------------------------*/
5 int SendData(int HostNumber, int run)
6   {
7     int j;
8
9     /* START EACH SLAVE */
10    for (j=0 ; j<Machine[HostNumber].NumSlavesAvailable ; j++)
11      {
12      pvm_initsend(PvmDataDefault);                          /* set up send buffer */
13      pvm_pkint(&DataOffset, 1, 1);                          /* slave ID */
14      pvm_pkfloat(reRef[DataOffset], AzmBin, 1);             /* real reference vector */
15      pvm_pkfloat(imRef[DataOffset], AzmBin, 1);             /* imag reference vector */
16      pvm_pkfloat(reInp[DataOffset], AzmBin, 1);             /* real input vector */
17      pvm_pkfloat(imInp[DataOffset], AzmBin, 1);             /* imag input vector */
18      pvm_send(tids[j], 0);                                  /* send actual msg to slave j */
19      DataOffset++;                                          /* increment slave ID */
20      }
21
22    printf(", successful\n");
23    return SEND_OK;
24  }
25
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* ReadData read the data back from each slave                             */
4 /*-------------------------------------------------------------------------*/
5 int ReadData(void)
6   {
7     int j;
8     int ReturnDataOffset;                                  /* returning slave ID */
9
10    for (j=0; j<SlavesPerRun; j++)
11      {
12      pvm_recv( -1, 5 );                                   /* receive msg type 5 */
13      pvm_upkint(&ReturnDataOffset, 1, 1);                 /*  receive slave ID */
14      pvm_upkfloat(reOut[ReturnDataOffset], (AzmBin*2), 1); /* real result vector */
15      pvm_upkfloat(imOut[ReturnDataOffset], (AzmBin*2), 1); /* imag result vector */
16      }
17
18    printf(", successful\n");
19    return READ_OK;
20  }
21
```

```
1
2  /*-------------------------------------------------------------------------*/
3  /* GetInitFile reads the setup file pvmcomp.ini for initialising the Range Compression parameters.          */
4  /*-------------------------------------------------------------------------*/
5  int GetInitFile (void)
6    {
7    char inpfile[]="/home/swuyts/pvmwork/pvmComp.ini";
8    FILE *Ifile;
9
10   char name[20];
11   int i = 0;
12
13
14   /* READ INITIALISATION FILE */
15   Ifile = fopen(inpfile,"rt");
16   if (!Ifile)
17     {
18     printf("Cannot open %s\n", inpfile);
19     return(0);
20     }
21
22
23   fscanf(Ifile,"%s %d", name, &RngBin);
24   if ((strcmp(vars[i++], name))!=0)
25     {fclose(Ifile); return 0;}
26
27   fscanf(Ifile,"%s %d", name, &AzmBin);
28   if ((strcmp(vars[i++], name))!=0)
29     {fclose(Ifile); return 0;}
30
31   fscanf(Ifile,"%s %d", name, &AzmFilt);
32   if ((strcmp(vars[i++], name))!=0)
33     {fclose(Ifile); return 0;}
34
35   fscanf(Ifile,"%s %f", name, &SimFreq);
36   if ((strcmp(vars[i++], name))!=0)
37     {fclose(Ifile); return 0;}
38
39   fscanf(Ifile,"%s %f", name, &PulseWidth);
40   if ((strcmp(vars[i++], name))!=0)
41     {fclose(Ifile); return 0;}
42
43   fscanf(Ifile,"%s %f", name, &Bandwidth);
44   if ((strcmp(vars[i++], name))!=0)
45     {fclose(Ifile); return 0;}
46
47   fscanf(Ifile,"%s %f", name, &MaxAmpl);
48   if ((strcmp(vars[i++], name))!=0)
49     {fclose(Ifile); return 0;}
50
51   fscanf(Ifile,"%s %f", name, &CentreFreq);
52   if ((strcmp(vars[i++], name))!=0)
53     {fclose(Ifile); return 0;}
54
55   fscanf(Ifile,"%s %f", name, &PRF);
56   if ((strcmp(vars[i++], name))!=0)
57     {fclose(Ifile); return 0;}
58
59   fscanf(Ifile,"%s %f", name, &TargetX);
60   if ((strcmp(vars[i++], name))!=0)
61     {fclose(Ifile); return 0;}
62
63   fscanf(Ifile,"%s %f", name, &TargetY);
64   if ((strcmp(vars[i++], name))!=0)
65     {fclose(Ifile); return 0;}
66
```

```
1    fscanf(Ifile,"%s %f", name, &TargetZ);
2    if ((strcmp(vars[i++], name))!=0)
3      {fclose(Ifile); return 0;}
4
5    fscanf(Ifile,"%s %f", name, &SourceZ);
6    if ((strcmp(vars[i++], name))!=0)
7      {fclose(Ifile); return 0;}
8
9    fscanf(Ifile,"%s %f", name, &SourceV);
10   if ((strcmp(vars[i++], name))!=0)
11     {fclose(Ifile); return 0;}
12
13   fscanf(Ifile,"%s %f", name, &RngOffset);
14   if ((strcmp(vars[i++], name))!=0)
15     {fclose(Ifile); return 0;}
16
17   fscanf(Ifile,"%s %f", name, &TxPower);
18   if ((strcmp(vars[i++], name))!=0)
19     {fclose(Ifile); return 0;}
20
21   fscanf(Ifile,"%s %f", name, &RCS);
22   if ((strcmp(vars[i++], name))!=0)
23     {fclose(Ifile); return 0;}
24
25   fscanf(Ifile,"%s %f", name, &Losses);
26   if ((strcmp(vars[i++], name))!=0)
27     {fclose(Ifile); return 0;}
28
29   fscanf(Ifile,"%s %f", name, &AntGain);
30   if ((strcmp(vars[i++], name))!=0)
31     {fclose(Ifile); return 0;}
32
33   fscanf(Ifile,"%s %f", name, &Amplify);
34   if ((strcmp(vars[i++], name))!=0)
35     {fclose(Ifile); return 0;}
36
37   fclose(Ifile);
38   return(1);
39   }
40
41
```

```
 1
 2 TEST 2:  COMBINATION 1
 3
 4 ============= Virtual Machine Description ==============
 5         Host    Slaves
 6 ========================================================
 7         pentium 32      OK
 8         alpha   16      OK
 9         hppal   16      OK
10 ========================================================
11 Runs : 8
12 --------------------------------------------------------
13
14 Range Compression Run time : 49 secs
15
16 [tB0040000] pentium LINUX 3.3.11
17 [tB0040000] ready Wed Sep 17 20:21:07 1997
18 [tB00c0000] hppal HPPA 3.3.11
19 [tB00c0000] ready Wed Sep 17 20:11:47 1997
20 [tB0080000] alpha ALPHA 3.3.11
21 [tB0080000] ready Wed Sep 17 20:19:54 1997
```

```
 1
 2 TEST 2:  COMBINATION 2
 3
 4 ============= Virtual Machine Description ==============
 5         Host    Slaves
 6 ========================================================
 7         pentium 16      OK
 8         alpha   32      OK
 9         hppal   16      OK
10 ========================================================
11 Runs : 8
12 --------------------------------------------------------
13
14 Range Compression Run time : 50 secs
15
16 [tB0040000] pentium LINUX 3.3.11
17 [tB0040000] ready Wed Sep 17 20:21:07 1997
18 [tB00c0000] hppal HPPA 3.3.11
19 [tB00c0000] ready Wed Sep 17 20:11:47 1997
20 [tB0080000] alpha ALPHA 3.3.11
21 [tB0080000] ready Wed Sep 17 20:19:54 1997
```

```
 1
 2 TEST 2:  COMBINATION 3
 3
 4 ============= Virtual Machine Description ==============
 5         Host    Slaves
 6 ========================================================
 7         pentium 44      OK
 8         alpha   16      OK
 9         hppal   4       OK
10 ========================================================
11 Runs : 8
12 ========================================================
13
14 Range Compression Run time : 51 secs
15
16 [t80040000] pentium LINUX 3.3.11
17 [t80040000] ready Wed Sep 17 20:21:07 1997
18 [t800c0000] hppal HPPA 3.3.11
19 [t800c0000] ready Wed Sep 17 20:11:47 1997
20 [t80080000] alpha ALPHA 3.3.11
21 [t80080000] ready Wed Sep 17 20:19:54 1997
```

