NASA Contractor Report 4456
DOT/FAA/RD-92/20

Real-Time Processing of Radar Return on a Parallel Computer

David D. Aalfs

GRANT NGT-50414
AUGUST 1992

NASA Contractor Report 4456
DOT/FAA/RD-92/20

Real-Time Processing of Radar Return on a Parallel Computer

David D. Aalfs
Radar Systems Laboratory
Electrical and Computer Engineering Department
Clemson University
Clemson, South Carolina

Prepared for
Langley Research Center
under Grant NGT-50414

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Program
1992

TABLE OF CONTENTS

Page

LIST OF FIGURES v

ACKNOWLEDGMENTS vii

CHAPTER

1 INTRODUCTION 1

The Microburst 1
Pulse Doppler Radar 5
Principles of Operation 5
Microburst Detection 11
Signal Processing 13
Parallel Processing 15
Problem Statement 17

2 COMPUTATIONAL REQUIREMENTS 19
Radar Parameters 20
Algorithms 21

3 CONCURRENT PERFORMANCE 28

Communication Overhead 28
Speedup Measures and Efficiency 31

4 THE TRANSPUTER 33

Transputer Architecture 33
Microprocessor and Memory 36
Links 37
Occam Model of Concurrent Processing 38
Primitive Processes and Channels 39
Constructions 40
Real Time Issues 41

Table of Contents (Continued)

Page

5 CONCURRENT PROCESSING IMPLEMENTATION 44
Algorithms 44
Pulse-Pair Estimator 44
Fast Fourier Transform 45
Autoregressive Modeling 46
Hardware 48
Benchmarks 48
Fast Fourier Transform 51
Autoregressive Model 52
Speedup 54

6 CONCLUSIONS 56

APPENDICES 59

A. Radar Parameters 60
B. Occam Software Listing 61

REFERENCES 73

LIST OF FIGURES

Figure		Page
1.1	Symmetric Microburst.	2
1.2	"S"–curve for a simulated microburst.	4
1.3	"F"–factor hazard index.	6
1.4	Block Diagram of a Doppler radar.	7
1.5	Transmitted signal for a pulse Doppler radar.	9
1.6	Gated return for a pulse Doppler radar.	9
1.7	Doppler return from a moving target.	10
1.8	Doppler power spectrum of a Gaussian process.	12
1.9	Doppler power spectrum of a typical range cell.	14
2.1	Algorithm Suite I Flow Diagram.	23
2.2	Algorithm Suite II Flow Diagram.	24
2.3	Algorithm Suite III Flow Diagram.	25
2.4	Algorithm Suite IV Flow Diagram.	26
2.5	Performance Parameters.	27
4.1	Transputer architecture	34
4.2	Examples of transputer networks	35
4.3	Transputer family statistics	37
5.1	Block diagram of the transputer evaluation set-up.	49
5.2	Range cell display.	50
5.3	FFT benchmarks with pruning.	51
5.4	FFT benchmarks with no zero padding.	52
5.5	Autoregressive modeling benchmarks.	53
5.6	Speedup and efficiency measurements.	54

List of Figures (Continued)

Page

5.7 Plot of speedup. 55

A.1 Radar Parameters. 60

ACKNOWLEDGMENTS

The author would like to express his gratitude to Dr. E. G. Baxa for his valuable guidance and encouragement. He also thanks Dr. J. J. Komo and Dr. F. M. Cholewinski for sacrificing the time to serve as committee members.

The National Aeronautics and Space Administration is to be thanked for its financial support under the Graduate Student Researchers Program and for supplying the INMOS Transputer hardware and software used in this research. Special thanks are extended to the Antenna and Microwave Research Branch at Langley Research Center for their technical support and to Corey Gehman and Jayanth Thyamagundlam for their advice and assistance.

Finally, he expresses his appreciation to his wife and family for their patience and support throughout his graduate studies.

CHAPTER 1

INTRODUCTION

1.1 The Microburst

Since the late 1970's, a form of low altitude windshear, called a microburst, has become recognized as a significant hazard to aircraft during takeoff and landing. The term microburst was first used by Fujita to describe a rapid downflow of air that upon impacting the ground forms a horizontal outflow extending less than 4 km in diameter (Figure 1.1) [1]. Observations of roughly 75 microbursts during the Joint Airport Weather Studies (JAWS) Project conducted at Denver's Stapleton International Airport in 1982 revealed that they have a relatively short life span, usually less than 15 minutes, with severe shear lasting only 2 to 4 minutes. Within that small window of time and distance, however, headwind to tailwind differentials can approach 100 MPH. Under such conditions, a pilot may have less than 30 seconds to make a successful recovery [2].

Typically, when an aircraft encounters a microburst at low altitude, it first experiences a performance increasing headwind, followed by a severe downdraft, and finally a performance decreasing tailwind. The rapid loss of altitude caused by the downdraft and the ensuing loss of altitude and airspeed due to the tailwind can bring an aircraft too close to the ground with too little airspeed to escape a crash. It has been determined from measurements taken in the JAWS Project that some microbursts are so severe that once entered, there is no chance for recovery [2]. Compounding the problem is the deceptively benign visual appearance of many microbursts. The warning signs of a microburst can be as subtle as a column of rain associated with the downdraft or, if the rain evaporates above the ground, packets of windblown dust near the ground. It should be noted that windshear detection devices are presently

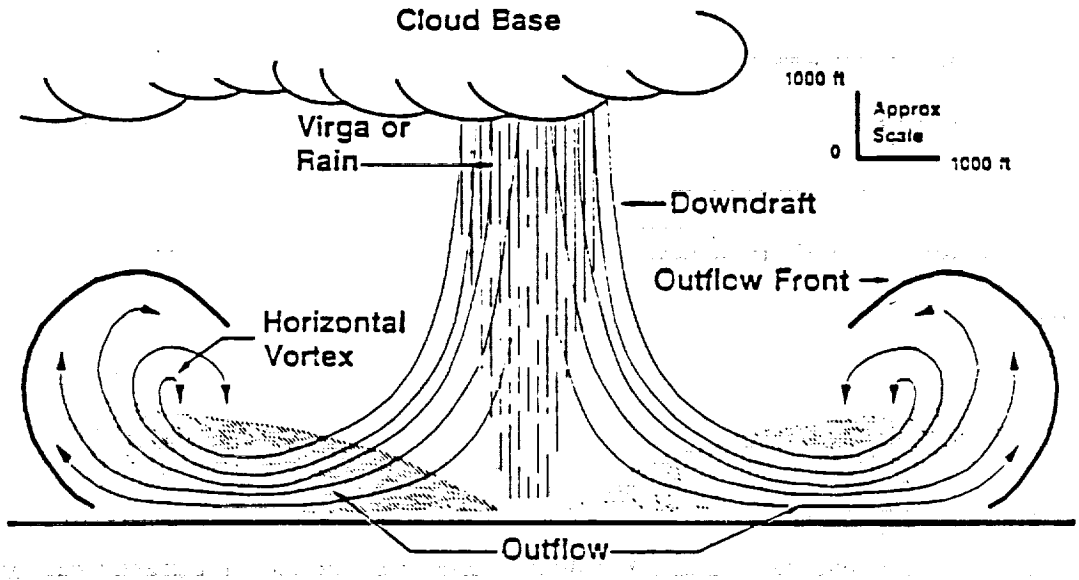

Figure 1.1 Symmetric Microburst.

installed on some commercial aircraft, but these are so called reactive systems. Reactive systems are based on in situ devices and can only detect a hazard once it has been entered.

The need to provide advanced warning of windshear conditions has spurred a joint effort between NASA and the FAA to develop a new generation of airborne Doppler weather radar operating in the X-band (8-12GHz). Doppler weather radar is actually one of several candidate sensors, including lidar and infrared, for a forward-looking airborne microburst detector. Each of these sensors requires similar signal processing, but they rely on different scattering mechanisms [3]. Since the predominant type of scattering target may vary from small particles of dust in one microburst to droplets of rain in another microburst, a combined system incorporating all three of the above sensors may be necessary to provide reliable detection under all atmospheric conditions. The upshot is that sophisticated signal processing will be necessary to identify microbursts in real time.

Part of the signal processing effort involves isolating a "signature" for microbursts that can be used to indicate potentially hazardous situations. The horizontal component of the windspeed versus range for the headwind/downdraft/tailwind sequence gives what is called the "S"–curve. Beginning at ranges nearest to the aircraft, the windspeed has a large negative value in the headwind, diminishing to zero at the heart of the downdraft, and becoming a large positive value in the tailwind. The "S"–curve is a signature detectable by a pulse Doppler radar capable of estimating average windspeed at various ranges. This sensor coupled with some form of pattern recognition scheme may be able identify the "S"–curve due to a microburst in the flight path. Figure 1.2 shows an "S"–curve generated by plotting mean windspeed estimates versus range for a simulated microburst generated by the Airborne Windshear Doppler Radar Simulation (AWDRS) Program [4].

Another form of hazard indication that has gained wide acceptance is the "F"–factor proposed by Bowles and Targ [5]. It is an index independent of the aircraft

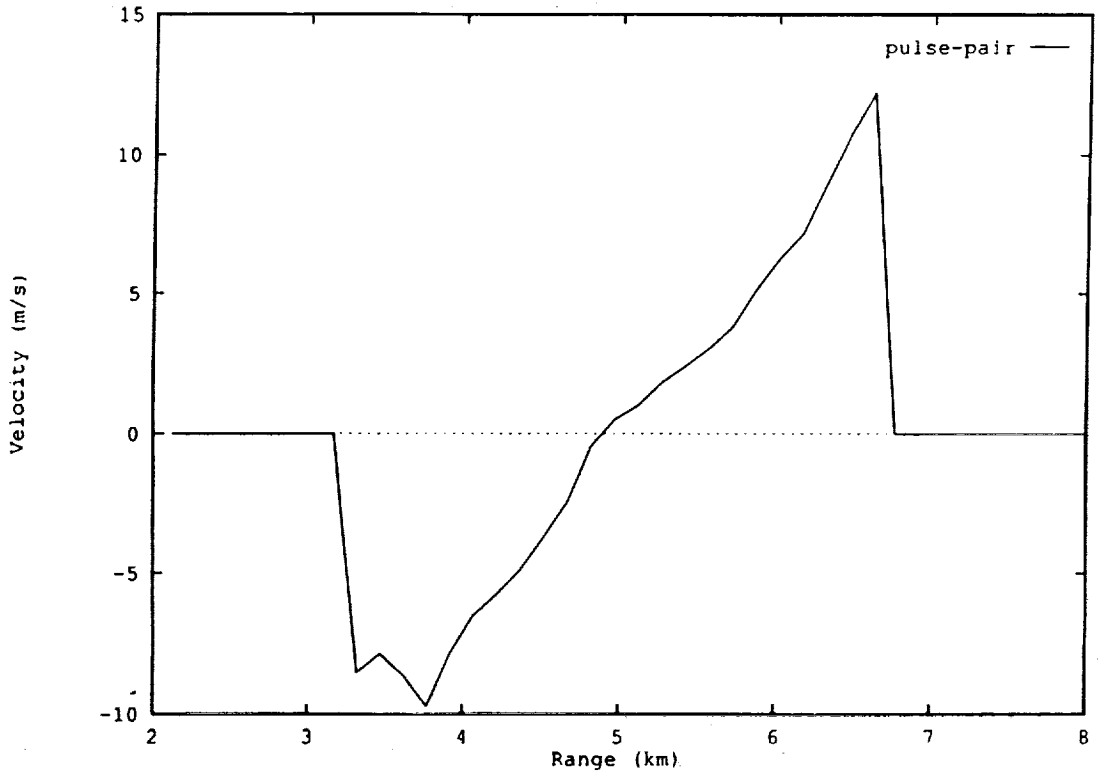

Figure 1.2 "S"–curve for a simulated microburst.

weight and thrust capability and is based upon measurable atmospheric conditions. The "F"–factor is defined as

$$F = \frac{\dot{W}_x}{g} - \frac{W_h}{V}$$

where:

$\dot{W}_x$ = spatial derivative of the horizontal component of wind velocity,

W_h = vertical component of wind velocity,

g = acceleration due to gravity,

V = the airspeed of the aircraft.

To get a feeling for what the "F"–factor means in physical terms, consider that a positive spatial derivative of the horizontal wind velocity indicates an increasing tailwind which has a performance decreasing effect, while a positive vertical wind velocity indicates a performance increasing updraft. Threshold values for F have been placed between 0.1 and 0.15 [5]. Figure 1.3 shows how "F"–factor can be interpreted as an indicator of safe operation.

1.2 Pulse Doppler Radar

Pulse Doppler radar is a sensor of particular interest in the windshear detection problem. This is not surprising since these instruments have been used for years in meteorological research of storm dynamics,hydrology , and cloud and precipitation physics [3, 6, 7, 8, 9].

1.2.1 Principles of Operation

The block diagram in Figure 1.4 shows the basic elements of a pulse Doppler radar. Beginning with the transmitter, a high power amplifier is used to produce a spectrally clean signal at a radio frequency f_t. A pulse modulator switches the amplifier on and off generating a train of pulses of duration τ and separated by the interpulse period (IPP), the reciprocal of the pulse repetition frequency (PRF). Figure 1.5 shows a conceptual drawing of the transmitted signal.

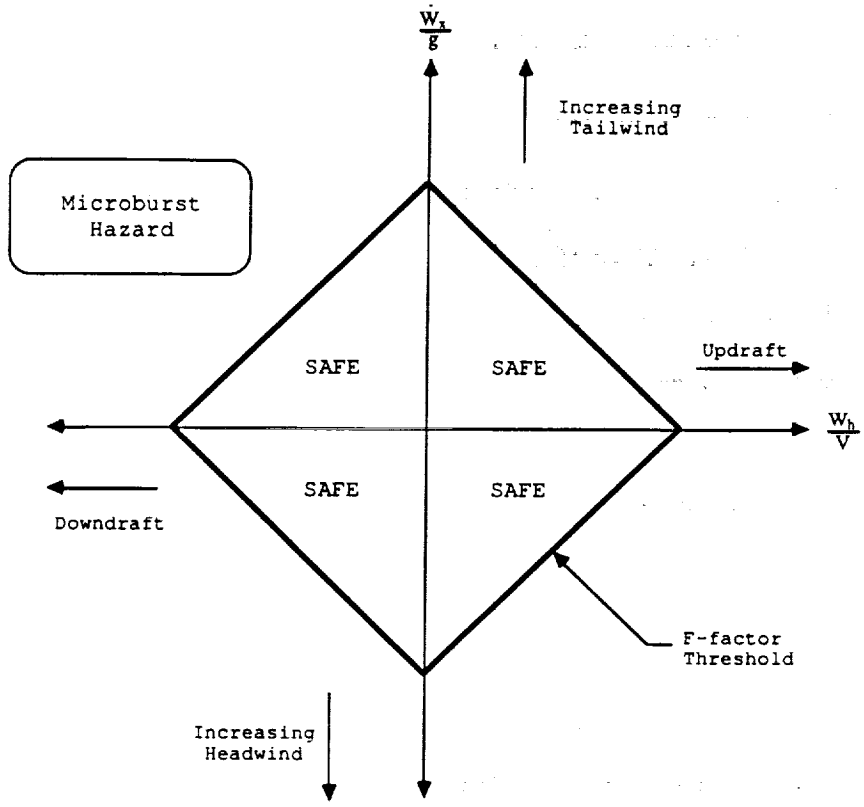

Figure 1.3 "F"–factor hazard index.

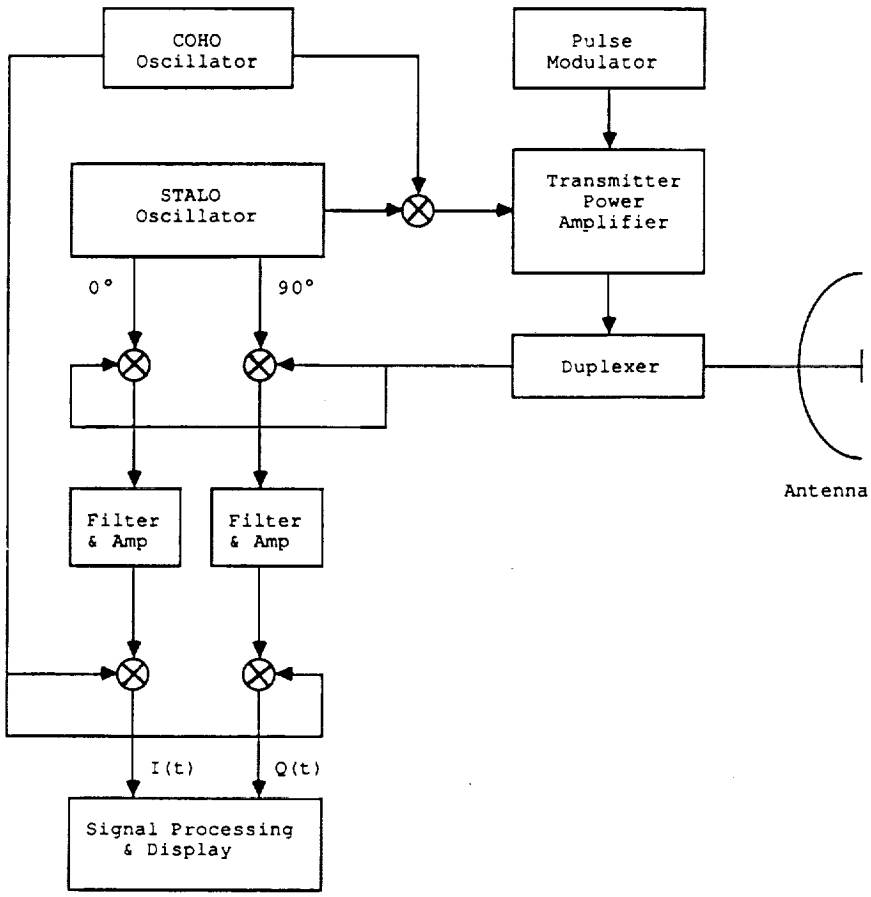

Figure 1.4 Block Diagram of a Doppler radar.

At the antenna, the pulse train is radiated as electromagnetic energy which is backscattered by targets in the path of the beam. In most cases, a single antenna is shared by the transmitter and receiver to reduce the space, weight, and cost requirements. In a shared antenna radar, referred to as monostatic, a duplexer is used to switch the antenna from the transmitter to the receiver during the period of time that the amplifier is turned off, (IPP $-\tau$). After a small delay to assure decoupling from the transmitter, the receiver "listens" for backscattered returns from the previous pulse.

The range of a target can be determined by measuring the time it takes for a pulse to travel to the target and have its reflected energy return to the receiver. The receive time between pulses can then be divided into time segments called range cells which indicate the range of returns that fall into a particular time interval (Figure 1.6). The maximum unambiguous range is given by

$$R_{max} = \frac{c}{2PRF},$$

where c is the speed of light.

By taking advantage of the Doppler shift principle, a pulse Doppler radar can also measure the radial velocity of a target. When an electromagnetic wave with frequency f_t is reflected by an object moving away from the radar platform at a relative velocity v, the frequency seen at the receiver is shifted by

$$f_d = -\frac{2vf_t}{c}.$$

Figure 1.7 illustrates the Doppler effect. The maximum unambiguous Doppler velocity is given by

$$V_{max} = PRF\left(\frac{c}{4f_t}\right).$$

With the aid of signal processing then, a pulse Doppler radar can map the radial velocities of targets out to its maximum unambiguous range.

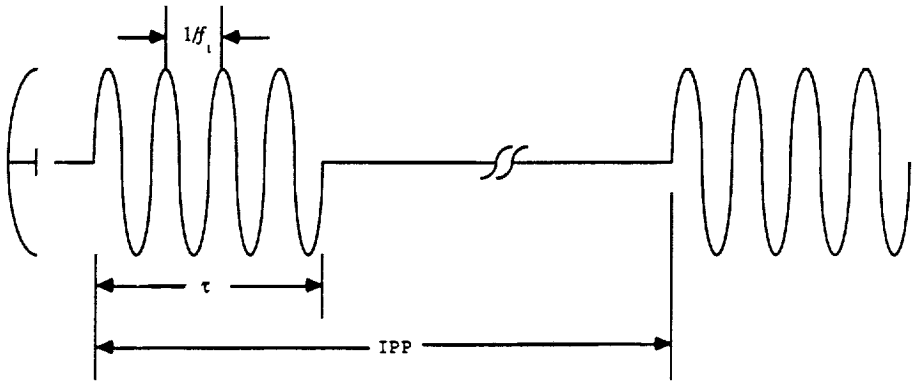

Figure 1.5 Transmitted signal for a pulse Doppler radar.

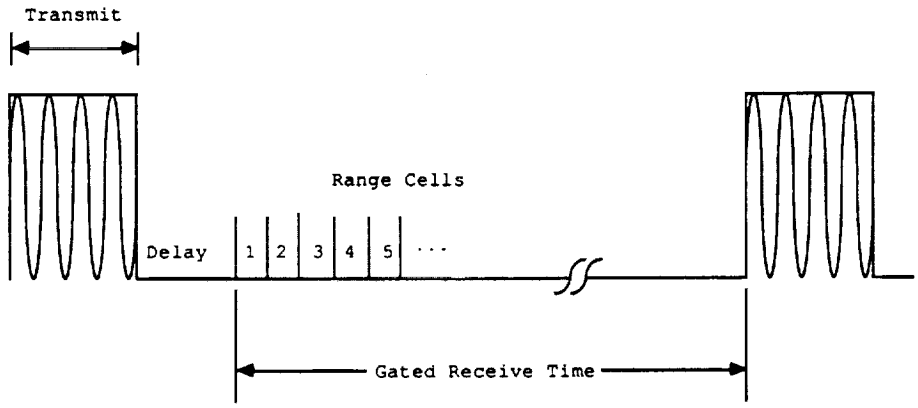

Figure 1.6 Gated return for a pulse Doppler radar.

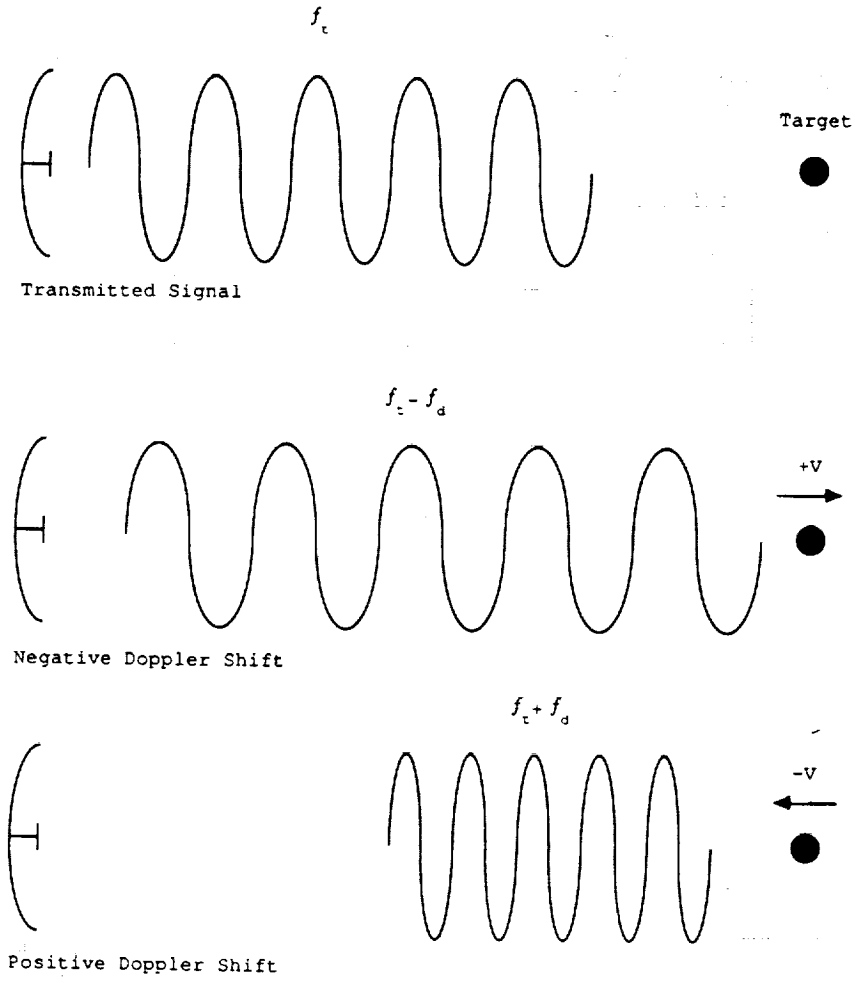

Figure 1.7 Doppler return from a moving target.

The targets that a weather radar is depending on as reflectors are windblown particles such as rain, insects, dust, or changes in the refractive index. Scattering targets that are swept along with the air mass and are illuminated by the radar provide information about the winds within the resolution volume. By the central limit theorem, the return signal of a large number of randomly distributed targets can be modelled by a narrowband Gaussian process. The effects of windshears, turbulence, and antenna motion combine to broaden the Doppler spectral width, but do not invalidate the Gaussian approximation [3, 10, 11]. Ideally, the Doppler power spectrum might look like the one represented in Figure 1.8 where the mean represents the average radial velocity of particles in the illuminated volume and the variance is the spread of the Doppler velocities. Estimates of these parameters are performed in the signal processing portion of the radar.

1.2.2 Microburst Detection

The detection of microbursts with a pulse Doppler radar is complicated by several factors. One of these factors is the low reflectivity levels exhibited by certain types of microbursts. Preliminary research has been directed towards microbursts with significant levels of precipitation known as wet microbursts [5, 12]. The classification of wet and dry microbursts is rather qualitative, but a rainfall rate greater than 25 dBZ generally denotes a wet microburst and a rainfall rate less than 20 dBZ generally denotes a dry microburst. The strong return levels from wet microbursts make them the simplest case for evaluating various detection algorithms. Dry microbursts will call for more sophisiticated signal processing since the primary sources of reflections, particles of dust and insects, exhibit a much lower reflectivity of radar energy.

Along with the possibility of a low energy weather returns, the problem of microburst detection by an airborne radar is complicated by the presence of strong ground return or clutter. When the aircraft is in low altitude flight, such as during takeoff or landing, a portion of the antenna beam is likely to illuminate objects on

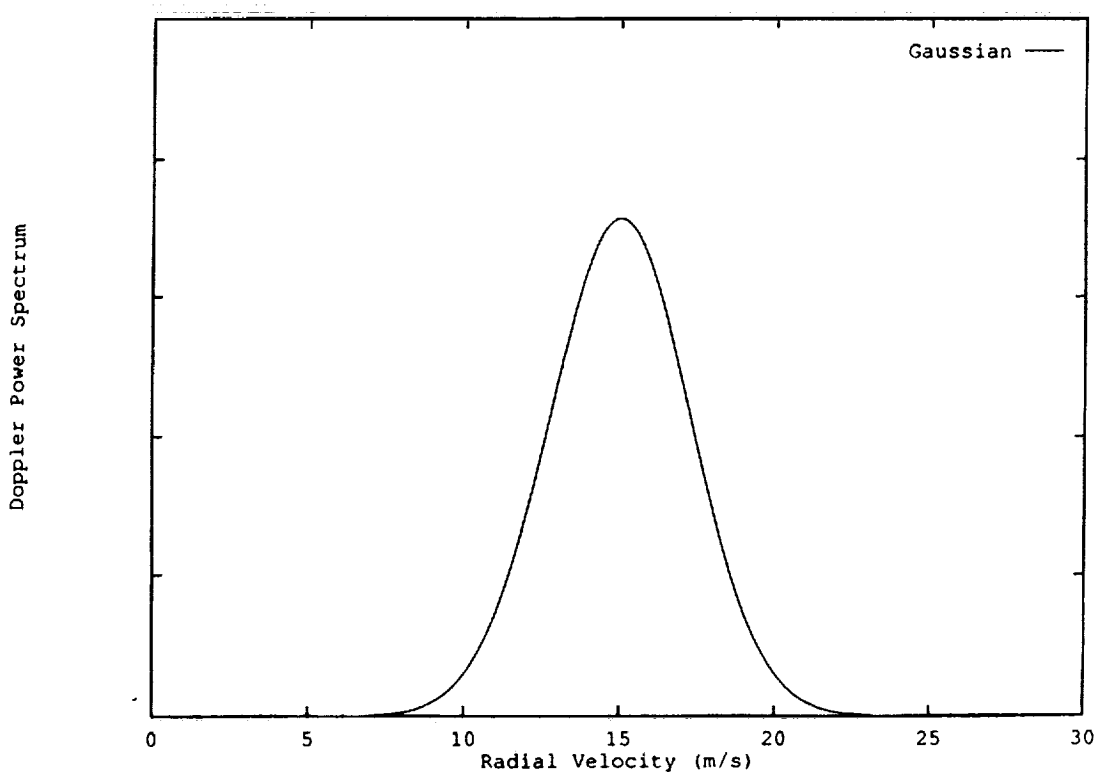

Figure 1.8 Doppler power spectrum of a Gaussian process.

the ground such as buildings, trees, or cars on a freeway. The return from these powerful reflectors can utterly obliterate any returns due to weather [10]. Most of the clutter energy appears around zero Doppler referenced to the aircraft ground speed and is the result of strong returns from stationary objects through the main beam of the antenna. Additional discrete clutter due to returns from moving objects on the ground or returns from large objects through the antenna sidelobes may appear at frequencies shifted away from zero Doppler. Figure 1.9 shows the Doppler power spectrum of range cell data generated by the AWDRS program [4]. Notice the narrow clutter mode located near 0 m/s and the broader weather mode located around 10 m/s. The clutter problem may be partially abated by tilting the antenna higher with respect to the glide slope so that less of the beam is directed towards the ground, but then the beam is directed more in the downdraft portion of the microburst, running the risk of missing the highly characteristic horizontal outflow. The upshot is that some form of clutter rejection digital filtering may be necessary to suitably enhance the signal to clutter ratio [13, 14, 15].

1.2.3 Signal Processing

Estimation of weather parameters is typically performed on a per range cell basis. To improve the accuracy of the estimate, sets of complex sampled data points are collected from the radar IF output over a number of pulses for each range cell. The number of points in each range cell record is determined by the length of time over which the statistical properties of the targets can be assumed to be stationary.