```
 1
 2 TEST 3:  SLAVES PERFORM COMPRESSION 5 TIMES
 3
 4 ============= Virtual Machine Description ==============
 5         Host    Slaves
 6 ========================================================
 7         pentium 16      OK
 8 --------------------------------------------------------
 9 Runs : 32
10 --------------------------------------------------------
11
12 Range Compression Run time : 93 secs
13
14 [t80040000] pentium LINUX 3.3.11
15 [t80040000] ready Wed Sep 17 21:21:01 1997
16
17
18
19 ============= Virtual Machine Description ==============
20         Host    Slaves
21 ========================================================
22       * pentium 32      OK
23         alpha   16      OK
24         hppal   16      OK
25 ========================================================
26 Runs : 8
27 ========================================================
28
29 Range Compression Run time : 90 secs
30
31 [t80040000] pentium LINUX 3.3.11
32 [t80040000] ready Wed Sep 17 21:25:54 1997
33 [t80080000] alpha ALPHA 3.3.11
34 [t80080000] ready Wed Sep 17 21:24:40 1997
35 [t800c0000] hppal HPPA 3.3.11
36 [t800c0000] ready Wed Sep 17 21:16:34 1997
```

```
 1
 2 TEST 3:  SLAVES PERFORM COMPRESSION 10 TIMES
 3
 4 ************* Virtual Machine Description *************
 5         Host    Slaves
 6 ********************************************************
 7         pentium 16      OK
 8 ********************************************************
 9 Runs : 32
10 ********************************************************
11
12 Range Compression Run time : 149 secs
13
14 [tB0040000] pentium LINUX 3.3.11
15 [tB0040000] ready Wed Sep 17 21:36:31 1997
16
17
18
19 ************* Virtual Machine Description *************
20         Host    Slaves
21 ********************************************************
22         pentium 32      OK
23         alpha   16      OK
24         hppa1   16      OK
25 ********************************************************
26 Runs : 8
27 ********************************************************
28
29 Range Compression Run time : 143 secs
30
31 [tB0040000] pentium LINUX 3.3.11
32 [tB0040000] ready Wed Sep 17 21:25:54 1997
33 [tB00B0000] alpha ALPHA 3.3.11
34 [tB00B0000] ready Wed Sep 17 21:24:40 1997
35 [tB00c0000] hppa1 HPPA 3.3.11
36 [tB00c0000] ready Wed Sep 17 21:16:34 1997
```

```
 1
 2 TEST 4:  MASTER SENDS DATA 5 TIMES
 3
 4 ************* Virtual Machine Description *************
 5         Host    Slaves
 6 ********************************************************
 7         pentium 16      OK
 8 ********************************************************
 9 Runs : 32
10 ********************************************************
11
12 Range Compression Run time : 71 secs
13
14 [tB0040000] pentium LINUX 3.3.11
15 [tB0040000] ready Wed Sep 17 20:41:46 1997
16
17
18
19 ************* Virtual Machine Description *************
20         Host    Slaves
21 ********************************************************
22         pentium 32      OK
23         alpha   16      OK
24         hppa1   16      OK
25 ********************************************************
26 Runs : 8
27 ********************************************************
28
29 Range Compression Run time : 68 secs
30
31 [tB0040000] pentium LINUX 3.3.11
32 [tB0040000] ready Wed Sep 17 21:13:47 1997
33 [tB00B0000] alpha ALPHA 3.3.11
34 [tB00B0000] ready Wed Sep 17 21:12:34 1997
35 [tB00c0000] hppa1 HPPA 3.3.11
36 [tB00c0000] ready Wed Sep 17 21:04:27 1997
```

```
1
2 TEST 4:  MASTER SENDS DATA 10 TIMES
3
4 ============ Virtual Machine Description ==============
5         Host    Slaves
6 ------------------------------------------------------
7         pentium 16      OK
8 ======================================================
9 Runs : 32
10 ------------------------------------------------------
11
12 Range Compression Run time : 94 secs
13
14 [tB0040000] pentium LINUX 3.3.11
15 [tB0040000] ready Wed Sep 17 21:57:32 1997
16
17
18
19 ============ Virtual Machine Description ==============
20         Host    Slaves
21 ------------------------------------------------------
22         pentium 32      OK
23         alpha   16      OK
24         hppa1   16      OK
25 ======================================================
26 Runs : 8
27 ======================================================
28
29 Range Compression Run time : 94 secs
30
31 [tB0040000] pentium LINUX 3.3.11
32 [tB0040000] ready Wed Sep 17 22:04:05 1997
33 [tB00B0000] alpha ALPHA 3.3.11
34 [tB00B0000] ready Wed Sep 17 22:02:51 1997
35 [tB00c0000] hppa1 HPPA 3.3.11
36 [tB00c0000] ready Wed Sep 17 21:54:44 1997
```

```
1
2 TEST 5:  MESSAGE IS 5 TIMES BIGGER
3
4 ============ Virtual Machine Description ==============
5         Host    Slaves
6 ------------------------------------------------------
7         pentium 16      OK
8 ======================================================
9 Runs : 32
10 ------------------------------------------------------
11
12 Range Compression Run time : 67 secs
13
14 [tB0040000] pentium LINUX 3.3.11
15 [tB0040000] ready Wed Sep 17 22:45:42 1997
16
17
18
19 ============ Virtual Machine Description ==============
20         Host    Slaves
21 ------------------------------------------------------
22         pentium 32      OK
23         alpha   16      OK
24         hppa1   16      OK
25 ======================================================
26 Runs : 8
27 ======================================================
28
29 Range Compression Run time : 63 secs
30
31 [tB0040000] pentium LINUX 3.3.11
32 [tB0040000] ready Wed Sep 17 22:42:49 1997
33 [tB00B0000] alpha ALPHA 3.3.11
34 [tB00B0000] ready Wed Sep 17 22:33:16 1997
35 [tB00c0000] hppa1 HPPA 3.3.11
36 [tB00c0000] ready Wed Sep 17 22:33:28 1997
```

```
 1
 2 TEST 6:  TIMING IS MEASURED OVER ENTIRE PROCESS
 3
 4 ============= Virtual Machine Description ==============
 5         Host    Slaves
 6 ------------------------------------------------------------
 7         pentium 16      OK
 8 ------------------------------------------------------------
 9 Runs : 32
10 ============================================================
11
12 Range Compression Run time : 81 secs
13
14 [tB0040000] pentium LINUX 3.3.11
15 [tB0040000] ready Wed Sep 17 20:41:46 1997
16
17
18
19 ============= Virtual Machine Description ==============
20         Host    Slaves
21 ------------------------------------------------------------
22         pentium 16      OK
23 ============================================================
24 Runs : 16
25 ============================================================
26
27 Azimuth Compression Run time : 142 secs
28 [tB0040000] pentium LINUX 3.3.11
29 [tB0040000] ready Thu Sep 18 16:52:30 1997
```

11. APPENDIX E: MATCHING THE CHIRP PULSE

11.1 The Theory

The transmitted Linear FM waveform consists of rectangular pulses of constant amplitude A and of pulse width T.

The pulses of carrier frequency f_0 are modulated with a frequency increasing linearly from f_1 to f_2 over the duration of the pulse. i.e. frequency f is $f = f_0 - \frac{B \cdot t}{2}$ where bandwidth $B = f_2 - f_1$ and angular frequency ω_0 is $\omega_0 = \left(\omega_0 - \frac{\mu \cdot t}{2}\right)$ where $\mu = 2 \cdot \pi \cdot \frac{B}{T}$

Letting amplitude A = unity, the waveform f(t) becomes $f(t) = \cos\left(\omega_0 \cdot t - \frac{\mu \cdot t^2}{2}\right)$ for $-\frac{T}{2} \le t \le \frac{T}{2}$

According to matched filter theory, the impulse response h(t) of the matched filter is the time-reversed complex conjugate of f(t), namely: $h(t) = \sqrt{\frac{2 \cdot \mu}{\pi}} \cdot \cos\left(\omega_0 \cdot t - \frac{\mu \cdot t^2}{2}\right)$ where the $\sqrt{\frac{2 \cdot \mu}{\pi}}$ is required for unity gain.

The return signal $f(t_0)$ from the target at time t_0 will have the characteristics of the transmitted signal except that it will be shifted in frequency proportional to the Doppler shift, ω_d as follows

$$f(t_0) := \cos\left[\left(\omega_0 + \omega_d\right)\cdot t_0 + \frac{\mu\cdot(t_0)^2}{2}\right]$$

The convolution of the return signal with the matched filter is given by

$$g(t) := \int_{-\infty}^{\infty} f(\tau)\cdot h(t-\tau)\, d\tau$$

$$g(t_0,\omega_d) := \int_{-\frac{T}{2}}^{\frac{T}{2}} \cos\left[\left(\omega_0 + \omega_d\right)\cdot\tau + \frac{\mu\cdot(\tau)^2}{2}\right]\cdot\left(\sqrt{\frac{2\cdot\mu}{\pi}}\right)\cdot\cos\left[\omega_0\cdot(t_0-\tau) - \frac{1}{2}\cdot\mu\cdot(t_0-\tau)^2\right] d\tau$$

$$g(t_0,\omega_d) := \sqrt{\frac{\mu}{2\cdot\pi}}\cdot\cos\left[\left(\omega_0 + \frac{\omega_d}{2}\right)\cdot t_0\right]\cdot\left(T - |t_0|\right)\cdot\frac{\sin\left[\frac{\omega_d + \mu\cdot t_0}{2}\cdot\left(T - |t_0|\right)\right]}{\left(\frac{\omega_d + \mu\cdot t_0}{2}\right)\cdot\left(T - |t_0|\right)}$$

For cases of no Doppler shift, $\omega_d := 0$

$g(t_0,0)$ becomes

$$g(t_0) = \frac{\mu}{2\cdot\pi}\cdot\cos(\omega_0\cdot t_0)\cdot(T - t_0)\cdot\frac{\sin\left(\frac{\mu\cdot t_0}{2}\cdot(T - t_0)\right)}{\left(\frac{\mu\cdot t_0}{2}\cdot(T - t_0)\right)}$$

Thus

$$g(t_0) = \Phi(t_0)\cdot(T - t_0)\cdot\frac{\sin\left(\frac{\mu\cdot t_0}{2}\cdot(T - t_0)\right)}{\left(\frac{\mu\cdot t_0}{2}\cdot(T - t_0)\right)}$$

Here Φ is the carrier harmonic which is modulated by a triangle pulse and a sinc of quadratic time.

The autocorrelation function is illustrated in Figure 55, showing the shape of the envelope and the compression of the chirp into a narrow spike.

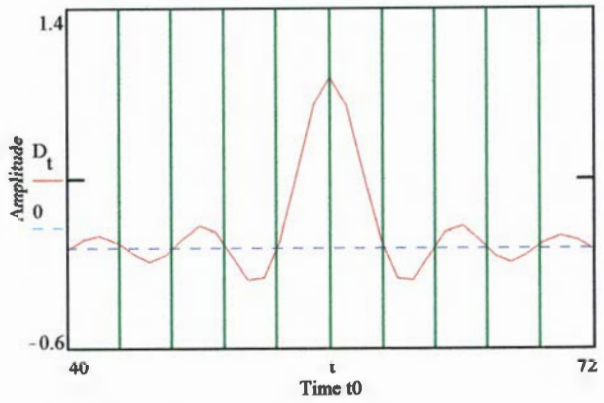


Figure 55: Chirp Autocorrelation Function

The first root of the sinc occurs when the argument of the sin is π , which is approximately at $t_0 := \frac{1}{B}$

Thus, the main lobe width is $\frac{2}{B}$ and the half power width is approximately $\frac{1}{B}$.

The gain in resolution, called the *pulse compression ratio*, is $\frac{T}{t_0}$ or $T \cdot B$ also known as the *time-bandwidth product* of the chirp signal.

Thus, for high resolution, we want a high time-bandwidth signal. Such signals are also called *sophisticated* because of their ability to carry more information. High time-bandwidth implies a large bandwidth B, since the pulse duration T is limited by a number of constraints, particularly the need to receive over a large interval in order to image a wide swath on the surface.

This is because the single antenna cannot transmit and receive at the same time, so transmit time must be a small fraction of the inter-pulse period. The bandwidth theorem puts a theoretical lower bound of 1 on the time-bandwidth product of any signal (the minimum being realised only by a Gaussian pulse).

Amplitude of the autocorrelation pulse is given relative to a maximum value $\sqrt{D}$ where D is defined as the *dispersion factor* and is equal to T.B, the time-bandwidth product.

In current radars using pulse compression techniques , dispersion factors of thousands can be achieved.

11.2 Windowing

The resolving capability of the system is determined by the narrowness of the main lobe and its separation from, and amplitude ratio to, the sidelobes.

After compression, the $\frac{\sin(x)}{x}$ pulse shape has the closest sidelobes at -13.2dB **[19]** down from the peak.

The large sidelobes are often objectionable since a large target might mask nearby, smaller targets. Also near-in sidelobes might at times be mistaken for separate targets. To compensate for this sidelobe ambiguity, it is useful to weight (or window) the returned signal over the integration time.

Although not used in the thesis software, the windowing can equivalently be done in the frequency domain where it can be efficiently incorporated into the reference function. This suppression occurs at the expense of reduced signal-to-noise ratio at the compression filter output and a small increase in pulse width.

The most widely used is a window function of the *cosine on a pedestal* form **[17]**,

$$W(f) = \alpha - \beta \cdot \cos\left(\frac{2 \cdot \pi \cdot f}{B}\right) \quad \text{where } \alpha - \beta = 1$$

The *Hamming Window* is popular for which $\alpha = 0.54$ and has been chosen for use in the thesis software.

BIBLIOGRAPHY

[1] *Alaska SAR Facility's Homepage*, v0guide@www.asf.alaska.edu

[2] Dobbe, J.G.G, *Dr Dobb's Journal: Faster FFTs*, #227, February 1995

[3] Dongarra, J.J, Geist, G.A, Manchek, R.J, Papadopoulos, P.M, *Adding Context and Static Groups into PVM*, Oak Ridge National Laboratory, 1995

[4] Dongarra, J.J, Geist, G.A, Manchek, R.J, Sunderam, V.S, *Integrated PVM Framework Supports Heterogeneous Network Computing*, Oak Ridge National Laboratory, 1993

[5] Elachi, C, *Spaceborne Radar Remote Sensing: Applications and Techniques*, IEEE Press, 1988

[6] Fischer, M, Dongarra, J, *Another Architecture: PVM on Windows 95/NT*, Oak Ridge National Laboratory, 1996

[7] Geist, A and others, *PVM 3 User's Guide and Reference Manual*, Oak Ridge National Laboratory, 1994

[8] Golda, P, *Radar Simulator (RADSIM),* version 4, Radar Laboratory, Electrical Engineering Dept, UCT, 1995

[9] Heng, A.W, *Short Notes on Chirp Scaling Algorithm for Processing Spaceborne SAR Data*, National University of Singapore, 1994

[10] Heng, A.W, *Synthetic Aperture Radar: A Simplified Review*, National University of Singapore, 1994

[11] Hovanessian, S.A, *Radar System Design and Analysis*, Artech House, 1984

[12] Kailasanathan, T, *Parallel Virtual Machine (PVM)*, Radar Laboratory, Electrical Engineering Dept, UCT, 1995

[13] Kovaly, J.J, *Synthetic Aperture Radar*, Artech, 1976

[14] Koloi, M.S, *Synthetic Aperture Sonar System*, Radar Laboratory, Electrical Engineering Dept, UCT, 1995

[15] Lengenfelder, R, *Radar Simulator (SARSIM)*, version 060597, Radar Laboratory, Electrical Engineering Dept, UCT, 1997

[16] Morris, G and Harkness, L, *Airborne Pulse Doppler Radar*, Artech House, 1996

[17] Olmsted, C, *Scientific SAR User's Guide*, Alaska SAR Facility, 1993

[18] *The PVM Project*, http://www.epm.ornl.gov/pvm/pvm_home.html

[19] Skolnik, M.I, *Introduction to Radar Systems*, McGraw-Hill, 1980

[20] Stimson, G.W, *Introduction to Airborne Radar*, Hughes Aircraft Company, 1983

[21] Sunderam, V.S, Geist, G.A, Dongarra, J and Manchek, R. , *The PVM Concurrent Computing System: Evolution, Experiences, and Trends*, Oak Ridge National Laboratory, 1993