The conventional approach to processing radar signals has been to use a series of special purpose arithmetic units followed by a programmable microcontroller for processing each range cell [16]. The typical stages of the range cell computation for a pulse Doppler weather radar might include some form of clutter rejection, estimation of the Doppler power spectrum and/or estimation of spectral parameters, and a suitable detection algorithm. The most popular technique for computing the Doppler

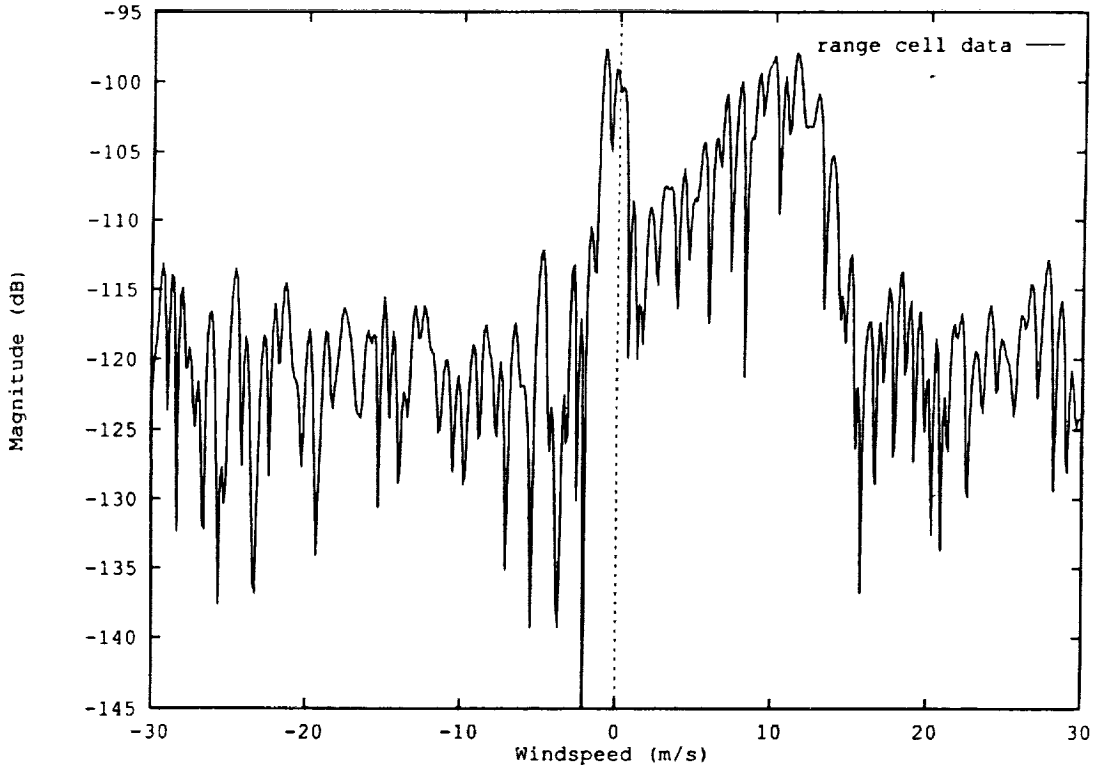

Figure 1.9 Doppler power spectrum of a typical range cell.

power spectrum is the fast Fourier transform (FFT) which has been implemented on high speed, programmable digital signal processing (DSP) chips. The mean wind-speed and spectral width can be computed from the Doppler power spectrum, or obtained directly using the pulse-pair algorithm [17, 18, 19].

Since each of the range cells can be processed independently using the same set of algorithms, this processor can operate on range cells in either pipeline fashion or in parallel depending on the throughput requirements. To make the computation fully parallel, a duplicate processing bank for each range cell is provided. Special purpose hardware has been necessary until recently to achieve real time data rates. A step towards greater flexibility has been made with the programmable DSP board which takes advantage of multiple high speed DSP chips. Many of these boards are programmable in high level Languages such as C. DSP boards are a powerful tool for radar signal processing, but they are, to an extent, still special purpose devices, and as such they can require significant development time. With the recent advances in computer technology and parallel processing, it has become feasible to perform real time radar signal processing on arrays of general purpose microprocessors.

1.3 Parallel Processing

Radar signal processing lends itself well to concurrent computation when the problem is decomposed by range cells. This is a natural way to break down the computation because interprocessor communication is unnecessary for range cell dependent products. Message passing is only necessary for distributing the data throughout the network and recollecting the results. In addition, a balanced computational load across all the processors is assured when the range cells are equally distributed.

Within the sphere of concurrent processing, there are a number of issues to be addressed. First consider the size of each individual processing element, known as the grain size. The grain size will be dictated by the complexity of the necessary algorithms and the memory requirements for each node. For the problem decomposition

described above, each element will be responsible for processing one or more range cells, each composed of a set of complex data points . The size of the range cell data records and the need for sophisticated signal processing algorithms calls for a large grain size parallel computer.

Next there is the choice of Single Instruction Multiple Data (SIMD) or Multiple Instruction Multiple Data (MIMD) architectures. On a SIMD machine, all the nodes execute the same sequence of instuctions on their respective sets of data. An example of SIMD architecture is the systolic array where a number of special purpose small grain elements are interconnected to perform a specific function such as matrix multiplication. This approach is reasonable for the radar application because each range cell is to receive exactly the same processing, but typically these machines are small grain size and/or massively parallel, consisting of 16K or more processors. A SIMD computer with a smaller number of large grain size nodes is a feasible architecture, but they are not generally available because for a large grain size a MIMD computer can offer essentially the same processing capability with greater flexibility [20]. The majority of large grain parallel computers are of the MIMD class where each node stores and executes its own instruction set independently.

MIMD computers can further be classified as shared memory or distributed memory. Shared memory computers have a common bus and memory for all the nodes. They use their memory more efficiently and are easier to program, but all the microprocessors have to compete for the same resources. This leads to bus contention, severely limiting the number of processors possible for a potential system. With distributed memory, each node has its own local memory and bus, thereby eliminating the problem of bus contention. Communication takes place through message passing, which can only exist between nodes that are directly connected through links. Conceptually, a shared memory machine can have "full interconnect" capability among its processors by communicating through common memory locations [20]. However,

there should be no significant drawback to the distributed memory machine as opposed to the shared memory machine in this respect for the radar signal processing application, since very little interprocessor communication is required when range cell decomposition is used to parallelize problem.

In addition to sequential and parallel processing, a third possibility is vector processing which can be considered a hybrid of the sequential and parallel schemes. These machines are optimized to operate on whole vectors of data at one time. By employing vector registers, fully pipelined vector functional units, and fully pipelined vector loads and stores, they are able to reduce the number of instruction and memory fetches, eliminate memory latency with register to register operations, as well as achieve arithmetic speedup. In the past, vector processors were mostly limited to use in super computers such as the Cray Y-MP, but more recently they have become available for more general purposes [21].

1.4 Problem Statement

In order to avoid a potential windshear hazard, it has been estimated that a pilot is going to need an advanced warning time of 15 to 40 seconds [12]. A pulse Doppler radar offers the potential for a look ahead capability over this time period, but to get the relevant information to the pilot in a timely manner, the required signal processing much be accomplished in real time. In the past, radar signal processing has been performed on special purpose computing devices that offer extremely high performance at the expense of flexibility. The emergence of concurrent processing techniques and hardware has opened the possibility of achieving real time data rates on an array of parallel computers that are easily reprogrammable and reconfigurable. The advantage gained will be to shift the primary development effort from hardware to software, greatly enhancing the adaptability of the system to future changes.

Investigation of parallel processing techniques has largely been carried out on a PC based parallel computer called the transputer. Chapter 2 will introduce the

processor requirements that have been specified for a microburst detector and consider how they translate into requirements for a multicomputer. Chapter 3 will discuss ways of analyzing the performance of multicomputers, and Chapter 4 will describe the transputer system and the Occam model of concurrent processing. Chapter 5 will describe a transputer based implementation of a concurrent radar signal processor including algorithms, hardware, and benchmarks. Finally, Chapter 6 will bring all these issues into focus, presenting conclusions drawn from this research and making recommendations for future work.

CHAPTER 2

COMPUTATIONAL REQUIREMENTS

The computational requirements for a real time microburst detector are determined by certain hardware parameters of the windshear radar, the computational load of the desired algorithms, and the amount of processor margin factored into the calculation. Estimates of these requirements put forth by E. G. Baxa and M. A. Richards are collected and summarized in a report by the Radar Signal/Data Processor (RSDP) Task Force [22]. Processor requirements for several sets of algorithms of varying complexity, referred to as algorithm suites, are given in terms of the following well known performance measures for uniprocessor computers:

1. million instructions per second (MIPS),
2. million floating point operations per second (MFLOPS) [23],
3. thousand bytes of memory storage (KByte).

These values are intended to provide a guideline for the kinds of hardware choices that are available. It should be noted that neither MIPS nor MFLOPS are in themselves sufficient measures for comparative computer performance. Differences in machine instruction sets and user programs make it impossible to construct a single performance metric that is meaningful for all types of computers and application programs. The only meaningful measure of computer performance is the execution time for the specific application program of interest to the user [21]. Since it is not reasonable to measure the execution time of the application on all possible types of computers, and the nature of the application may not be completely defined beforehand, these performance numbers serve to reduce the field of candidate systems. To illustrate how values for these parameters are obtained, an analysis following that given in the RSDP report is recreated below and extended to the case of concurrent processing.

2.1 Radar Parameters

An X-band airborne Doppler weather radar currently operated by the Antenna and Microwave Research Branch (AMRB) of NASA Langley Research Center to flight test the windshear dectection system is the source of hardware parameters used to calculate the processing requirements [24]. Many of the radar's characteristics are adjustable allowing it to operate at several PRFs, pulse widths, scan rates, *et cetera.* The flexibility of the AMRB test radar makes it a reasonable choice for a benchmark system since a production windshear radar is likely to operate within its worst case configuration. Appendix A lists the relevant radar parameters and their possible settings, but the most challenging configuration for a real time processor would be as follows:

1. PRF = 9581 pulses per second,
2. number of range cells (N_R) = 69,
3. scan rate ($\dot{\theta}$) = 37.5° per second,
4. scan width (θ_S) = 60°,
5. azimuth lines per scan width sector (N_θ) = 40.

Note that these values do not represent the worst case for each parameter independently, rather they constitute a collective worst case configuration.

With the radar parameters specified, the maximum allowed processing time can now be calculated. Consider the case where an averaged and sampled value from each range bin is collected over a certain number of IPPs to form a range cell record and all of these range cell records are block processed in real time. Then the time available in which to perform the necessary processing, t_p, is determined by the PRF and the number of complex data points, N_I, in each range cell data record.

$$t_p = \frac{N_I}{PRF}.$$

The value of t_p reflects the amount of time to process the returns from one pulse of the radar, so on a uniprocessor that time must be divided among all the range cells leaving t_R as the time to process each range cell where t_R is given by

$$t_R = \frac{t_p}{N_R}.$$

If all the range cells can be processed concurrently, then nearly the full t_p could be dedicated to processing each range cell record. Given the worst case radar parameters and a N_I of 128 pulses, t_p would be 13.36 msec and t_R would be 0.19 msec.

The time available for processing can be increased somewhat by taking advantage of the entire scan time between azimuth lines. Typically the returns from N_I pulses are processed per line of azimuth and any additional time required for the antenna to scan to the next azimuth line can be considered free for processing. Thus the full t_p is determined by

$$t_p = \frac{\theta_S}{\dot{\theta}\, N_\theta}.$$

The degree to which t_p is increased depends on the PRF. For large values of the PRF the improvement can be significant. Applying the worst case radar parameters to the revised expression for t_p gives 40 msec (or 0.58 msec per range cell on a uniprocessor). That amounts to nearly a three fold increase over the time calculated by the previous method.

2.2 Algorithms

Once the available processing time has been identified, then the algorithmic complexity is assessed to determine the number and type of operations that must occur within that time and to determine the amount of memory storage necessary to hold the program and data. To facilitate analysis, four suites of algorithms are

presented, ranging from a minimal baseline system to a growth system. The following is a list of algorithms presently being considered for the microburst detector:

1. time domain clutter rejection filtering,
2. spectral domain clutter rejection filtering,
3. Fourier spectral estimate,
4. inverse Fourier transform,
5. autoregressive spectral estimate,
6. time domain mean and width estimate (pulse-pair),
7. Fourier domain mean and width estimate (pulse-pair),
8. hazard factor computation.

The four algorithm suites are shown in flow diagram form in Figures 2.1 – 2.4. Suite I consists of a 2^{nd} order IIR Moving Target Indicator (MTI) clutter rejection filter followed by a pulse-pair processor and hazard factor computation. This suite is considered to offer the baseline level of acceptable performance. In suites II and III the 2^{nd} order IIR filter is replaced by a 39^{th} order FIR filter followed by a spectral estimate, some additional clutter filtering in the spectral domain, and finally spectral domain mean and width estimates. In suite II the spectral estimation is done by a Fast Fourier Transform (FFT) while in suite III it is done by autoregressive modeling. The growth system, suite IV, includes all of the algorithms listed in the first three suites. Obviously some of these algorithms are redundant, but this suite is designed to anticipate future changes in the system.

For each of the algorithm suites, an estimate of the number of instructions, number of floating point operations (FLOPs), and required memory locations is made. Dividing the number of instructions and the number of floating point operations by

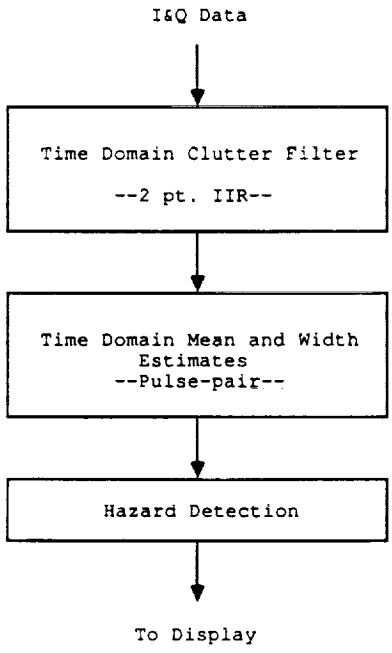

Figure 2.1 Algorithm Suite I Flow Diagram.

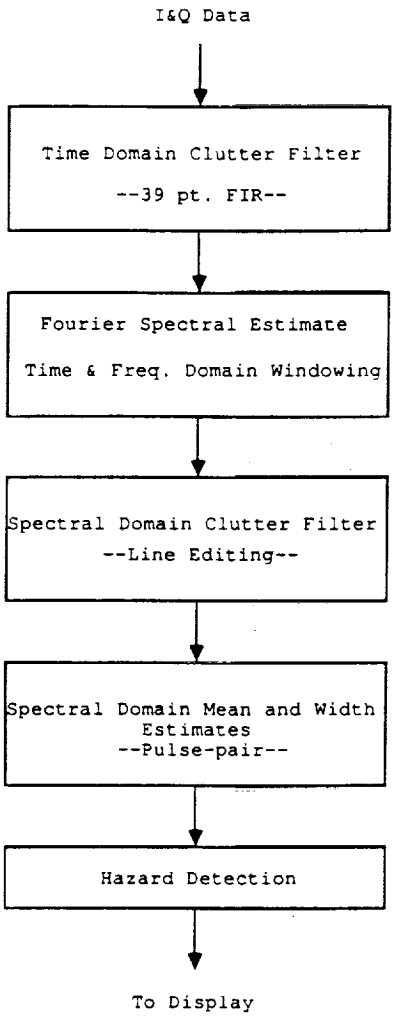

Figure 2.2 Algorithm Suite II Flow Diagram.

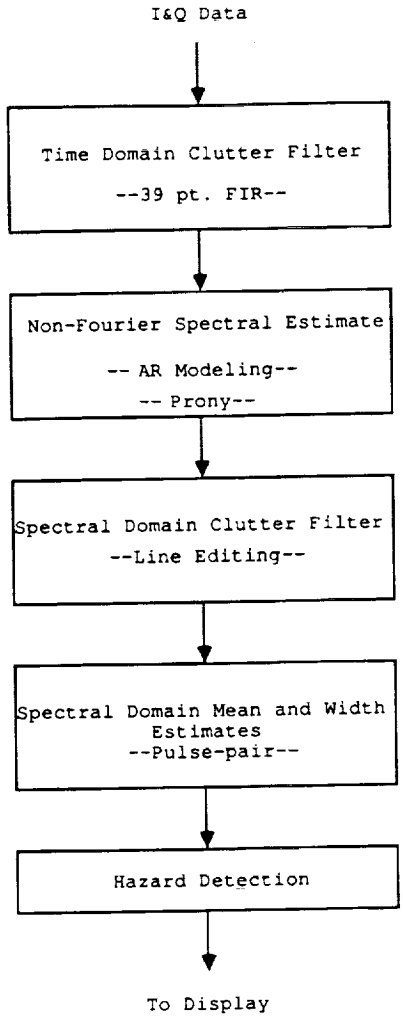

Figure 2.3 Algorithm Suite III Flow Diagram.