```
1
2  /*-------------- ------------------------------------------------------------------------------*/
3  /* GetConfigFile reads the virtual machine file config.ini for initialising the PVM machine parameters.     */
4  /*-------------------- ---------- ------------------------- ---------------------- ----------------*/
5  int GetConfigFile (void)
6    {
7    char inpfile[]="/home/swuyts/pvmwork/config.ini";
8    FILE *Ifile;
9    int i;
10
11   /* READ CONFIG FILE */
12   Ifile = fopen(inpfile,"rt");
13   if (!Ifile)
14     {
15     printf("Cannot open %s\n", inpfile);
16     return(0);
17     }
18
19   fprintf(logfile,"============= Virtual Machine Description =============\n");
20   fprintf(logfile,"\tHost\tSlaves\n");
21   fprintf(logfile,"=======================================================\n");
22
23   i = 0;
24   SlavesPerRun = 0;
25
26   while (!feof(Ifile))
27     {
28     fscanf(Ifile,"%s %d",Machine[i].HostName, &Machine[i].NumSlavesAvailable);
29     SlavesPerRun += Machine[i].NumSlavesAvailable;
30     if (Machine[i].NumSlavesAvailable != 0)
31       {
32       fprintf(logfile,"\t%s\t%d\tOK\n",Machine[i].HostName,Machine[i].NumSlavesAvailable);
33       i++;
34       }
35     }
36   NumMachines = i;
37
38   fclose(Ifile);
39
40
41   /* CHECK THAT TOTAL SLAVES < RngBin */
42   if (SlavesPerRun > RngBin)
43     {
44     printf ("The number of total slaves available in config.ini must be less than RNGBIN");
45     return(0);
46     }
47
48
49   /* CHECK THAT TOTAL SLAVES = POWER OF 2 */
50   if (fmod((float)SlavesPerRun,2.0))
51     {
52     printf ("The number of total slaves available in config.ini must be a power of 2");
53     return(0);
54     }
55
56
57   /* DETERMINE THE NUMBER RUNS NEEDED */
58   NumRuns = RngBin / SlavesPerRun;
59
60   /* CHECK THAT TOTAL SLAVES IS A FACTOR OF RNGBIN */
61   if (fmod((float)RngBin, (float)SlavesPerRun))
62     {
63     printf ("The number of total slaves available in config.ini must be\n");
64     printf ("a factor of RngBin\n");
65     return(0);
66     }
```



```
1
2    fprintf(logfile,"=======================================================\n");
3    fprintf(logfile,"Runs : %d\n",NumRuns);
4    fprintf(logfile,"-------------------------------------------------------\n\n");
5
6
7    return(1);
8    }
9
10
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* PvmAzmOutput prints the Azimuth Compression output into the file azmComp.asc */
4 /*-------------------------------------------------------------------------*/
5
6 void PvmAzmOutput( float *rePrn, float *imPrn)
7   {
8   int i,j;
9   FILE *outf;
10
11   outf = fopen(OutputFile,"wt");
12   if (!outf)
13     {
14     printf("Cannot open azmmast output file %s\n",OutputFile);
15     return;
16     }
17
18   for ( i=0; i<RngBin ; i++ )
19     for ( j=0; j<(AzmBin*2) ; j++ )
20       fprintf(outf,"%12.6f %12.6f\n",
21         *(rePrn + i*MAXAZM*2 + j), *(imPrn + i*MAXAZM*2 + j));
22
23   fclose(outf);
24   }
25
26 /*=========================================================================*/
```

```
1 RNGBIN 256
2 AZMBIN 512
3 AZMFILT 140
4 SIMFREQ 0.15
5 PULSEWIDTH 700.0
6 BANDWIDTH 0.1
7 MAXAMPL 10.0
8 CENTREFREQ 0.141
9 PRF 180.0
10 TARGETX 0.0
11 TARGETY 20000.0
12 TARGETZ 0.0
13 SOURCEZ 10000.0
14 SOURCEV 246.0
15 RNGOFFSET 22302.0
16 TXPOWER 1.0
17 RCS 1.0
18 LOSSES 0.0
19 ANTGAIN 7.0
20 AMPLIFY 10000000.0
```

```
/*-------------------------------------------------------------------------------------------------*/
/*                                                                                                 */
/*-------------------------------------------------------------------------------------------------*/
/* Author: Shirley Wuyts                                                                           */
/* Date:  August 1997                                                                              */
/* Project: PVMCOMP                                                                                */
/* Task: PvmRngIn.c                                                                                */
/*                                                                                                 */
/* Description: This function acts as a PVM master task for performing range compression on a two  */
/* dimensional image vector.                                                                       */
/*                                                                                                 */
/* The master reads the setup file then reads the complex image data and enrolls in the PVM environment.  */
/* The master generates multiple slaves running in the PVM environment, each to perform range compression */
/* on a single azimuth vector.                                                                     */
/*                                                                                                 */
/* A time stamp is generated at the start of the run and again at the end and a time difference is computed */
/* and printed.                                                                                    */
/*-------------------------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------------------------*/
/* Include Files                                                                                   */
/*-------------------------------------------------------------------------------------------------*/
#include "globals.h"

/*-------------------------------------------------------------------------------------------------*/
/* Exported Variable Definitions                                                                   */
/*-------------------------------------------------------------------------------------------------*/
int RngBin, AzmBin, AzmFilt;                                    /* range and azimuth dimensions */
float Bandwidth, SimFreq, MaxAmpl, PulseWidth;                        /* chirp pulse parameters */
struct timeval StartTime, EndTime;                                  /* time in secs, microsecs */
struct MachineType Machine[4];                          /* list of machines in PVM virtual machine */
int NumMachines;
int NumRuns, SlavesPerRun;                      /* number of runs needed to run SlavesNeeded slaves */
int tids[MAXSLAVES];                                                    /* slave task ids */
FILE *logfile;

int DataOffset;                                                                  /* slave ID */

float reRef[MAXRNG], imRef[MAXRNG];                                        /* reference vars */
float reInp[MAXAZM][MAXRNG], imInp[MAXAZM][MAXRNG];                           /* input vars */
float reOut[MAXAZM][MAXRNG], imOut[MAXAZM][MAXRNG];                  /* matched output vars */

/*-------------------------------------------------------------------------------------------------*/
/* Command Line Variable Definitions                                                               */
/*-------------------------------------------------------------------------------------------------*/
char InputFile[128], OutputFile[128];

/*-------------------------------------------------------------------------------------------------*/
/* Module Macro Definitions                                                                        */
/*-------------------------------------------------------------------------------------------------*/
#define MAXVARS 7
#define SLAVENAME "rngslave"

/*-------------------------------------------------------------------------------------------------*/
/* Module Variable Definitions                                                                     */
/*-------------------------------------------------------------------------------------------------*/
char *vars[MAXVARS] = {"RNGBIN", "AZMBIN", "AZMFILT", "SIMFREQ",
                       "PULSEWIDTH", "BANDWIDTH", "MAXAMPL"};

/*-------------------------------------------------------------------------------------------------*/
/* Function Definitions                                                                            */
/*-------------------------------------------------------------------------------------------------*/
int GetInitFile (void);
int GetConfigFile (void);
int PVMRng (void);
int StartSlaves(int HostNumber);
```



```
int SendData(int HostNumber, int run);
int ReadData(void);
void PvmRngOutput (float *rePrn, float *imPrn);

```

```
1
2  /*-------------------------------------------------------------------------------------*/
3  /* PvmRngIn calls the function GetInitFile for initialising the program parameters.    */
4  /* PvmRngIn calls the function PVMRng to generate the range compression reference parameters.  */
5  /* PvmRngIn reads the the input file, inprng.asc.                                      */
6  /* PvmRngIn spawns pvmcomp slaves and sends/receives the compression data to/from them individually.  Each  */
7  /* slave is given a single azimuth vector of range returns to compare with the reference data.  */
8  /* The time measured between spawning the slaves and finishing compression is recorded.  */
9  /*-------------------------------------------------------------------------------------*/
10 int main(int argc, char *argv[])
11   {
12   int i,j,run,host;
13   int masterid;                                                          /* my task id */
14
15   FILE *txf,*rxf;
16
17   printf("PVMRNGIN execution starting!\n");
18
19   if (argc != 3)
20     {
21     printf("Usage error:\n");
22     printf("Expecting input file and output file on command line\n");
23     return(0);
24     }
25   else
26     {
27     strncpy(InputFile,argv[1],128);
28     strncpy(OutputFile,argv[2],128);
29     printf("Command line read!\n");
30     printf("\tInput : %s\n",InputFile);
31     printf("\tOutput: %s\n",OutputFile);
32     }
33
34
35   /* READ SETUP FILE */
36   printf("READ FROM RNGMAST SETUP FILE pvmcomp.ini\n");
37   if (!GetInitFile())
38     {
39     printf("problem with Initialising File pvmcomp.ini\n");
40     return(0);
41     }
42
43   /* PVM TEST LOG FILE */
44   logfile = fopen("/home/swuyts/pvmwork/logrng.asc","wt");
45   if (logfile == NULL) { printf("Cannot open log file, logrng.asc!\n"); exit(2); }
46
47
48
49   /* READ CONFIG FILE */
50   printf("READ FROM VIRTUAL MACHINE CONFIG FILE config.ini\n");
51   if (!GetConfigFile())
52     {
53     printf("problem with Initialising File config.ini\n");
54     return(0);
55     }
56
57
58   /* GENERATE RANGE REFERENCE DATA */
59   printf("GENERATE RANGE REFERENCE DATA - GENERATE rng.asc FILE\n");
60   if (!PVMRng())
61     {
62     printf("problem with PVMRng\n");
63     return(0);
64     }
65
66
```