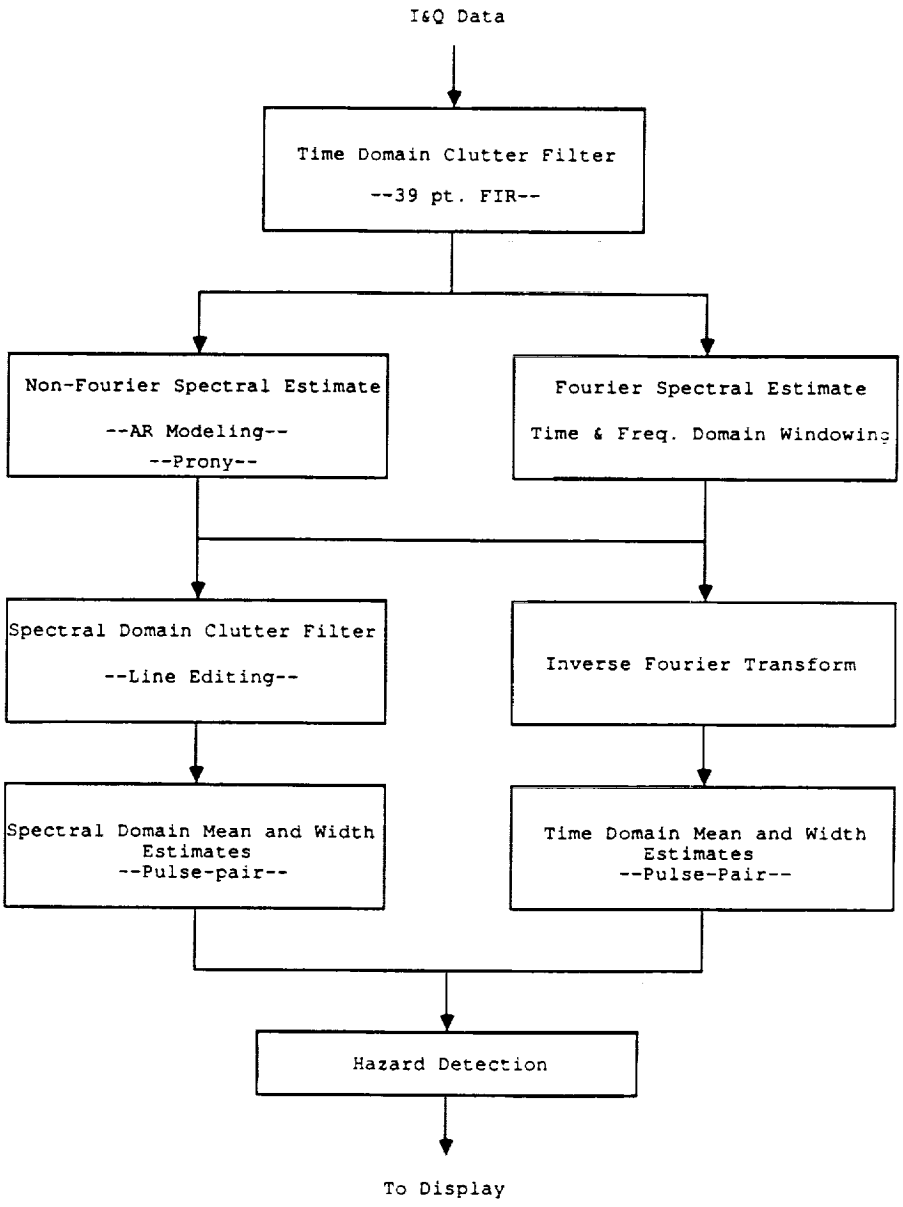

Figure 2.4 Algorithm Suite IV Flow Diagram.

	Total			Per Range Cell		
Suite	MFLOPS	MIPS	Kbytes	MFLOPS	MIPS	Kbytes
I	13	0.03	285	0.19	0.0004	4.13
II	68	0.006	566	0.99	0.0001	8.20
III	121	5.31	566	1.75	76.96	8.20
IV	169	5.34	711	2.45	77.39	10.30

Figure 2.5 Performance Parameters.

the available processing time gives the desired performance parameters MIPS and FLOPS respectively.

Finally, a processor margin of 100% is factored into the analysis. Processor margin is an attempt to account for inaccuracies of MFLOPS and MIPS as measures of computer performance and allow for future growth of the system. For a parallel processor, the processor margin can also be used to account for overheads associated with concurrency, for example communication overhead. Figure 2.5 summarizes the resulting computational requirements for a uniprocessor and for a multicomputer in terms of performance per range cell.

CHAPTER 3

CONCURRENT PERFORMANCE

The previous chapter introduced some performance metrics that are commonly used to evaluate single processor computers; MFLOPS, MIPS, and KBytes. These same parameters, along with the interprocessor communication bandwidth (measured in Mbytes/sec), are used to characterize the hardware of a multicomputer. Additional measures are useful to quantify application specific issues such as how the problem scales with the number of processors applied to it, what communication and synchronization overheads are associated with it, and what fraction of the problem is inherently sequential. While total execution time is still the ultimate criterion for the performance of multiple processor computers, measures such as speedup, efficiency, and communication overhead add valuable insight into the behavior of concurrent machines and may point to ways of improving their performance.

3.1 Communication Overhead

Time spent on interprocessor communication on a parallel computer, in the simplest view, can be interpreted as a drawback to the multiprocessor approach in comparison with the uniprocessor alternative. It has been argued that this is an unfair evaluation since interprocessor communication times should in fact be compared to the time to access tiered or virtual memory on a sequential computer [20]. The following discussion will ignore this point and concentrate on ways of estimating the overhead due to communication.

Following the notation suggested by Fox [20], the fractional communication overhead of an application is defined as

$$f_C = \frac{T_{comm}}{T_{calc}},$$

where T_{comm} is the total time spent on communication and T_{calc} is the total time spent on calculations. T_{comm} and T_{calc} can be written in terms of hardware parameters

$$t_{comm} = \text{the average time to communicate a word between two nodes.}$$

$$t_{calc} = \text{the average time the perform a calculation.}$$

Both of these parameters are loosely defined because the best strategy for measuring their values often depends on the specific application. For example, the average length of a message can have a significant effect on t_{comm} due to the startup latencies associated with each communication. In scientific applications t_{calc} is often assumed to be the time for a floating point operation, but the various types of floating point operations can involve a wide range of execution times. The relative occurrence of each type of floating point operation in the code may need to be taken into account to arrive at an accurate estimate of t_{calc}. These issues, among others, would have to be considered in designing a scheme for accurately measuring t_{comm} and t_{calc}, but often a rough estimate is sufficient for analysis purposes. A simple method for estimating the ratio of t_{comm} to t_{calc} is to use the ratio of a multicomputer's floating point performance to its communication bandwidth as follows:

$$\frac{t_{comm}}{t_{calc}} \sim \frac{MFLOPS}{\frac{Mbytes}{sec}}.$$

The importance of f_C is that for good performance, it should have a relatively small value, usually on the order of unity, and it should not increase dramatically as more processors are applied to the problem. If f_C is strongly dependent on the number of processors, that may indicate that the problem is not very scalable.

As an illustration of how f_C might be calculated, consider a hypothetical multicomputer intended to provide signal processing for a pulse Doppler radar windshear detector. For simplicity, it will comply with the following assumptions.

1. The computational requirements from Chapter 2 are satisfied.

2. The number of nodes is equal to the number of range cells, N_R.

3. A single communication is required to distribute each range cell.

4. The radar is in the worst case configuration.

5. N_I equals 128 samples (arbitrary).

6. The algorithm suite corresponds to the growth system.

Given the above conditions, T_{comm} and T_{calc} could be estimated by:

$$T_{comm} \sim 2N_I N_R t_{comm},$$
$$T_{calc} \sim N_{FLOPS} t_{calc},$$

where N_{FLOPS} is the total number of floating poing operations determined in the computational requirements. By plugging in the values for the worst case configuration, the communication overhead is approximated by

$$f_C \sim c \frac{t_{comm}}{t_{calc}},$$

where

$$c = \frac{2N_I N_R}{N_{FLOPS}} = 0.0026.$$

The small value for c is due to the computational overkill of the growth suite of algorithms and, to a certain extent, an underestimation of the amount of communication resulting from assumption 3. Such a small value of c may indicate that distributing each range cell computation onto more than one processesor may yield better performance for that particular algorithm suite.

3.2 Speedup Measures and Efficiency

Perhaps the most important measure of performance for parallel computers is speedup, S(N), which is the ratio of the execution time on a single processor to the execution time on a parallel computer with N processors.

$$S(N) = \frac{T_1}{T_N}$$

Dividing the speedup by N gives the efficiency which is often interpreted as the speedup per node.

$$E(N) = \frac{S(N)}{N}$$

where,

$$0 \leq E(N) \leq 1.$$

The efficiency can be reduced from its ideal value of unity by a number of factors. The concurrent algorithm may not be as good as its sequential counterpart, or even where the same algorithm is used, the concurrent implementation may require additional instructions to control the distribution of the problem over multiple processors. Load balancing can also be an important issue since the problem can only run as fast as the slowest node, and of course the communication overhead that was discussed in the previous section has an effect on the efficiency. Of all these factors, the communication overhead is the most significant for the radar signal processing application. The algorithms being used require little modification to accomodate concurrency, and load balancing is assured as long as an equal number of range cells are processed on each node.

Bounds on both the speedup and efficiency have been established in previous work by Amdahl [25] and Eager *et al* [26]. Since the efficiency follows directly from the speedup, only bounds on speedup will be considered here.

Every algorithm can be divided into portions that are independent and can thus be performed in parallel and portions that are inherently sequential. The speedup of

an algorithm is limited by the extent to which its code is sequential. More specifically, if f is the fraction of the code that is sequential, then Amdahl's law [25] states that speedup is bounded by:

$$S(N) \leq \frac{1}{f + \frac{(1-f)}{N}}.$$

For example, if 20% of an algorithm is inherently sequential, then no matter how many processors are applied to the problem, the maximum achievable speedup is 5. It has been argued that Amdahl's law leads to an overly pessimistic evaluation of large scale parallel computing. One argument is that many algorithms can be written to reduce the number of sequential operations to an acceptably low level if they are designed for concurrency from the start rather than modified from existing sequential code [20]. It has also been suggested that instead of running a fixed sized problem on different numbers of processors, a more realistic approach is to fix the execution time and solve the largest size problem possible [27].

A technique for bounding speedup similar in spirit to that of Amdahl is the use of the average parallelism measure proposed by Eager, Zahorjan and Lazowska [26]. The average parallelism of an algorithm, A, can be defined in a number of ways, but here it will be defined as the average number of processors that would be active during the execution of an algorithm if an unlimited number of processors were available. The upper and lower bounds on the speedup in terms of the average parallelism and the number of processors are given by

$$\frac{NA}{N + A - 1} \leq S(N) \leq \min(N, A).$$

For range cell processing, the average parallelism can be estimated by the number of range cells, N_R. Assuming there is a node for each range cell the bounds on speedup become

$$\frac{{N_R}^2}{2N_R - 1} \leq S_N \leq N_R.$$

The theoretical limits on speedup will be compared to actual measured values in Chapter 5.

CHAPTER 4

THE TRANSPUTER

A system for processing pulse Doppler radar signals has been implemented on a transputer-based multicomputer. Transputers are a family of computers that combine a microprocessor, memory, and four serial communication links all on a single VLSI chip. Figure 4.1 shows a block diagram of the T-800 transputer architecture. Transputers can be housed in an ordinary PC on full length add-in cards and the transputer nodes on those cards can be arranged in a variety of network configurations controllable in software. While compilers for the transputer include FORTRAN, C, and PASCAL, they are most easily programmed in OCCAM, a language designed for concurrent processing that is tightly coupled to the architecture of the transputer. In the sections below the transputer architecture will be discussed, OCCAM will be introduced both as a language and as a model for concurrent processing, and finally transputer and OCCAM support for real time processing will be covered.

4.1 Transputer Architecture

A distinguishing characteristic of the transputer is the cohesiveness and simplicity of its design. This can be seen in the hardware support provided for the OCCAM model of concurrent processing, in the freedom that the software developer is given from considering strictly hardware issues, in the ease with which members of the transputer processor line can be interchanged with only minor software changes, and in the inherent self-sufficiency of the transputer chip. With very little external circuitry a transputer chip can be made into a fully functional multicomputer node. Networks of transputer nodes communicate by passing messages over their serial links. Since each node is equipped with four links, transputer networks can assume topologies with degree less than or equal to four. Some examples of possible transputer networks are shown in Figure 4.2 [28].

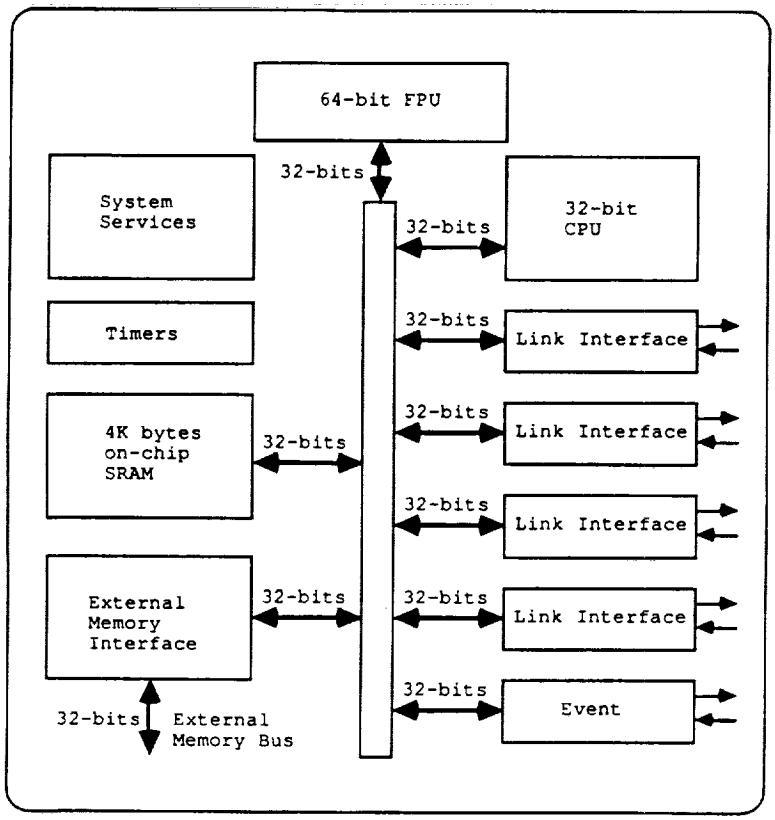

Figure 4.1 Transputer architecture.