```
1   /* READ RANGE REFERENCE DATA */
2   printf("READ FROM RANGE REFERENCE FILE rng.asc\n");
3   txf = fopen("/home/swuyts/pvmwork/rng.asc","rt");
4   if (!txf)
5     {
6     printf("Cannot open range reference file rng.asc\n");
7     return(0);
8     }
9   for ( i=0 ; i<RngBin; i++)
10    if (!feof(txf))
11      fscanf(txf,"%f %f", &reRef[i], &imRef[i]);
12    else
13      {
14      printf("Not enough data in rng.asc\n");
15      fclose(txf);
16      exit(1);
17      }
18  fclose(txf);
19
20
21  /* READ RNGMAST INPUT FILE */
22  printf("READ FROM RNGMAST INPUT FILE %s\n",InputFile);
23  rxf = fopen(InputFile,"rt");
24  if (!rxf)
25    {
26    printf("Cannot open rngmast input file %s\n",InputFile);
27    return(0);
28    }
29  for ( j=0; j<AzmBin; j++ )
30    for ( i=0; i<RngBin; i++ )
31      if (!feof(rxf))
32        fscanf(rxf,"%f %f,", &reInp[j][i], &imInp[j][i]);
33      else
34        {
35        printf("Not enough data in %s\n",InputFile);
36        fclose(rxf);
37        exit(1);
38        }
39  fclose(rxf);
40
```

```
1
2  /* OBTAIN START TIME */
3  gettimeofday(&StartTime,NULL);
4
5
6  /* ENROLL MASTER IN PVM */
7  printf("ENROL MASTER IN PVM, ");
8  masterid = pvm_mytid();
9  printf("masterid = %d\n\n",masterid);
10
11
12 /* LOOP OF NumRuns SLAVES. */
13 DataOffset = 0;
14 for (run=0; run<NumRuns; run++)
15   {
16   printf("=============================================\n");
17   printf("STARTING RUN %d OF %d\n",run+1,NumRuns);
18
19   /* .PROCESS EACH HOST */
20   for (host=0; host<NumMachines; host++)
21     {
22     /* ..START SLAVES ON EACH HOST */
23     printf("  %s starting %d slaves",Machine[host].HostName,
24       Machine[host].NumSlavesAvailable);
25     StartSlaves(host);
26
27     /* ..SEND DATA TO EACH HOST */
28     printf("  sending data to %s slaves",Machine[host].HostName);
29     SendData(host,run);
30
31     } /* NEXT HOST */
32
33   /* ..RECEIVE DATA FROM SLAVES */
34   printf("\n  reading data from all slaves");
35   ReadData();
36
37   } /* NEXT SET OF NumRuns SLAVES*/
38
39 printf("=============================================\n");
40
41 /* OBTAIN END TIME */
42 gettimeofday(&EndTime,NULL);
43
44
45 /* PRINT TIMING RESULTS */
46 printf("\nTotal Run time : %d secs\n",EndTime.tv_sec - StartTime.tv_sec);
47
48 fprintf(logfile,"Range Compression Run time : %d secs\n\n",EndTime.tv_sec - StartTime.tv_sec);
49 fclose(logfile);
50
51 /* SAVE RNGMAST OUTPUT FILE */
52 printf("\nSAVE TO RNGMAST OUTPUT FILE %s\n",Outputfile);
53 PvmRngOutput ((float *)reOut,(float *)imOut);
54
55 return (1);
56 }
57
58
```



```
1
2  /*-------------------------------------------------------------------------------------------------*/
3  /* StartSlaves starts a set of slaves on a remote machine and if not                               */
4  /* successful will automatically clean up the remote machine.  The parameter passed is             */
5  /* offset in the Machine array that describes the host                                             */
6  /*-------------------------------------------------------------------------------------------------*/
7  int StartSlaves(int HostNumber)
8    {
9      int j,numt;
10
11     /* .SPAWNING OF SLAVES */
12     numt=pvm_spawn(SLAVENAME, (char**)0, PvmTaskHost,
13       Machine[HostNumber].HostName, Machine[HostNumber].NumSlavesAvailable,
14       tids);
15
16     if (numt < Machine[HostNumber].NumSlavesAvailable)                    /* if spawning problems */
17       {
18       printf(", FAILED!\n");
19       printf("** Trouble spawning slaves on %s. Ignoring host. Error codes are:\n",
20         Machine[HostNumber].HostName);
21
22       for (j=numt ; j<Machine[HostNumber].NumSlavesAvailable ; j++)
23         printf("TID %d %d\n",j,tids[j]);
24
25       for (j=0 ; j<numt ; j++)
26         pvm_kill(tids[j]);                                              /* kill healthy slaves */
27
28       return(START_FAILED);
29       }
30
31     printf(", successful\n");
32     return START_OK;
33   }
34
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* SendData sends the data required by each slave                          */
4 /*-------------------------------------------------------------------------*/
5 int SendData(int HostNumber, int run)
6   {
7     int j;
8
9     /* START EACH SLAVE */
10    for (j=0 ; j<Machine[HostNumber].NumSlavesAvailable ; j++)
11      {
12      pvm_initsend(PvmDataDefault);                          /* set up send buffer */
13      pvm_pkint(&DataOffset, 1, 1);                                  /* slave ID */
14      pvm_pkfloat(reRef, RngBin, 1);                       /* real reference vector */
15      pvm_pkfloat(imRef, RngBin, 1);                       /* imag reference vector */
16      pvm_pkfloat(reInp[DataOffset], RngBin, 1);              /* real input vector */
17      pvm_pkfloat(imInp[DataOffset], RngBin, 1);              /* imag input vector */
18      pvm_send(tids[j], 0);                              /* send actual msg to slave j */
19      DataOffset++;                                          /* increment slave ID */
20      }
21
22    printf(", successful\n");
23    return SEND_OK;
24  }
25
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* ReadData read the data back from each slave                             */
4 /*-------------------------------------------------------------------------*/
5 int ReadData(void)
6   {
7     int j;
8     int ReturnDataOffset;                                  /* returning slave ID */
9
10    for (j=0; j<SlavesPerRun; j++)
11      {
12      pvm_recv( -1, 5 );                                   /* receive msg type 5 */
13      pvm_upkint(&ReturnDataOffset, 1, 1);                  /*  receive slave ID */
14      pvm_upkfloat(reOut[ReturnDataOffset], RngBin, 1);    /* real result vector */
15      pvm_upkfloat(imOut[ReturnDataOffset], RngBin, 1);    /* imag result vector */
16      }
17    printf(", successful\n");
18
19    return READ_OK;
20  }
21
```

```
1
2  /*-------------------------------------------------------------------------*/
3  /* GetInitFile reads the setup file pvmcomp.ini for initialising the Range Compression parameters.  */
4  /*-------------------------------------------------------------------------*/
5  int GetInitFile (void)
6    {
7    char inpfile[]="/home/swuyts/pvmwork/pvmcomp.ini";
8    FILE *Ifile;
9
10   char name[20];
11   int i = 0;
12
13
14   /* READ INITIALISATION FILE */
15   Ifile = fopen(inpfile,"rt");
16   if (!Ifile)
17     {
18     printf("Cannot open %s\n", inpfile);
19     return(0);
20     }
21
22   fscanf(Ifile,"%s %d", name, &RngBin);
23   if ((strcmp(vars[i++], name))!=0)
24     {fclose(Ifile); return 0;}
25
26   fscanf(Ifile,"%s %d", name, &AzmBin);
27   if ((strcmp(vars[i++], name))!=0)
28     {fclose(Ifile); return 0;}
29
30   fscanf(Ifile,"%s %d", name, &AzmFilt);
31   if ((strcmp(vars[i++], name))!=0)
32     {fclose(Ifile); return 0;}
33
34   fscanf(Ifile,"%s %f", name, &SimFreq);
35   if ((strcmp(vars[i++], name))!=0)
36     {fclose(Ifile); return 0;}
37
38   fscanf(Ifile,"%s %f", name, &PulseWidth);
39   if ((strcmp(vars[i++], name))!=0)
40     {fclose(Ifile); return 0;}
41
42   fscanf(Ifile,"%s %f", name, &Bandwidth);
43   if ((strcmp(vars[i++], name))!=0)
44     {fclose(Ifile); return 0;}
45
46   fscanf(Ifile,"%s %f", name, &MaxAmpl);
47   if ((strcmp(vars[i++], name))!=0)
48     {fclose(Ifile); return 0;}
49
50   fclose(Ifile);
51   return(1);
52   }
53
54
```



```
1
2  /*-------------------------------------------------------------------------*/
3  /* GetConfigFile reads the virtual machine file confg.ini for initialising the PVM machine parameters.  */
4  /*-------------------------------------------------------------------------*/
5  int GetConfigFile (void)
6    {
7    char inpfile[]="/home/swuyts/pvmwork/config.ini";
8    FILE *Ifile;
9    int i;
10
11   /* READ CONFIG FILE */
12   Ifile = fopen(inpfile,"rt");
13   if (!Ifile)
14     {
15     printf("Cannot open %s\n", inpfile);
16     return(0);
17     }
18
19   fprintf(logfile,"============= Virtual Machine Description ==============\n");
20   fprintf(logfile,"\tHost\tSlaves\n");
21   fprintf(logfile,"=======================================================\n");
22
23   i = 0;
24   SlavesPerRun = 0;
25
26   while (!feof(Ifile))
27     {
28     fscanf(Ifile,"%s %d",Machine[i].HostName, &Machine[i].NumSlavesAvailable);
29     SlavesPerRun += Machine[i].NumSlavesAvailable;
30     if (Machine[i].NumSlavesAvailable != 0)
31       {
32        fprintf(logfile,"\t%s\t%d\tOK\n",Machine[i].HostName,Machine[i].NumSlavesAvailable);
33        i++;
34       }
35     }
36   NumMachines = i;
37
38   fclose(Ifile);
39
40
41   /* CHECK THAT TOTAL SLAVES < AzmBin */
42   if (SlavesPerRun > AzmBin)
43     {
44     printf ("The number of total slaves available in config.ini must be less than AZMBIN");
45     return(0);
46     }
47
48
49   /* CHECK THAT TOTAL SLAVES = POWER OF 2 */
50   if (fmod((float)SlavesPerRun,2.0))
51     {
52     printf ("The number of total slaves available in config.ini must be a power of 2");
53     return(0);
54     }
55
56
57   /* DETERMINE THE NUMBER RUNS NEEDED */
58   NumRuns = AzmBin / SlavesPerRun;
59
60   /* CHECK THAT TOTAL SLAVES IS A FACTOR OF AZMBIN */
61   if (fmod((float)AzmBin,(float)SlavesPerRun))
62     {
63     printf ("The number of total slaves available in config.ini must be\n");
64     printf ("a factor of AzmBin!\n");
65     return(0);
66     }
```

```c
fprintf(logfile,"=====================================================================\n");
fprintf(logfile,"Runs : %d\n",NumRuns);
fprintf(logfile,"=====================================================================\n\n");

return(1);
}


```

```c

/*--------------------------------------------------------------------------------------------------*/
/* PvmRngOutput prints the Range Compression output into the file rngcomp.asc                       */
/*--------------------------------------------------------------------------------------------------*/
void PvmRngOutput( float *rePrn, float *imPrn)
 {
 int i,j;
 FILE *outf;

 outf = fopen(OutputFile,"wt");
 if (!outf)
   {
   printf("Cannot open rngmast output file %s\n",OutputFile);
   return;
   }

 for( j=0 ; j<AzmBin ; j++)
   for (i=0; i<RngBin; i++)
     fprintf(outf,"%12.6f %12.6f\n",
        *(rePrn + j*MAXRNG + i), *(imPrn + j*MAXRNG + i));

 fclose(outf);
 }

/*==================================================================================================*/
```

```
1  /*-------------------------------------------------------------------------------------------------*/
2  /* PvmRng generates the range compression reference parameters by generating a chirp pulse at time 0.      */
3  /*-------------------------------------------------------------------------------------------------*/
4  #include "globals.h"
5
6
7  int PVMRng(void)
8    {
9    int i;
10   float Pulsetime, MidPulse, Offset, Phase;                          /* chirp pulse parameters */
11   float reRg[MAXRNG], imRg[MAXRNG];                                        /* range variables */
12   FILE *Rfile;
13
14
15
16   /* GENERATE PULSE FREQ BEHAVIOR */
17   PulseTime = PulseWidth*SimFreq;
18   MidPulse = (PulseWidth/2.0)*SimFreq-0.5;
19
20
21   /* GENERATE CHIRP PULSE */
22   for ( i=0 ; i<PulseTime; i++)
23     {
24     Offset = ((float)i-MidPulse)*((float)i-MidPulse);
25     Phase = Offset*PI*Bandwidth/(SimFreq*PulseTime);
26     reRg[i] = (float)cos((double)Phase) * MaxAmpl;
27     imRg[i] = (float)sin((double)Phase) * MaxAmpl;
28     }
29   for ( i=PulseTime; i<RngBin ; i++)
30     {
31     reRg[i] = 0.0;
32     imRg[i] = 0.0;
33     }
34
35
36   /* SEND TO RANGE FILE */
37   Rfile = fopen("/home/sawyts/pvmwork/rng.asc","wt");
38   if (!Rfile)
39     {
40     printf("Cannot open transmit output file\n");
41     return(0);
42     }
43   for ( i=0 ; i<RngBin; i++)
44     fprintf(Rfile,"%6.6f  %6.6f\n", reRg[i], imRg[i]);
45   fclose(Rfile);
46
47
48   return(1);
49   }
50
51 /*-------------------------------------------------------------------------------------------------*/
```

```c
/*-----------------------------------------------------------------------------------------*/
/*                                                                                         */
/*-----------------------------------------------------------------------------------------*/
/* Author: Shirley Wuyts                                                                   */
/* Date:  August 1997                                                                      */
/* Project: PVMCOMP                                                                        */
/* Task: RngComp.c                                                                         */
/*                                                                                         */
/* Description: This function acts as a PVM slave task for performing range compression on a one */
/* dimensional image vector.                                                               */
/*                                                                                         */
/* The slave reads the setup file then sends/receives compression data from the PVM master. */
/* Each slave is given a single azimuth vector of range returns to compare with the reference data. */
/*-----------------------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------------------*/
/* Include Files                                                                           */
/*-----------------------------------------------------------------------------------------*/
#include "globals.h"

/*-----------------------------------------------------------------------------------------*/
/* Exported Variable Definitions                                                           */
/*-----------------------------------------------------------------------------------------*/
int RngBin, AzmBin, AzmFilt;                          /* range and azimuth dimensions */
float PulseWidth, SimFreq;

/*-----------------------------------------------------------------------------------------*/
/* Module Macro Definitions                                                                */
/*-----------------------------------------------------------------------------------------*/
#define MAXVARS 5

/*-----------------------------------------------------------------------------------------*/
/* Module Variable Definitions                                                             */
/*-----------------------------------------------------------------------------------------*/
char *vars[MAXVARS] = {"RNGBIN", "AZMBIN", "AZMFILT", "SIMFREQ", "PULSEWIDTH"};
float reWin[MAXRNG*2], imWin[MAXRNG*2];                         /* window vars */
float reRef[MAXRNG*2], imRef[MAXRNG*2];                      /* reference vars */
float reInp[MAXRNG*2], imInp[MAXRNG*2];                          /* input vars */
float reTmpR[MAXRNG], imTmpR[MAXRNG];                         /* temp ref vars */
float reTmpI[MAXRNG], imTmpI[MAXRNG];                       /* temp input vars */
float reOut[MAXRNG*2], imOut[MAXRNG*2];                /* matched output vars */
char  destination[64];
int me;                                                            /* slave ID */