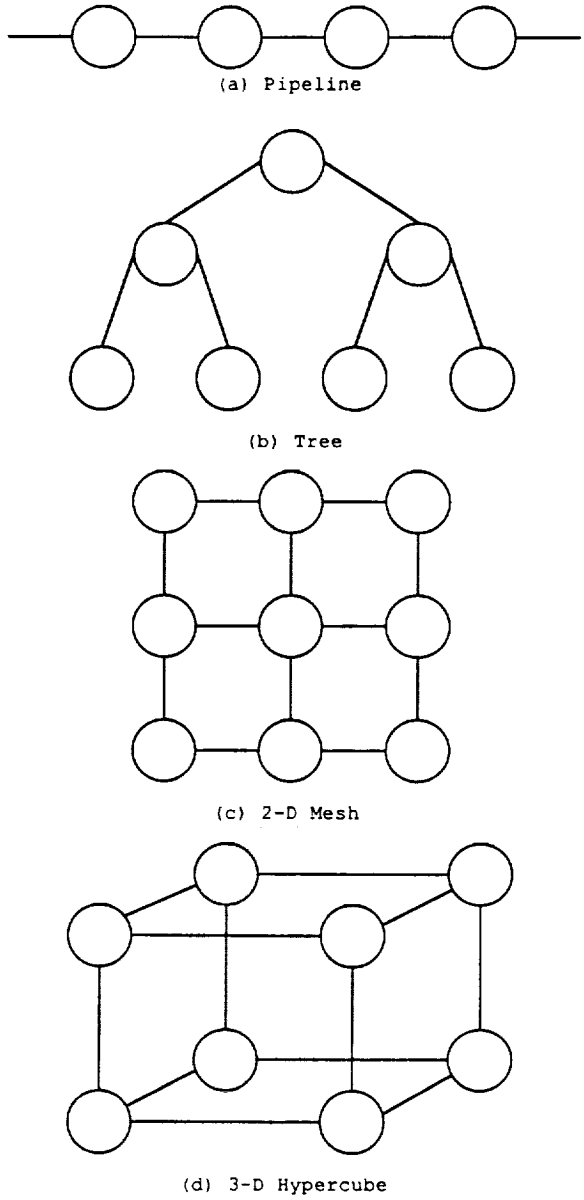

Figure 4.2 Examples of transputer networks.

4.1.1 Microprocessor and Memory

The transputer microprocessors come in 16 and 32 bit varieties and are essentially reduced instruction set computers (RISC) except for certain instructions that support scheduling and message passing [29]. An individual transputer is a sequential processor operating on a single instruction stream. A powerful feature of the transputer is its hardware process scheduler that performs high speed multitasking. Task switching times are typically about 1 microsecond for OCCAM processes [30]. The ability to multitask allows individual transputers to simulate concurrency. A program made up of multiple processes can be run on a single transputer or a network of transputers with only slight software modifications. This means that applications can be developed and tested on a single transputer and then mapped on to a network without changing the program logic.

There are two levels of memory hierarchy on a transputer node. The fastest is the on-chip static RAM (SRAM), normally used by compilers as a register stack for storing often used addresses and variables but also addressable by programmers who wish to optimize performance. The access time for on-chip memory is one cycle which on a 30 MHz T800 equals 33 nanoseconds. External dynamic RAM (DRAM) requires more cycles to access, but up to 4 Gbytes can be addressed. An on-chip external memory interface drives the external DRAM without any additional circuitry. The difference between using internal or external memory is generally transparent to the user except for the speed.

Figure 4.3 summarizes the characteristics of the more well known members of the transputer family. The T800 has an on-chip 64-bit IEEE floating-point processor (FPU) [23]. The T800 FPU provides much greater floating point performance than that of T212 or T414 which use a run-time subroutine package to do floating point operations. The latest generation of transputer is the T9000 which is not yet commercially avialable, but boasts an order of magnitude increase in performance over the T800.

	T212	T414	T800			T9000
Clock speed (MHz)	20	20	17.5	20	30	50
Bus width (bits)	16	32	32	32	32	32
On chip SRAM (Kbytes)	2	2	4	4	4	16
MIPS	10	10	8.75	10	15	200
MFLOPS	*	0.1	1.31	1.5	2.25	25

Figure 4.3 Transputer family statistics.

Based on the computational requirements from Chapter 2 and the transputer performance specifications and assuming linear speedup, a network of roughly 6 T800 transputers would be required to implement the baseline suite, 30 T800s for suite II, 54 T800s for suite III, and 76 T800s for the growth suite. If the T9000 was used, the required number of processors would be reduced to roughly 1 for the baseline suite, 3 for suite II, 5 for suite III, and 7 for the growth suite.

4.1.2 Links

The four communication links provide the sole means of interprocessor communication in transputer networks. Within a network, processors run asynchronously until a communication needs to take place between two of the nodes. At this point the two processors are synchronised by the links' hand-shaking hardware, the communication takes place, and the processors proceed to run asynchronously.

When a communication takes place, data is transferred directly from the memory space of one processor to the memory of the other via the link's DMA controller. The T212, T414, and T800 links can all operate at 0.625, 1.25, or 2.5 Mbytes/sec and the T9000 links will be able to operate at 12.5 Mbytes/sec. Even though the T9000 has much better maximum performance than previous models, all the transputer models are link compatable so that mixed networks can be built.

A high degree of concurrency is built into the transputers link architecture. Each link is bi-directional, or full duplex, meaning that communication can take place in

both directions simultaneously. In addition, all four links can be active simultaneously, while at the same time the processor can be executing instructions. If the chip has a floating point unit, as does the T800, then that too can operate concurrently.

4.2 Occam Model of Concurrent Processing

OCCAM takes its name from the 14^{th} century English philosopher William of Occam and his famed "Occam's razor": *Entia non sunt multiplicanda praeter necessitatem.* Loosely translated this means don't add complication beyond necessity and is the underlying philosophy of OCCAM, to keep it simple. Concurrency and communication are built into the language according to the Communicating Sequential Processes (CSP) model of concurrency proposed by C. A. R. Hoare of Oxford University [31]. The CSP model of concurrency is based on the concept of building parallel applications from networks of sequential processors communicating through synchronized point-to-point channels. Key elements of CSP that have been subsequently adopted by OCCAM are include the following:

1. a guarded command to allow for non-determinism,
2. a parallel command to specify concurrent execution,
3. input and output primitives for communication between parallel processes,
4. point-to-point one way channels,
5. strong typing of variables and channels to assure secure communications.

In OCCAM the basic programming entities are processes which begin, perform some action, and eventually terminate [30]. Processes executing in parallel and communicating through channels form the basis of the OCCAM model of concurrency.

4.2.1 Primitive Processes and Channels

All processes are ultimately made up of three primative processes: assignment, input, and output. The purpose of an assignment command is to change the value of a variable. This can occur by changing it to a specified constant or to the value of an expression. An example of assignment is

$$x := y + 7$$

where the value of x is changed to the result of the expression $y + 7$.

Input and output are built into OCCAM at the most basic level. They send and receive messages on channels that connect parallel processes. Channels are the only way that two processes running in parallel can share information because sharing of global variables would lead to unpredictable behavior. Since parallel processes execute asynchronously, a process needing to access such a variable would have no way of knowing when that variable was last updated and whether or not it contained the desired information. To simplify the situation, OCCAM allows only point-to-point one way communication between processes. Before they can be used, channels must be declared as a certain type, i.e. INT for integer, and given a name. This may seem restrictive, but protocol definitions can be substituted for the type allowing a virtually unlimited number of combinations, and the benefit is highly secure communications. If *channel*1 and *channel*2 connect two processes in parallel, then

$$channel1 \; ? \; x$$

would read the value from *channel*1 into the variable x in the current process, and

$$channel2 \; ! \; y + 5$$

would send the value of $y+5$ down *channel*2 to the other process where a corresponding input command would read it into a variable local to that process. The symbols ? and ! are the commands for inputting and outputting on a channel respectively.

4.2.2 Constructions

Compound processes are built by combining primitive processes within constructions. Various constructions supported by OCCAM are illustrated by example below. In a language designed for concurrency, sequential operation can not be assumed as it is in sequential languages. In OCCAM sequential operation must be explicitly stated with the SEQ command.

```
SEQ
  proc1
  proc2
  proc3
```

The processes, proc1, proc2, and proc3, are then executed one after the other in the order listed. Termination of SEQ process occurs when the last process, in this case proc3, terminates. The scope of a construction is marked by an indentation of two spaces. In OCCAM, indentation is used for grouping commands in the same way that BEGIN...END and {...} are used in PASCAL and C respectively. The command to stipulate parallel execution is PAR.

```
PAR
  proc1
  proc2
  proc3
```

This time proc1, proc2, and proc3 will be executed concurrently with no regard for the order in which they are listed. Termination of a PAR process occurs when the slowest process within its scope terminates.

For cases where a process may need to input from one of several channels and perform a task based on which of the channels is communicating, OCCAM provides a command for alternation.

```
ALT
  chan1 ? x
    proc1
  chan2 ? x
    proc2
  chan3 ? x
    proc3
```

In the example, one of the three processes is executed depending on which of the channels first becomes ready to communicate. The input statement in an alternation, called a 'guard', can consist of an input process and Boolean expressions to further define which of two simultaneously ready channels will be accepted.

SEQ, PAR, and ALT statements can be "replicated" to form arrays of processes. The simplest example is a replicated SEQ which is roughly equivalent to a FOR statement in most conventional languages.

```
SEQ i = 0 FOR n
  proc1
  proc2
  proc3
```

The remaining flow control constructs are similar to those found in conventional languages. IF and CASE statements provide conditional branching and a WHILE statement provides conditional looping. For examples of OCCAM code, see Appendix B.

4.3 Real Time Issues

From their beginnings, transputers were targeted for embedded systems applications. As a result, they have many features that make them attractive as real time processors. Most of the essential functions for simulated and true concurrency, such as multitasking and message passing are strongly supported in hardware for maximum efficiency. The process scheduler also supports two priority levels (high and

low) for processes. The level of the priority can be specified in the OCCAM code. A high priority process will run to the exclusion of all other processes until it can no longer proceed or terminates. Low priority processes can be interrupted by high priority processes and are serviced in a round robin fashion. A system of priorities allows short but time-critical processes to be executed without interruption which is often a requirement in real time systems.

A hardware timer gives programs access to a 1 microsecond clock for high priority processes and a 64 microsecond clock for low priority processes. Considering the time dependent nature of real time programming, hardware timing support is a significant asset. It facilitates the scheduling of time critical events and clocking of benchmarks. In OCCAM, timers are accessed through a special variable type called a timer.

In a real time system, there is often a need to communicate with external devices. Interfacing of peripherals can be handled by one of four methods.

1. The external memory bus.

2. Dual-ported memory.

3. An interrupt.

4. Links.

The first two options are not in keeping with the transputers design philosophy, but they are possible for special purposes. One interrupt pin, or event pin, is provided on T800 and earlier transputers that can be used to trigger interrupt service routines. Turnaround times for servicing interrupts are under 1 microsecond. The favored method of handling any form of communications on the transputer, however, is through the links which can also be interfaced with parallel ports by the use of external link adapters.

One of the stated advantages of OCCAM is that it allows code to be developed independent of the hardware on which it will eventually run. At some point, however,

processes must be loaded and run on physical transputer nodes, and channels must be placed on their corresponding links. Resource allocation in OCCAM is handled by the commands PLACE and PLACED which are used to assign channels and processes to their respective physical devices. In addition, OCCAM allows insertion of low level T-code into OCCAM programs. Fortunately, the close relationship between OCCAM and the transputer hardware usually eliminates the need to resort to low level programming.

CHAPTER 5

CONCURRENT PROCESSING IMPLEMENTATION

5.1 Algorithms

Some of the algorithms listed in the suites from section 2.2 have been translated into OCCAM for implementation on the transputers. In certain cases special techniques have been used to speed up the algorithms taking advantage of the nature of the problem, the transputer, or OCCAM. In the following sections, these algorithms will be described along with any details of their implementation. Appendix B provides OCCAM listings of all the algorithms described below.

5.1.1 Pulse-Pair Estimator

The pulse-pair algorithm has been introduced previously as an estimator of the windspeed mean and variance. The technique is based on estimating the complex autocovariance of the I&Q return from the pulse Doppler radar. If successive pairs of pulses are statistically independent, then it has been shown [32] that a maximum likelihood estimator of the complex autocovariance, $\hat{R}_{ZZ}(T_s)$, is given by

$$\hat{R}_{ZZ}(T_s) = \frac{1}{M} \sum_{i=0}^{M-1} Z^*(iT_s)Z([i+1]T_s),$$

where M is the number of pulses, T_s is the sampling time between pulses (IPP), and $Z(iT_s)$ is the complex I&Q sample at time iT_s. An estimate of the mean windspeed is then given by

$$\hat{v}_{pp} = \frac{\lambda}{4\pi T_s} arg[\hat{R}_{ZZ}(T_s)],$$

and the width estimate by

$$\hat{w}_{pp} = \frac{2}{(2\pi T_s)^2} \left[1 - \frac{|\hat{R}_{ZZ}(T_s)|}{\hat{R}_{ZZ}(0)} \right].$$

These equations describe a method for obtaining mean and width estimates of the windspeed given a complex series of pulse returns in the time domain. An equivalent estimate can be obtained from the spectrum by noting that

$$\hat{R}_{ZZ}(T_s) = \sum_{k=0}^{M-1} S_Z(k) e^{\frac{2\pi k T_s}{M-1}},$$

where $S_Z(k)$ denotes the power spectral density. Then taking the argument of $\hat{R}_{ZZ}(T_s)$ gives

$$arg\, \hat{R}_{ZZ}(T_s) = \arctan \left\{ \frac{\sum_{k=0}^{M-1} S_Z(k) \sin(\frac{2\pi k T_s}{M-1})}{\sum_{k=0}^{M-1} S_Z(k) \cos(\frac{2\pi k T_s}{M-1})} \right\},$$

and estimates of $\hat{v}_{pp}$ and $\hat{w}_{pp}$ can be found as before [33].

Both implementations of the pulse-pair estimator have the advantage of being more computationally efficient than methods based on the FFT. It has also been shown that the pulse-pair estimate has a smaller variance than the FFT estimator for low signal to noise levels [19, 17]. The OCCAM implementation of the pulse-pair estimator is straightforward.

5.1.2 Fast Fourier Transform

A class of algorithms known as FFTs provide a way of calculating the discrete Fourier transform, DFT, that is computationally efficient and works well for a wide range of problems. The algorithm used here is a complex radix-2 decimation in frequency implementation of the FFT translated and modified from a FORTRAN version taken from *IEEE Programs for Digital Signal Processing* [34].

In order to speed up the calculation, various modifications have been made to the basic FFT algorithm. First it was noticed that a large percentage of the computation time (roughly 38%) was spent on calculating the sine and cosine values for the twiddle factors. For fixed data record lengths, a fixed set of these twiddle factors are needed, so they are computed beforehand and stored in memory for later use by the FFT. Techniques for performing the bit reversal are another area of interest for speeding

up the FFT [35, 36]. The implementation of the FFT in OCCAM benefited from an efficient bit reversal function built into the compiler libraries, and no other steps were taken to improve the bit reversal algorithm.

A method for improving the efficiency of the FFT algorithm when there are a large number of zero-valued samples in relation to actual data, is FFT pruning [37]. This technique elimates unnecessary computations that involve zeros by "pruning" off those branches of the computation. The necessary modifications to the algorithm are trivial. It has been shown that the time saved by pruning, tr, is approximately

$$tr = \frac{[L + 2(1 - 2^{-(M-L)})]}{M},$$

where 2^L data points are padded with 2^{M-L} zeros to form 2^M-point FFT [37]. Pruning was left in place even in cases where no zero padding was used since the computational overhead for pruning was found to be extremely small.

The FFT has two primary disadvantages inherent in its approach. The first disadvantage is that frequency resolution is limited by the inverse of the length of the data record. This limits the ability of the FFT to resolve two or more signals closely spaced in frequency. The second disadvantage involves the use of finite length sequences to represent signals of infinite extent. By assuming the sequence to be zero valued outside a finite number of samples, an implied windowing is imposed on the data. It is equivalent to multiplying the data by a rectangular window with unity amplitude. In the spectral domain, that is corresponds to convolving the spectrum with a sinc function. This effect is known as spectral leakage because energy at frequencies not represented in the basis set "leak" into frequencies over the full range of the basis [38, 37]. These limitations of the FFT are especially problematic for short sequences of data which may be the case in a pulse Doppler radar application.

5.1.3 Autoregressive Modeling

An alternative to the FFT for estimating the spectrum is autoregressive (AR) spectral estimation. This technique has been applied previously to radar applications

[39, 40, 41, 42, 43, 44, 45]. The general approach is to fit an AR model to the sampled time series, and from the coefficients of the model generate the spectrum. The 'goodness' of the spectral estimate often depends on how appropriate an AR model is for the underlying process [46, 47].

An AR model is an all pole model of the form

$$H(f) = \frac{1}{1 + \sum_{k=1}^{p} a_k \exp(-j2\pi fkT)},$$

where a_k are the model coefficients, p is the model order, and T is the sampling interval. Once the AR coefficients are determined, the power spectral density can be found as

$$P_{AR}(f) = \frac{T\sigma_p^2}{|1 + \sum_{k=1}^{p} a_k \exp(-j2\pi fkT)|^2},$$

where σ_p^2 is the white noise power. Thus, the parameters that need to be estimated are the a_k's and σ_p^2. Many methods for computing these parameters have been developed, some that rely on estimates of the autocorrelation function and others that are based on a least squares linear prediction approach [48]. The algorithm chosen for this application is of the latter class. It is a block data, as opposed to recursive, modified covariance algorithm that has been translated into OCCAM from a FORTRAN program supplied with Marple's book, *Digital Spectral Analysis with Applications* [48].

Advantages of using the modified covariance method of spectral estimation are an improved frequency resolution compared to the FFT, for low signal to noise ratios, and an ability to estimate the spectrum at any frequency within the processing bandwidth instead of at frequencies predetermined by the length of the data record. Also the problem of spectral leakage is eliminated because the model does not force the sequence to be zero valued outside the range of the data. Disadvantages are that it is more computationally intensive than the FFT, requiring $Np + 6p^2$ operations as opposed to $N\log_2(N)$ for the FFT [48]. Also the AR model may not be well suited to the problem, often resulting in high order values of p to adequately represent it.

Choosing an appropriate model order is not always a straight forward process. Several algorithms have been proposed for this purpose [48, 49, 50, 51], but it has been shown that none of these methods works well for short segments of actual data [52].

5.2 Hardware

The transputer network used to test the above algorithms and obtain benchmarks for processing radar return concurrently is shown in Figure 5.1 as a block diagram. It consists of 9 transputers, eight of which are T414s configured in a 3-D hypercube and the nineth, a T800 root node, serving as a gateway to the PC. The hypercube topology was chosen because it is flexible, in the sense that many other topologies can be mapped on the hypercube, and because it is a popular network that has received considerable attention in parallel processing literature [20]. The most important attribute of a network for this application is that a minimal number of interprocessor communications be necessary to distribute the range cell data. The hypercube satisfies this requirement.

On the PC, Radar I&Q data is stored in a file which is read into the memory of the root node by an interfacing program written in the C language. At this time the user is asked to specify eight range cells to be displayed to the screen. All the data is then distributed by range cell throughout the network by routing programs resident in each node. A spectral estimate is performed in each node, and then the results are recollected in the root node where Doppler spectrums for the eight specified range cells are read directly into the PC memory by the C interfacing program. Finally, a C program resident on the PC host is invoked to display the results on the screen. Figure 5.2 shows a display of some simulated data.

5.3 Benchmarks

Measurement of benchmarks was complicated by the use of two different transputer processor models operating at different clock speeds. The transputer network was made up of one T800 running at 17.5MHz and eight T414s running at 20MHz.

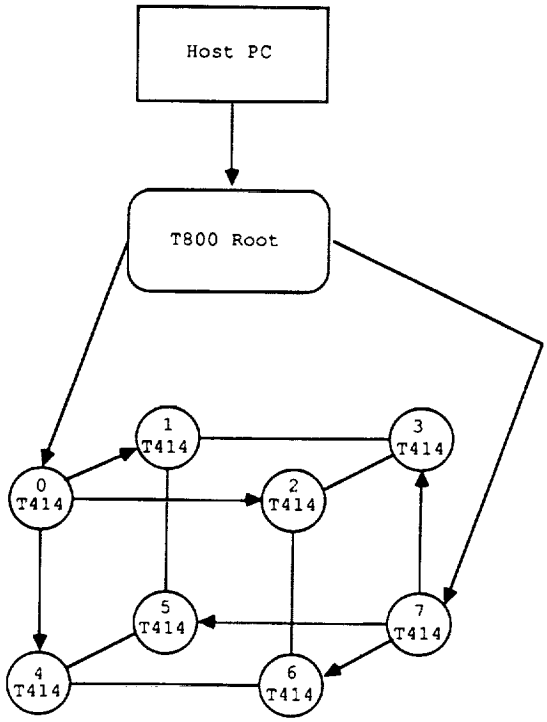

Figure 5.1 Block diagram of the transputer evaluation set-up.

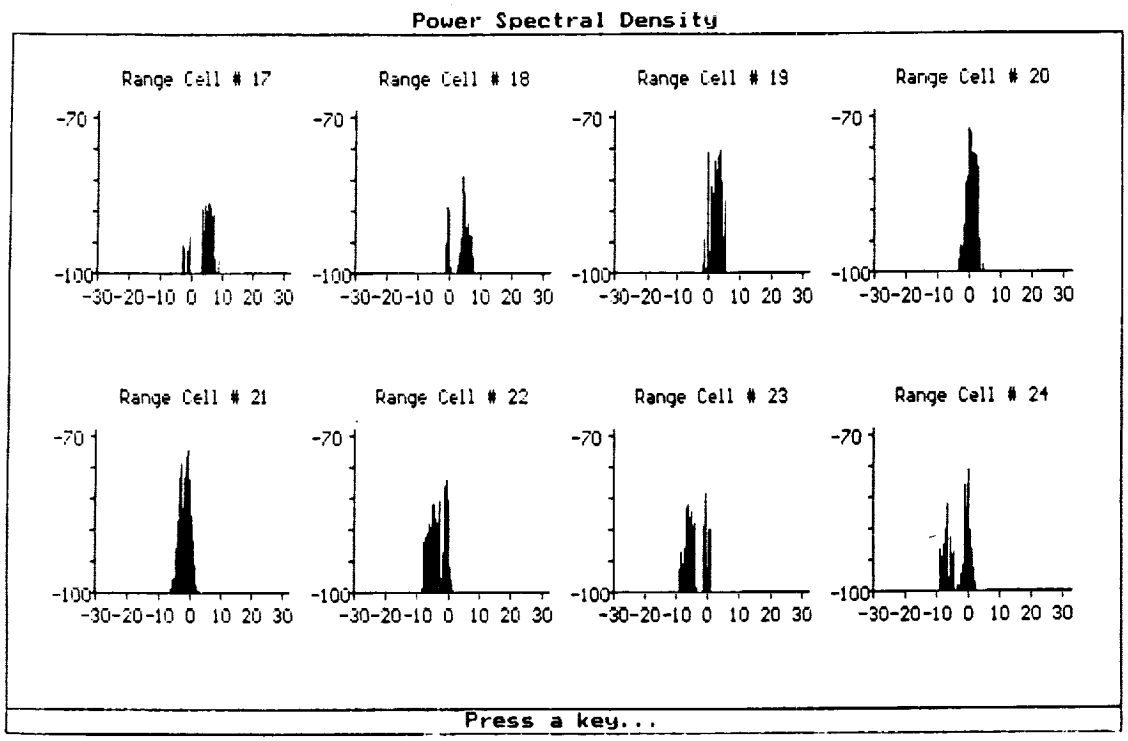

Figure 5.2 Range cell display.

Pulses	Zeros	Total points	T800 @ 17.5MHz Time(msec)	T800 @ 30MHz Time(msec)
512	0	512	78.67	45.89
256	256	512	78.67	45.89
128	384	512	74.43	43.42
64	448	512	68.82	40.15
32	480	512	63.02	36.76

Figure 5.3 FFT benchmarks with pruning.

The primary difference between the T800 and the T414 is the T800's FPU which gives it much better floating point performance than the T414. For this reason, benchmarks for individual algorithms were performed on the T800 and the execution times were scaled to show the expected performance on the fastest available transputer, a T800 running at 30MHz. Measuring speedup, however, required the use of multiple transputers so those measurements were performed on networks of the T414s.

5.3.1 Fast Fourier Transform

Two sets of benchmarks were compiled for the FFT. The first, shown in Figure 5.3, illustrates the effect of pruning. The number of actual data points was varied from 512 down to 32 with zeros added as required to make a total of 512 points. Notice that a large ratio of zeros to data points is necessary before significant time savings are seen. For a 512 point FFT on a 30MHz T800, Figure 5.3 shows that pruning 480 zeros only saved 9.13 msec, or 20% of the total execution time. In fact for a one to one ratio of zeros to data points pruning gives no improvement. It is evident that even with pruning, zero padding incurs a significant computational cost.

Figure 5.4 shows the FFT benchmarks with no zero padding. Recall that the window of time available for processing all the range cells was found in section 2.1 to be 40 ms. Comparing this value with the execution times listed in the Figure 5.4,

Pulses	T800 @ 17.5MHz Time(msec)	T800 @ 30MHz Time(msec)
512	78.67	45.89
256	36.79	21.46
128	16.71	9.75
64	7.74	4.52
32	3.56	2.08

Figure 5.4 FFT benchmarks with no zero padding.

notice that even on the 17.5MHz T800 for 128 or fewer samples the computation can take place within the available processing window. On a 30MHz T800 the number of samples can be extended to 256. The assumptions here, of course, are that a single processor is allocated to each range cell and that the overheads for using such a large number of processors are ignored.

5.3.2 Autoregressive Model

The AR spectral estimate was run for model orders 3, 5, and 10. Order 3 was chosen as a baseline because it has enough poles to account for a clutter mode and weather mode plus an extra pole to provide some additional detail. It would not be unreasonable to use a 2^{nd} order model just to capture those two dominant modes, and it should be noted a 1^{st} order AR model is equivalent to the pulse pair estimator [53]. Model order 5 was chosen as an intermediate level because it seems to adequately represent the Doppler power spectrum of simulated microburst data based on qualitative comparison with the FFT, and order 10 was chosen as a growth case based on similar qualitative grounds and previous analysis of simulated clutter [15]. Figure 5.5 shows the benchmarks for all three model orders for range cell records of 32 to 512 pulses. On a 30MHz T-800 transputer, notice that the 3^{rd} and 5^{th} order models can accommodate up to 256 pulses within the allowable processing window and the 10^{th} order model can accommodate up to 128 pulses.

Model order	Pulses	T800 @ 17.5MHz Time(msec)	T800 @ 30MHz Time(msec)
3	512	84.14	49.08
3	256	42.94	25.05
3	128	22.01	12.84
3	64	11.86	6.92
3	32	3.56	2.08
5	512	110.83	64.65
5	256	57.65	33.63
5	128	30.56	17.83
5	64	17.43	10.17
5	32	10.81	6.31
10	512	173.16	101.01
10	256	95.55	55.74
10	128	54.79	31.96
10	64	35.13	20.49
10	32	25.18	14.69

Figure 5.5 Autoregressive modeling benchmarks.

Processors (N)	S(N)	E(N)
1	1.000	1.000
2	1.988	0.994
4	3.877	0.969
8	7.552	0.944

Figure 5.6 Speedup and efficiency measurements.

5.3.3 Speedup

The speedup was measured on networks of 1, 2 ,4, and 8 processors connected in hypercubes of dimension 0, 1, 2, and 3 respectively. Figure 5.6 shows the speedup and efficiency measured for all four networks. Notice that the speedup is nearly linear for up to 8 processors with an efficiency at 8 processors of 0.944. If the speedup were perfectly linear the efficiency would be a constant of 1. Figure 5.7 plots the speedup measurements from Figure 5.6 versus the number of processesors. In the plot, the solid line corresponds to linear speedup and the dashed line is the actual measured speedup. This figure shows that for a low number of processors, the speedup is nearly linear. Recall from section 3.2 that the bounds on speedup for this application were found to be

$$\frac{NA}{N+A-1} \leq S(N) \leq \min(N, A).$$

For a network of $N = 8$ processors and $A = N_R = 40$ range cells, this equation gives a speedup bound of $6.81 \leq S(N) \leq 8.00$ which agrees with the measured value.

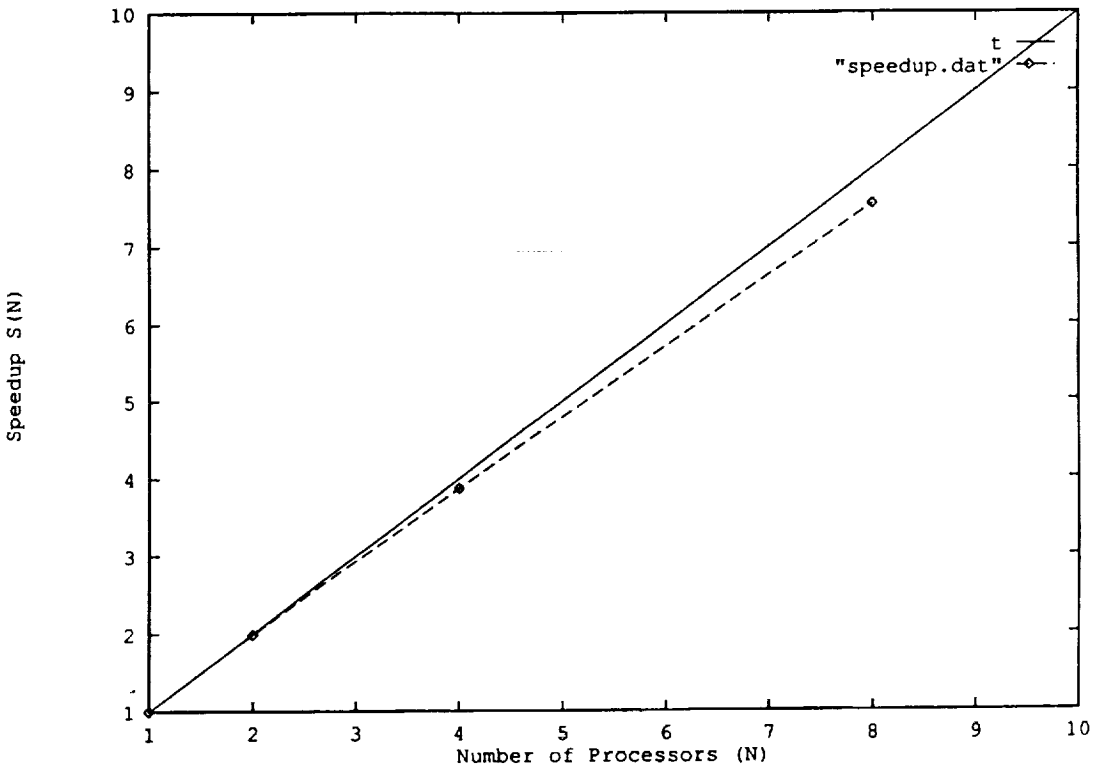

Figure 5.7 Plot of speedup.

CHAPTER 6

CONCLUSIONS

Regardless of the type of sensor eventually used to provide airborne microburst detection, the signals generated will have to be processed in real time. In order to assure reliable detection under all atmospheric conditions, it is likely that a combination of sensors and sophisticated signal processing techniques will be required. A promising approach to achieving the high data rates necessary for real time processing is to perform the signal processing concurrently on an array of multiprocessors. This thesis has examined issues involved in implementing a concurrent real time microburst detection system using a PC based parallel computer called the transputer.