/*-----------------------------------------------------------------------------------------*/
/* Function Definitions                                                                    */
/*-----------------------------------------------------------------------------------------*/
int GetInitFile (void);
void Comp(void);
void RngCompOutput(float *rePrn, float *imPrn);
void fft(float *re, float *im, int bin, int dir);

```

```c

/*-----------------------------------------------------------------------------------------*/
/* RngComp calls the function GetInitFile for initialising the program parameters.        */
/* RngComp receives the range return data in a single azimuth vector and the reference data from the PVM */
/* master.                                                                                 */
/* RngComp calls the function Comp to perform range compression on these two vectors.     */
/* When a specific azimuth vector is received by a slave, RngComp sets up Comp to generate the rngspike file*/
/* which is the result of the reference data being range compressed with itself.  This is used for later */
/* azimuth compression on the data.                                                        */
/*-----------------------------------------------------------------------------------------*/
int main()
  {
  int i,j;

  int mytid;                                                     /* my task id */
  int master, msgtype;                                         /* comms stuff */



  /* INITIALISE ARRAYS AND ZERO PAD */
  for (i=0; i<(RngBin*2); i++)
    {
    reWin[i] = imWin[i] = 0.0;
    reRef[i] = imRef[i] = 0.0;
    reInp[i] = imInp[i] = 0.0;
    reOut[i] = imOut[i] = 0.0;
    }


  /* SET UP INITIALISATION FILE - READ FROM INI FILE */
  if (!GetInitFile())
    {
    printf("LINUX slave : (rngcomp) problem with Initialising File");
    return(0);
    }



  /* ENROLL SLAVE IN PVM */
  mytid = pvm_mytid();                                           /* my task id */



  /* RECEIVE DATA FROM MASTER */
  msgtype = 0;
  pvm_recv( -1, msgtype );                               /* receive msg type 0 */
  pvm_upkint(&me, 1, 1);                                           /* slave ID */
  pvm_upkfloat(reRef, RngBin, 1);                      /* real reference vector */
  pvm_upkfloat(imRef, RngBin, 1);                      /* imag reference vector */
  pvm_upkfloat(reInp, RngBin, 1);                          /* real input vector */
  pvm_upkfloat(imInp, RngBin, 1);                          /* imag input vector */



  /* CALCULATE RNGSPIKE FOR AZIMUTH COMPRESSION */
  if (me == RngBin/2)
    {
    for (i=0; i<RngBin; i++)
      {
      reTmpI[i] = reInp[i];
      imTmpI[i] = imInp[i];
      reTmpR[i] = reRef[i];
      imTmpR[i] = imRef[i];
      reInp[i] = reRef[i];
      imInp[i] = imRef[i];
      }
```

```
    Comp();
    sprintf(destination,"/home/swuyts/pvmwork/rngspike.asc");
    RngCompOutput((float *)reOut,(float *)imOut);
    for (i=0; i<MAXRNG; i++)
      {
      reInp[i] = reTmpI[i];
      imInp[i] = imTmpI[i];
      reRef[i] = reTmpR[i];
      imRef[i] = imTmpR[i];
      }
    }


  /* DO RANGE COMPRESSION CALCULATIONS */
  Comp();



  /* SEND RESULT TO MASTER */
  pvm_initsend( PvmDataDefault );                                   /* set up send buffer */
  pvm_pkint(&me, 1, 1);                                              /* send my slave id */
  pvm_pkfloat(reOut, RngBin, 1);                                     /* send real result */
  pvm_pkfloat(imOut, RngBin, 1);                                     /* send imag result */
  msgtype = 5;
  master = pvm_parent();
  pvm_send( master, msgtype );                                    /* send msg type 5 to master */



  /* TASK FINISHED, EXIT PVM BEFORE STOPPING */
  pvm_exit();
  return(1);
  }

/*-------------------------------------------------------------------------------------------------*/
/* Comp achieves range compression by windowing the reference function, converting both the windowed */
/* reference function and the input function to the frequency domain (by performing Fast Fourier Transforms)*/
/* , multiplying the two, then converting back to the time domain.                                  */
/* To perform the fast fourier transforms, Comp calls the function fft.                             */
/*-------------------------------------------------------------------------------------------------*/

void Comp(void)
  {
  int i,isign;
  float BW;

  if (me == AzmBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/rnginp.asc");
    RngCompOutput((float *)reInp,(float *)imInp);
    sprintf(destination,"/home/swuyts/pvmwork/rngref.asc");
    RngCompOutput((float *)reRef,(float *)imRef);
    }


  /* WINDOW REFERENCE FUNCTION */
  BW = PulseWidth*SimFreq-1.0;
  for (i=0; i<(RngBin*2); i++)
    {
    if (i<=(int)BW)
      {
      reWin[i]=(0.54+0.46*cos(2*PI*((float)i-BW/2.0)/BW));
      imWin[i]= reWin[i]*(-1);
      }
    reRef[i] *= reWin[i];
    imRef[i] *= imWin[i];
    }

  if (me == AzmBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/rngwin.asc");
    RngCompOutput((float *)reWin,(float *)imWin);
    sprintf(destination,"/home/swuyts/pvmwork/rngrefw.asc");
    RngCompOutput((float *)reRef,(float *)imRef);
    }


  /* PERFORM FOURIER TRANSFORM ON REFERENCE DATA */
  isign=1;
  fft(reRef,imRef,(RngBin*2),isign);

  if (me == AzmBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/rngreff.asc");
    RngCompOutput((float *)reRef,(float *)imRef);
    }


  /* PERFORM FOURIER TRANSFORM ON INPUT DATA */
  isign=1;
  fft(reInp,imInp,(RngBin*2),isign);

  if (me == AzmBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/rnginpf.asc");
    RngCompOutput((float *)reInp,(float *)imInp);
    }

  /* MULTIPLY BY REFERENCE FUNCTION */
  for (i=0; i<(RngBin*2); i++)
    {
    reOut[i]=reRef[i]*reInp[i]-imRef[i]*imInp[i];
    imOut[i]=imRef[i]*reInp[i]+reRef[i]*imInp[i];
    }

  if (me == AzmBin/2)
    {
    sprintf(destination,"/home/swuyts/pvmwork/rngcompf.asc");
    RngCompOutput((float *)reOut,(float *)imOut);
    }

  /* PERFORM IFFT ON ROW OF OUTPUT DATA */
  isign=-1;
  fft(reOut,imOut,(RngBin*2),isign);

  }
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* GetInitFile reads the setup file pvmcomp.ini for initialising the Range Compression parameters.  */
4 /*-------------------------------------------------------------------------*/
5
6 int GetInitFile (void)
7   {
8   char inpfile[]="/home/sunyts/pvmwork/pvmcomp.ini";
9   FILE *Ifile;
10
11  char name[20];
12  int i = 0;
13
14
15  /* READ INITIALISATION FILE */
16  Ifile = fopen(inpfile,"rt");
17  if (!Ifile)
18    {
19    printf("LINUX slave : (rngcomp) Cannot open %s\n", inpfile);
20    return(0);
21    }
22
23  fscanf(Ifile,"%s %d", name, &RngBin);
24  if ((strcmp(vars[i++], name))!=0)
25    {fclose(Ifile); return 0;}
26
27  fscanf(Ifile,"%s %d", name, &AzmBin);
28  if ((strcmp(vars[i++], name))!=0)
29    {fclose(Ifile); return 0;}
30
31  fscanf(Ifile,"%s %d", name, &AzmFilt);
32  if ((strcmp(vars[i++], name))!=0)
33    {fclose(Ifile); return 0;}
34
35  fscanf(Ifile,"%s %f", name, &SimFreq);
36  if ((strcmp(vars[i++], name))!=0)
37    {fclose(Ifile); return 0;}
38
39  fscanf(Ifile,"%s %f", name, &PulseWidth);
40  if ((strcmp(vars[i++], name))!=0)
41    {fclose(Ifile); return 0;}
42
43
44  fclose(Ifile);
45  return(1);
46  }
```