The specific strategy investigated here is to distribute the processing of range cells over a network of single instruction stream computers that pass messages on high speed serial communication links. The processing within a range cell is entirely sequential so that communication overheads are minimized. Also, existing algorithms can be used with only slight modification. With regard to network topology, a hypercube arrangement is adopted because it allows the minimum number of link communications to distribute the range cell data throughout the network. The hypercube also has the benefit of having been well studied and documented in previous works [20].

A number of algorithms are implemented to demonstrate that real time processing is feasible. This effort focuses on the most computationally intensive step in the signal processing, estimating the spectrum. Two methods are considered, the FFT and AR modeling. The FFT is a well known method for estimating the spectrum that is computationally efficient and robust. To improve the algorithms suitability for real time processing, look-up tables are used to calculate the twiddle factors and pruning

is used to eliminate needless computations due to zero padding of the data. Drawbacks of the FFT are the limited frequency resolution and spectral leakage. Both of these limitations are most severe for short sequences of data as may be the case for range cell records based on the stationarity of the weather spectrum. Spectral estimates based on AR modeling do not suffer from spectral leakage and the frequency resolution is not as limited; however, AR modeling is much more computationally intensive than the FFT. It is also not clear that the most appropriate model for the weather plus clutter time series is autoregressive, or what the optimum model order is. Analysis of simulated data has indicated that a 5^{th} order AR model might adequately represent the spectrum and it may be possible to go as low as 2^{nd} order to capture just the two modes associated with the clutter and the weather [54].

Benchmarks of the FFT and AR modeling algorithms show that the returns from a single range cell could be processed in real time on a single transputer. Unfortunately that means that a network of transputers with a number of nodes equal to the number of range cells would be required to process the complete return signal. From measurements of the speedup, it is apparent that overheads associated with concurrency and especially communication overhead will begin to significantly degrade performance for large numbers of processors so that even allowing one processor for each range cell would probobly not be sufficient for real time. This analysis is based on technology that is currently available, but an inherent property of concurrent processing is that a small improvement in the performance of the individual nodes can lead to vast improvements in the overall system performance. In 1992 a new generation of transputer called the T9000 will be available that promises an order of magnitude increase in performance over previous models. Using T9000 transputers, a system such as the hypercube arrangement described in section 5.2 would meet real time requirements.

The transputer evaluation system provides a parallel processing platform for conducting a number of research efforts including development and benchmarking

algorithms and measurements of concurrent performance in anticipation of advances in computer hardware. In addition, it can serve as a quick-look processor for identifying interesting sections of data during post-processing of flight test data.

Future work in this area may include investigating the use of medium grain SIMD computers to perform the necessary radar signal processing since a single instruction stream could be used to perform identical computations on each range cell. Another approach might be to distribute the range cell computation over multiple processors for implementation on massively parallel networks. It may also be possible to integrate some of the radar control functions such as automatic gain control (AGC) on the same parallel processor that performs the other signal processing tasks. Finally, application of sensor fusion and artificial intelligence principles to microburst detection may be called for if a combination of sensor technologies is necessary to provide reliable detection.

APPENDICES

Appendix A

Radar Parameters

Parameter	Value
Frequency	9.33625 GHz 9.33772 GHz
Pulse width	0.96 to 8.16 μsec
Power	200 or 2000 Watts
Pulse Repetition Frequency	1198, 2395, 3755, 4791, or 9581 pulses/sec
Antenna Gain	34 dBi
Two-way Antenna Beamwidth	2.5°
Antenna Scan Width	0, ±5°, ±10°, ±15°, ±20°, ±30°, ±45°, or ±60°
Antenna Scan Center	±90°, adjustable in .25° increments
Antenna Tilt Range	−32° to +40°
Azimuth Scan Rate: All PRFs except 3755 At PRF of 3755	 37.5, 18.75, 9.375, or 4.6875 °/sec 29.25, 14.625 °/sec
Antenna Polarization	Horizontal or Vertical
Data Encoding	12-bit I&Q samples
Number of Range Cells	1 to 124

Figure A.1 Radar Parameters.

Appendix B

Occam Software Listing

B.1 Pulse-Pair

```
--{{{  pulse pair
#INCLUDE "ws.inc"

PROC PP(CHAN OF C from.pc, CHAN OF PPP to.pc)

  #USE "cmplxlib"
  #USE "snglmath.lib"

  VAL N IS pulses:
  VAL prf IS 3720.0(REAL32):
  VAL prp IS 1.0(REAL32)/prf:
  VAL pi IS 3.141592654(REAL32):
  VAL tick.rate IS 1000000.0(REAL32) :
  VAL amult IS 1.0(REAL32)/((((2.0(REAL32)*pi)*pi)*prp)*prp):

  REAL32 argrts,amagnrts,sum,real.sum,aimag.sum,bmult,tr,ti,r0,time:
  REAL32 eofmean,eofstd :
  [N+1]REAL32 areal,aimag :
  INT I,t1,t2:
  TIMER clock :

  SEQ
    real.sum:=0.0(REAL32)
    aimag.sum:=0.0(REAL32)
    I:=1
    from.pc ? [areal FROM 1 FOR N]; [aimag FROM 1 FOR N]
    clock ? t1
    WHILE I<N
      SEQ
        CMul(areal[I],-aimag[I],areal[I+1],aimag[I+1],tr,ti)
        real.sum:=real.sum+tr
        aimag.sum:=aimag.sum+ti
        I:=I+1
    amagnrts:=SQRT((real.sum*real.sum)+(aimag.sum*aimag.sum))/
              (REAL32 ROUND(N-1))
    argrts:= ATAN2(real.sum,aimag.sum)
    sum:=0.0(REAL32)
    SEQ I=1 FOR N
      sum:=sum + ((areal[I]*areal[I])+(aimag[I]*aimag[I]))
```

```
    r0:=sum/(REAL32 ROUND(N))
    eofmean:=(30.02(REAL32)*argrts)/pi
    bmult:=ABS(1.0(REAL32)-(ABS(amagnrts)/r0))
    eofstd:=((30.02(REAL32)*SQRT(amult*bmult))*2.0(REAL32))/prf
    clock ? t2
    time := (REAL32 ROUND (t2 MINUS t1))/tick.rate
    to.pc ! time; eofmean; eofstd
:
}
```

B.2 Fast Fourier Transform

```
#INCLUDE "hostio.inc"
#INCLUDE "ws.inc"

--{{{  FUN PWR2
INT FUNCTION PWR2 (VAL INT i)
  INT b :
  VALOF
    SEQ
      b:=1
      b:=ASHIFTLEFT(b,i)
    RESULT b
:

PROC INDEX.TRIG(VAL INT J,I,REAL32 C,S,VAL [(pulses/2)+1]REAL32
                cos.table, sin.table)

  INT index:
  SEQ
    index:=(J-1)*PWR2(I-1)
    C:=cos.table[index]
    S:=sin.table[index]
:

--{{{  PROC FFT
PROC FFT(CHAN OF C from.pc, CHAN OF AR to.pc)

  #USE "snglmath.lib"

  VAL twopi IS 6.283185307179586(REAL32) :
  VAL tick.rate IS 1000000.0(REAL32) :
  VAL INT N IS pulses:
  VAL REAL32 NN IS (REAL32 ROUND N):
  REAL32 time:
  [(N/2)+1]REAL32 cos.table,sin.table :
  [N+1]REAL32 X,Y :
```

```
--{{{  Cooley-Tukey Radix-2
TIMER clock:
INT I,N2,t1,t2:
SEQ
  SEQ i=0 FOR ((N/2)+1)
    REAL32 r:
    SEQ
      r:=(((REAL32 ROUND i)*twopi)/NN)
      cos.table[i]:=COS(r)
      sin.table[i]:=SIN(r)
  WHILE TRUE
    SEQ
      N2:=N
      I:=1
      from.pc ? [X FROM 1 FOR N];[Y FROM 1 FOR N]
      clock ? t1
      WHILE I<=M
        INT J,N1,LL:
        SEQ
          N1 := N2
          N2 := N2/2
          LL := N2
          IF
            I < (M-LO)
              LL := L2
            I >= (M-LO)
              SKIP
          J:=1
          WHILE J<=N2
            REAL32 C,S:
            INT K:
            SEQ
              INDEX.TRIG(J,I,C,S,cos.table,sin.table)
              K:=J
              WHILE K <= N
                REAL32 XT,YT:
                INT L:
                SEQ
                  L:=K+N2
                  XT:=X[K]-X[L]
                  YT:=Y[K]-Y[L]
                  X[K]:=X[K]+X[L]
                  Y[K]:=Y[K]+Y[L]
                  X[L]:=(C*XT)+(S*YT)
                  Y[L]:=(C*YT)-(S*XT)
                  K:=K+N1
              J:=J+1
```

```
            I:=I+1

          VAL REAL32 c IS 60.0(REAL32)/NN :
          [N]REAL32 ws,z,xtemp,ytemp :
          INT p :
          SEQ
            [xtemp FROM 0 FOR N] := [X FROM 1 FOR N]
            [ytemp FROM 0 FOR N] := [Y FROM 1 FOR N]
            SEQ I=0 FOR N
              SEQ
                p:=BITREVNBITS(I,M)
                z[I]:=10.0(REAL32)*ALOG10((xtemp[p]*xtemp[p])+
                      (ytemp[p]*ytemp[p]))
        ws[I]:=((REAL32 ROUND I) - (REAL32 ROUND (N/2)))*c
            [xtemp FROM 0 FOR N/2] := [z FROM N/2 FOR N/2]
            [xtemp FROM N/2 FOR N/2] := [z FROM 0 FOR N/2]
            clock ? t2
            time := (REAL32 ROUND (t2 MINUS t1))/tick.rate
            to.pc ! time; ws; xtemp
:
```