```
1
2 /*-------------------------------------------------------------------------*/
3 /* RngCompOutput prints the output of the range compression processes into the file "destination" named when*/
4 /* called by the Comp routine.                                                                              */
5 /*-------------------------------------------------------------------------*/
6
7 void RngCompOutput( float *rePrn, float *imPrn)
8   {
9   int j;
10  FILE *outf;
11
12  outf = fopen(destination,"wt");
13  if (!outf)
14    {
15    printf("LINUX slave : (rngcomp) Cannot open file %s for writing!\n",destination);
16    exit(1);
17    }
18    *
19  for (j=0; j<(RngBin*2); j++)
20        fprintf(outf,"%12.6f %12.6f\n",*(rePrn + j),*(imPrn + j));
21
22  fclose(outf);
23  }
24
25
26 /*=========================================================================*/
```

10. APPENDIX D: RESULTS OF PVM COMPRESSION SOFTWARE

```
TEST 1:  PENTIUM

============= Virtual Machine Description ==============
        Host    Slaves
=========================================================
        pentium 16      OK
=========================================================
Runs : 32
=========================================================

Range Compression Run time : 49 secs

[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 19:03:00 1997


============= Virtual Machine Description ==============
        Host    Slaves
=========================================================
        pentium 16      OK
=========================================================
Runs : 16
=========================================================

Azimuth Compression Run time : 35 secs

[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 20:41:46 1997
```

```

============= Virtual Machine Description ==============
        Host    Slaves
=========================================================
        pentium 32      OK
=========================================================
Runs : 16
=========================================================

Range Compression Run time : 50 secs

[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 19:03:00 1997


============= Virtual Machine Description ==============
        Host    Slaves
=========================================================
        pentium 32      OK
=========================================================
Runs : 8
=========================================================

Azimuth Compression Run time : 34 secs
[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 22:15:22 1997
```

```
 1
 2 ************* Virtual Machine Description ************
 3         Host    Slaves
 4 ******************************************************
 5         pentium 64      OK
 6 ******************************************************
 7 Runs : 8
 8 ******************************************************
 9
10 Range Compression Run time : 53 secs
11
12 [tB0040000] pentium LINUX 3.3.11
13 [tB0040000] ready Wed Sep 17 19:03:00 1997
14
15
16
17 ************* Virtual Machine Description ************
18         Host    Slaves
19 ******************************************************
20         pentium 64      OK
21 ******************************************************
22 Runs : 4
23 ******************************************************
24
25 Azimuth Compression Run time : 36 secs
26 [tB0040000] pentium LINUX 3.3.11
27 [tB0040000] ready Wed Sep 17 22:15:22 1997
```

```
 1
 2 ************* Virtual Machine Description ************
 3         Host    Slaves
 4 ******************************************************
 5         pentium 128     OK
 6 ******************************************************
 7 Runs : 4
 8 ******************************************************
 9
10 Range Compression Run time : 58 secs
11
12 [tB0040000] pentium LINUX 3.3.11
13 [tB0040000] ready Wed Sep 17 22:45:42 1997
14
15
16
17 ************* Virtual Machine Description ************
18         Host    Slaves
19 ******************************************************
20         pentium 128     OK
21 ******************************************************
22 Runs : 2
23 ******************************************************
24
25 Azimuth Compression Run time : 42 secs
26 [tB0040000] pentium LINUX 3.3.11
27 [tB0040000] ready Wed Sep 17 22:45:42 1997
```

```
TEST 1:  ALPHA

============== Virtual Machine Description ==============
         Host    Slaves
=========================================================
         alpha   16      OK
=========================================================
Runs : 32
=========================================================

Range Compression Run time : 74 secs

[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 19:48:28 1997
[tB0080000] alpha ALPHA 3.3.11
[tB0080000] ready Wed Sep 17 19:47:15 1997


============== Virtual Machine Description ==============
         Host    Slaves
=========================================================
         alpha   16      OK
=========================================================
Runs : 16
=========================================================

Azimuth Compression Run time : 44 secs
[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 22:22:29 1997
[tB0080000] alpha ALPHA 3.3.11
[tB0080000] ready Wed Sep 17 22:21:15 1997
```

```
============== Virtual Machine Description ==============
         Host    Slaves
=========================================================
         alpha   32      OK
=========================================================
Runs : 16
=========================================================

Range Compression Run time : 83 secs

[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 19:48:28 1997
[tB0080000] alpha ALPHA 3.3.11
[tB0080000] ready Wed Sep 17 19:47:15 1997


============== Virtual Machine Description ==============
         Host    Slaves
=========================================================
         alpha   32      OK
=========================================================
Runs : 8
=========================================================

Azimuth Compression Run time : 52 secs
[tB0040000] pentium LINUX 3.3.11
[tB0040000] ready Wed Sep 17 22:22:29 1997
[tB0080000] alpha ALPHA 3.3.11
[tB0080000] ready Wed Sep 17 22:21:15 1997
```

```
1
2 ============= Virtual Machine Description =============
3         Host    Slaves
4 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
5         alpha   64      OK
6 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
7 Runs : 8
8 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
9
10 Range Compression Run time : 86 secs
11
12 [tB0040000] pentium LINUX 3.3.11
13 [tB0040000] ready Wed Sep 17 19:48:28 1997
14 [tB0040000] [tB040b] libpvm [pid16055]: mksocs() connect: Connection refused
15 [tB0040000] [tB040e] libpvm [pid16054]: mksocs() connect: Connection refused
16 [tB0040000] [tB040d] libpvm [pid16056]: mksocs() connect: Connection refused
17 [tB0040000] [tB040f] libpvm [pid16039]: mksocs() connect: Connection refused
18 [tB00B0000] alpha ALPHA 3.3.11
19 [tB00B0000] ready Wed Sep 17 19:47:15 1997
```

```
1
2 TEST 1:  HPPA1
3
4 ============= Virtual Machine Description =============
5         Host    Slaves
6 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
7         hppa1   16      OK
8 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
9 Runs : 32
10 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
11
12 Range Compression Run time : 79 secs
13
14 [tB0040000] pentium LINUX 3.3.11
15 [tB0040000] ready Wed Sep 17 20:01:00 1997
16 [tB00B0000] hppa1 HPPA 3 3.11
17 [tB00B0000] ready Wed Sep 17 19:51:41 1997
18
19
20
21 ============= Virtual Machine Description =============
22         Host    Slaves
23 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
24         hppa1   16      OK
25 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
26 Runs : 16
27 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
28
29 Azimuth Compression Run time : 59 secs
30 [tB0040000] pentium LINUX 3.3.11
31 [tB0040000] ready Wed Sep 17 22:29:49 1997
32 [tB00B0000] hppa1 HPPA 3.3.11
33 [tB00B0000] ready Wed Sep 17 22:20:28 1997
```

```
 1
 2 Range Compression Run
 3
 4 [t80040000] pentium LINUX 3.3.11
 5 [t80040000] ready Wed Sep 17 20:01:00 1997
 6 [t80040000] [t80418] libpvm [pid7616]: pvm mytid(): Can't contact local daemon
 7 [t80040000] [t8041d] libpvm [pid7621]: pvm mytid(): Can't contact local daemon
 8 [t80040000] [t80420] libpvm [pid7624]: pvm mytid(): Can't contact local daemon
 9 [t80040000] [t8041d] libpvm [pid7621]: pvm recv(): Can't contact local daemon
10 [t800B0000] hppa1 HPPA 3.3.11
11 [t800B0000] ready Wed Sep 17 19:51:41 1997
12 [t800B0000] loclconn() accept: Too many open files
13 [t800B0000] loclconn() accept: Too many open files
14 [t800B0000] tm connect() can't open t-auth file: Too many open files
15 [t800B0000] tm connect() can't open t-auth file: Too many open files
```



```
 1
 2 TEST 1:  HPPA2
 3
 4 Range Compression Run
 5
 6 [t80040000] pentium LINUX 3.3.11
 7 [t80040000] ready Wed Sep 17 20:10:28 1997
 8 [t80040000] netoutput() timed out sending to hppa2 after 15, 182.543284
 9 [t80040000]  hd dump() ref 1 tB0000 n "hppa2" a "" ar "HPPA"
10 [t80040000]            lo "" so "sm" dx "/usr/local/pvm3/lib/pvmd" ep "" bx "" wd "" sp 1000
11 [t80040000]            se hppa2 mtu 4096 f 0x0 e 0 txq 31
12 [t80040000]            tx 6 rx 5 rtt 0.477862
13 [t80040000] netinput() bogus pkt from hppa2
14 [t80040000] netinput() bogus pkt from hppa2
15 [t800B0000] hppa2 HPPA 3.3.11
16 [t800B0000] ready Wed Sep 17 20:03:07 1997
```