B.3 Autoregressive Model

```
#INCLUDE "ws.inc"

--{{{  PROC FFT
PROC FFT(VAL [(pulses/2)+1]REAL32 cos.table, sin.table,
         [pulses+1]REAL32 S,f, [ORD+2]REAL32 a.r,a.i,REAL32 sigma)

  #USE "snglmath.lib"

  VAL INT N IS pulses :

  [N+1]REAL32 R,Q:
  INT I,N2:
  --{{{  Cooley-Tukey Radix-2
  SEQ
    R[1]:=1.0(REAL32)
    Q[1]:=0.0(REAL32)
    SEQ G=2 FOR ORD
      SEQ
        R[G]:=a.r[G-1]
        Q[G]:=a.i[G-1]
    SEQ G= (ORD+2) FOR (N-(ORD+1))
      SEQ
        R[G]:=0.0(REAL32)
        Q[G]:=0.0(REAL32)
```

```
N2:=N
I:=1
WHILE I<=M
  INT J,N1,LL:
  SEQ
    N1 := N2
    N2 := N2/2
    LL := N2
    IF
      I < (M-LO)
        LL := L2
      I >= (M-LO)
        SKIP
    J:=1
    WHILE J<=LL
      REAL32 C,S:
      INT K:
      SEQ
        INDEX.TRIG(J,I,C,S,cos.table,sin.table)
        K:=J
        WHILE K <= N
          REAL32 XT,YT:
          X IS [R FROM 0 FOR (N+1)]:
          Y IS [Q FROM 0 FOR (N+1)]:
          INT L:
          SEQ
            L:=K+N2
            XT:=X[K]-X[L]
            YT:=Y[K]-Y[L]
            X[K]:=X[K]+X[L]
            Y[K]:=Y[K]+Y[L]
            X[L]:=(C*XT)+(S*YT)
            Y[L]:=(C*YT)-(S*XT)
            K:=K+N1
        J:=J+1
    I:=I+1

-- AR Spectral extimate
VAL REAL32 NN IS (REAL32 ROUND N) :
[N]REAL32 temp :
INT p :
SEQ
  SEQ I=1 FOR N
    SEQ
      S[I-1]:=sigma/((R[I]*R[I])+(Q[I]*Q[I]))
  SEQ I=0 FOR N
    SEQ
      p:=BITREVNBITS(I,M)
```

```
          temp[I]:=10.0(REAL32)*ALOG10(S[p])
          f[I]:=(((REAL32 ROUND I)-(NN/2.0(REAL32)))*60.0(REAL32))/NN
        [S FROM 0 FOR N/2] := [temp FROM N/2 FOR N/2]
        [S FROM N/2 FOR N/2] := [temp FROM 0 FOR N/2]
:

PROC MODCOVAR(CHAN OF C from.pc,CHAN OF AR to.pc)

  #USE "snglmath.lib"
  #USE "cmplxlib"

  VAL N IS pulses :
  VAL twopi IS 6.283185307179586(REAL32) :
  VAL tick.rate IS 1000000.0(REAL32) :
  VAL REAL32 NN IS (REAL32 ROUND N) :

  INT M,t1,t2 :
  REAL32 r1,r2,r3,r4,P,delta,gamma,lambda.r,lambda.i,tmp.r,tmp.i :
  [N+1]REAL32 x.r,x.i :
  [ORD+2]REAL32 a.r,a.i,c.r,c.i,d.r,d.i :
  [(N/2)+1]REAL32 cos.table,sin.table :
  TIMER clock :

  SEQ
    r1 := 0.0(REAL32)
    M := 0
    SEQ I=0 FOR ((N/2)+1)
      REAL32 r :
      SEQ
        r:=(((REAL32 ROUND I)*twopi)/NN)
        cos.table[I] := COS(r)
        sin.table[I] := SIN(r)
    SEQ I=1 FOR (ORD+1)
      SEQ
        a.r[I] := 0.0(REAL32)
        a.i[I] := 0.0(REAL32)
    from.pc ? [x.r FROM 1 FOR N];[x.i FROM 1 FOR N]
    clock ? t1
    SEQ I=2 FOR (N-2)
      r1 := r1+(2.0(REAL32)*CMag2(x.r[I],x.i[I]))
    r2 := CMag2(x.r[1],x.i[1])
    r3 := CMag2(x.r[N],x.i[N])
    r4 := 1.0(REAL32)/(r1+(2.0(REAL32)*(r2+r3)))
    P := r1+(r2+r3)
    delta := 1.0(REAL32)-(r2*r4)
    gamma := 1.0(REAL32)-(r3*r4)
    CMul(x.r[1],x.i[1],x.r[N],x.i[N],tmp.r,tmp.i)
    CSmul(tmp.r,-tmp.i,r4,lambda.r,lambda.i)
```

```
CSmul(x.r[N],x.i[N],r4,c.r[1],c.i[1])
CSmul(x.r[1],-x.i[1],r4,d.r[1],d.i[1])
--
--                    Main Loop
--
WHILE M<ORD
  REAL32 save1.r,save1.i,theta.r,theta.i,PSI.r,PSI.i,dummy:
  REAL32 XI.r,XI.i,EF.r,EF.i,EB.r,EB.i,r5,tmp1.r,tmp1.i :
  REAL32 C1.r,C1.i,C2.r,C2.i,C3.r,C3.i,C4.r,C4.i :
  [ORD+1]REAL32 R.r,R.i :
  INT J :
  SEQ
    M := M+1
    save1.r := 0.0(REAL32)
    save1.i := 0.0(REAL32)
    SEQ I=(M+1) FOR (N-M)
      SEQ
        CMul(x.r[I],x.i[I],x.r[I-M],-x.i[I-M],tmp1.r,tmp1.i)
        save1.r := save1.r+tmp1.r
        save1.i := save1.i+tmp1.i
    save1.r := 2.0(REAL32)*save1.r
    save1.i := 2.0(REAL32)*save1.i
    R.r[M] := save1.r
    R.i[M] := -save1.i
    CMul(x.r[N],x.i[N],d.r[1],d.i[1],theta.r,theta.i)
    CMul(x.r[N],x.i[N],c.r[1],c.i[1],PSI.r,PSI.i)
    CMul(x.r[1],-x.i[1],d.r[1],d.i[1],XI.r,XI.i)
    IF
      M<>1
        SEQ I=1 FOR (M-1)
          REAL32 tmp.r,tmp.i,tmp1.r,tmp1.i :
          SEQ
            CMul(x.r[N-I],x.i[N-I],d.r[I+1],d.i[I+1],tmp.r,tmp.i)
            theta.r := theta.r+tmp.r
            theta.i := theta.i+tmp.i
            CMul(x.r[N-I],x.i[N-I],c.r[I+1],c.i[I+1],tmp.r,tmp.i)
            PSI.r := PSI.r+tmp.r
            PSI.i := PSI.i+tmp.i
            CMul(x.r[I+1],-x.i[I+1],d.r[I+1],d.i[I+1],tmp.r,tmp.i)
            XI.r := XI.r+tmp.r
            XI.i := XI.i+tmp.i
            CMul(x.r[N+(1-M)],x.i[N+(1-M)],x.r[N+((1-M)+I)],
                 -x.i[N+((1-M)+I)],tmp.r,tmp.i)
            CMul(x.r[M-I],x.i[M-I],x.r[M],-x.i[M],tmp1.r,tmp1.i)
            R.r[I] := R.r[I]-(tmp.r+tmp1.r)
            R.i[I] := R.i[I]-(tmp.i+tmp1.i)
            CMul(R.r[I],-R.i[I],a.r[M-I],a.i[M-I],tmp.r,tmp.i)
            save1.r := save1.r+tmp.r
```

```
        save1.i := save1.i+tmp.i
  M = 1
    SKIP
--
--          Order update of coefficient (a) vector
--
CSmul(-save1.r,-save1.i,1.0(REAL32)/P,C1.r,C1.i)
a.r[M] := C1.r
a.i[M] := C1.i
P := P*(1.0(REAL32)-CMag2(C1.r,C1.i))
IF
  M <> 1
    INT MI :
    SEQ I=1 FOR (M/2)
      REAL32 tmp.r,tmp.i:
      SEQ
        MI := M-I
        save1.r := a.r[I]
        save1.i := a.i[I]
        CMul(C1.r,C1.i,a.r[MI],-a.i[MI],tmp.r,tmp.i)
        CAdd(save1.r,save1.i,tmp.r,tmp.i,a.r[I],a.i[I])
        IF
          I <> MI
            SEQ
              CMul(C1.r,C1.i,save1.r,-save1.i,tmp.r,tmp.i)
              a.r[MI] := a.r[MI]+tmp.r
              a.i[MI] := a.i[MI]+tmp.i
          I = MI
            SKIP
  M = 1
    SKIP
IF
  M = ORD
    REAL32 time :
    [N+1]REAL32 S,f:
    SEQ
      P := 0.5(REAL32)*(P/(REAL32 ROUND (N-M)))
      FFT(cos.table,sin.table,S,f,a.r,a.i,P)
      clock ? t2
      time := (REAL32 ROUND (t2 MINUS t1))/tick.rate
      to.pc ! time;[f FROM 0 FOR N];[S FROM 0 FOR N]
--
-- Time update of C,D vectors and gamma,delta,lambda scalars
--
M <> ORD
  SEQ
    r1 := 1.0(REAL32)/((delta*gamma)-
          CMag2(lambda.r,lambda.i))
```

```
CMul(theta.r,theta.i,lambda.r,-lambda.i,tmp.r,tmp.i)
CAdd(tmp.r,tmp.i,(delta*PSI.r),(delta*PSI.i),
     tmp1.r,tmp1.i)
CSmul(tmp1.r,tmp1.i,r1,C1.r,C1.i)
CMul(PSI.r,PSI.i,lambda.r,lambda.i,tmp.r,tmp.i)
CAdd(tmp.r,tmp.i,(gamma*theta.r),(gamma*theta.i),
     tmp1.r,tmp1.i)
CSmul(tmp1.r,tmp1.i,r1,C2.r,C2.i)
CMul(XI.r,XI.i,lambda.r,-lambda.i,tmp.r,tmp.i)
CAdd(tmp.r,tmp.i,(delta*theta.r),(delta*theta.i),
     tmp1.r,tmp1.i)
CSmul(tmp1.r,tmp1.i,r1,C3.r,C3.i)
CMul(theta.r,theta.i,lambda.r,lambda.i,tmp.r,tmp.i)
CAdd(tmp.r,tmp.i,(gamma*XI.r),(gamma*XI.i),
     tmp1.r,tmp1.i)
CSmul(tmp1.r,tmp1.i,r1,C4.r,C4.i)
SEQ I=1 FOR ((M-1)/3)
  INT MI :
  REAL32 save2.r,save2.i,save3.r,save3.i :
  REAL32 save4.r,save4.i,tmp.r,tmp.i,tmp1.r,tmp1.i :
  SEQ
    MI := (M+1)-I
    save1.r := c.r[I]
    save1.i := -c.i[I]
    save2.r := d.r[I]
    save2.i := -d.i[I]
    save3.r := c.r[MI]
    save3.i := -c.i[MI]
    save4.r := d.r[MI]
    save4.i := -d.i[MI]
    CMul(C1.r,C1.i,save3.r,save3.i,tmp.r,tmp.i)
    CMul(C2.r,C2.i,save4.r,save4.i,tmp1.r,tmp1.i)
    c.r[I] := c.r[I]+(tmp.r+tmp1.r)
    c.i[I] := c.i[I]+(tmp.i+tmp1.i)
    CMul(C3.r,C3.i,save3.r,save3.i,tmp.r,tmp.i)
    CMul(C4.r,C4.i,save4.r,save4.i,tmp1.r,tmp1.i)
    d.r[I] := d.r[I]+(tmp.r+tmp1.r)
    d.i[I] := d.i[I]+(tmp.i+tmp1.i)
    IF
      I <> MI
        SEQ
          CMul(C1.r,C1.i,save1.r,save1.i,tmp.r,tmp.i)
          CMul(C2.r,C2.i,save2.r,save2.i,tmp1.r,tmp1.i)
          c.r[MI] := c.r[MI]+(tmp.r+tmp1.r)
          c.i[MI] := c.i[MI]+(tmp.i+tmp1.i)
          CMul(C3.r,C3.i,save1.r,save1.i,tmp.r,tmp.i)
          CMul(C4.r,C4.i,save2.r,save2.i,tmp1.r,tmp1.i)
          d.r[MI] := d.r[MI]+(tmp.r+tmp1.r)
```

```
          d.i[MI] := d.i[MI]+(tmp.i+tmp1.i)
        I = MI
          SKIP
r2 := CMag2(PSI.r,PSI.i)
r3 := CMag2(theta.r,theta.i)
r4 := CMag2(XI.r,XI.i)
CMul(PSI.r,PSI.i,lambda.r,lambda.i,tmp.r,tmp.i)
CMul(tmp.r,tmp.i,theta.r,-theta.i,tmp1.r,tmp1.i)
r5 := gamma - (r1*((r2*delta)+((r3*gamma)+(2.0(REAL32)*
      tmp1.r))))
CMul(theta.r,theta.i,lambda.r,lambda.i,tmp.r,tmp.i)
CMul(tmp.r,tmp.i,XI.r,-XI.i,tmp1.r,tmp1.i)
r2 := delta - (r1*((r3*delta)+((r4*gamma)+(2.0(REAL32)*
      tmp1.r))))
gamma := r5
delta := r2
CMul(C3.r,C3.i,PSI.r,-PSI.i,tmp.r,tmp.i)
CMul(C4.r,C4.i,theta.r,-theta.i,tmp1.r,tmp1.i)
lambda.r := lambda.r+(tmp.r+tmp1.r)
lambda.i := lambda.i+(tmp.i+tmp1.i)
IF
  P<0.0(REAL32)
    STOP
  P>0.0(REAL32)
    SKIP
IF
  (delta>0.0(REAL32)) AND (delta<1.0(REAL32)) AND
  (gamma>0.0(REAL32)) AND (gamma<1.0(REAL32))
    SKIP
  TRUE
    STOP
--
--  Time update of "a" vector; order updates of c,d
--  vectors and gamma, delta, lambda scalars.
--
r1 := 1.0(REAL32)/P
r2 := 1.0(REAL32)/((delta*gamma)-
      CMag2(lambda.r,lambda.i))
EF.r := x.r[M+1]
EF.i := x.i[M+1]
EB.r := x.r[N-M]
EB.i := x.i[N-M]
SEQ I=1 FOR M
  SEQ
    CMul(a.r[I],a.i[I],x.r[(M+1)-I],x.i[(M+1)-I],
         tmp.r,tmp.i)
    EF.r := EF.r+tmp.r
    EF.i := EF.i+tmp.i
```

```
    CMul(a.r[I],-a.i[I],x.r[(N-M)+I],x.i[(N-M)+I],
         tmp.r,tmp.i)
    EB.r := EB.r+tmp.r
    EB.i := EB.i+tmp.i
    CSmul(EB.r,EB.i,r1,C1.r,C1.i)
    CSmul(EF.r,-EF.i,r1,C2.r,C2.i)
    CMul(EF.r,EF.i,lambda.r,lambda.i,tmp.r,tmp.i)
    CAdd(tmp.r,tmp.i,(delta*EB.r),((-delta)*EB.i),
         tmp1.r,tmp1.i)
CSmul(tmp1.r,tmp1.i,r2,C3.r,C3.i)
CMul(EB.r,EB.i,lambda.r,lambda.i,tmp.r,tmp.i)
CAdd(tmp.r,-tmp.i,(gamma*EF.r),(gamma*EF.i),
     tmp1.r,tmp1.i)
CSmul(tmp1.r,tmp1.i,r2,C4.r,C4.i)
J := M
WHILE J>0
  REAL32 tmp.r,tmp.i :
  SEQ
    save1.r := a.r[J]
    save1.i := a.i[J]
    CMul(C3.r,C3.i,c.r[J],c.i[J],tmp.r,tmp.i)
    CAdd(save1.r,save1.i,tmp.r,tmp.i,a.r[J],a.i[J])
    CMul(C4.r,C4.i,d.r[J],d.i[J],tmp.r,tmp.i)
    a.r[J] := a.r[J]+tmp.r
    a.i[J] := a.i[J]+tmp.i
    CMul(C1.r,C1.i,save1.r,save1.i,tmp.r,tmp.i)
    c.r[J+1] := c.r[J]+tmp.r
    c.i[J+1] := c.i[J]+tmp.i
    CMul(C2.r,C2.i,save1.r,save1.i,tmp.r,tmp.i)
    d.r[J+1] := d.r[J]+tmp.r
    d.i[J+1] := d.i[J]+tmp.i
    J := J-1
c.r[1] := C1.r
c.i[1] := C1.i
d.r[1] := C2.r
d.i[1] := C2.i
r3 := CMag2(EB.r,EB.i)
r4 := CMag2(EF.r,EF.i)
CMul(EB.r,EB.i,lambda.r,lambda.i,tmp.r,tmp.i)
CMul(tmp.r,tmp.i,EF.r,EF.i,tmp1.r,tmp1.i)
P := P - (r2*((r3*delta)+((r4*gamma)+
     (2.0(REAL32)*tmp1.r))))
delta := delta-(r4*r1)
gamma := gamma-(r3*r1)
CMul(EF.r,EF.i,EB.r,EB.i,tmp.r,tmp.i)
lambda.r := lambda.r+(r1*tmp.r)
lambda.i := lambda.i-(r1*tmp.i)
IF
```

```
  P<0.0(REAL32)
    STOP
  P>0.0(REAL32)
    SKIP
IF
  (delta>0.0(REAL32)) AND (delta<1.0(REAL32)) AND
  (gamma>0.0(REAL32)) AND (gamma<1.0(REAL32))
    SKIP
  TRUE
    STOP
```

REFERENCES

[1] T. T. Fujita. "The Downburst–Microburst and Macroburst". SMRP Research Paper No. 210, Univ. of Chicago, 1985. (available from NTIS as PB85-148-880).

[2] J. McCarthy and R. Serafin. "The Microburst: Hazard to Aviation". *Weatherwise*, vol. 37, no. 3, 1984, pp. 120–127.

[3] D. Atlas, editor. *"Radar in Meteorology"*. American Meteorological Society, Boston, 1990.

[4] C. L. Britt. "Users Guide for an Airborne Windshear Doppler Radar Simulation (AWDRS) Program". NASA CR–182025, DOT/FAA/RD-91/2, June 1990.

[5] R. L. Bowles and R. Targ. "Windshear Detection and Avoidance: Airborne Systems Perspective". Presented at the 16^{th} Congress of the ICAS, Jerusalem, Israel, August 28 – September 2, 1988.

[6] P. L. Smith, Jr., K. R. Hardy, and K. M. Glover. "Applications of Radar to Meteorological Operations and Research". *Proceedings of the IEEE*, vol. 62, June 1974, pp. 724–745.

[7] R. M. Lhermitte. "Probing of Atmospheric Motion by Airborne Pulse-Doppler Radar Techniques". *J. Appl. Meteor.*, vol. 10, April 1971, pp. 234–246.

[8] L. J. Battan. *Radar Observations of the Atmosphere.* University of Chicago Press, Chicago, 1973.

[9] D. Atlas. "Advances in Radar Meteorology". *Advan. in Geophys.*, vol. 10, 1964, pp. 317–478.

[10] M. I. Skolnik, editor. *Radar Handbook.* McGraw Hill, New York, NY, 1990.

[11] R. J. Doviak and D. S. Zrnić, editors. *Doppler Radar and Weather Observations.* Academic Press, Orlando, Florida, 1984.

[12] E. M. Bracalente, R. W. Jones, and C. L. Britt. "Airborne Doppler Radar Detection of Low Altitude Windshear". Presented at the AIAA Conference on Sensor and Measurement Techniques for Aeronautical Application, Atlanta, Georgia, September 7–9, 1988.

[13] J. E. Evans. "Ground Clutter Cancellation for the NEXRAD System". Project Report ATC–122, Lincoln Laboratory, 1983. (available from NTIS).

[14] B. M. Keel. "Adaptive Clutter Rejection Filters for Airborne Doppler Weather Radar Applied to the Detection of Low-altitude Windshear". NASA CR–186211, December 1989.

[15] B. M. Keel and E. G. Baxa, Jr. "Adaptive Least Square Complex Lattice Clutter Rejection Filters Applied to the Radar Detection of Low Altitude Windshear". In *Proc. Int. Conf. Acoust., Speech, Sig. Proc.*, Albuquerque, April 1990, pp. 1469–1472.

[16] R. H. Daly and J. M. Glass. "Digital Signal Processing for Radar". *Electronic Progress*, vol. 17, no. 1, 1975, pp. 24–30.

[17] J. J. Sitterle. "Estimation of Spectral Moments in Small Wavelength Doppler Weather Radar". Master's thesis, Clemson University, Clemson, SC, August 1983.

[18] W. D. Rummler. "Introduction of a New Estimator for Velocity Spectral Parameters". Tech. Memo. MM-68-4121-5, Bell Telephone Laboratories, Whippany, NJ, 1968.

[19] R. J. Doviak, D. S. Zrnić, and D. S. Sirmans. "Doppler Weather Radar". *Proceedings of the IEEE*, vol. 67, November 1979, pp. 1522–1553.

[20] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors Volume I.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[21] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, Palo Alto, CA, 1989.

[22] "Real-Time Signal Processor Development". Final Report RTI/4500/02-01F, Prepared for NASA, Contract NAS1-18925, Research Triangle Institute, January 1991.

[23] Published by the IEEE, New York, NY. *"IEEE Standard for Binary Floating-Point Arithmetic"*, ANSI/IEEE Std 754-1985, 1985.

[24] W. R. Jones, O. Alitz, P. R. Schaffner, J. H. Schrader, and J. H. C. Blume. "Description, Characteristics, and Testing of the NASA Airborne Radar". *NASA Conference Publication 10060, DOT/FAA/RD-91/2-II*, January 1991, pp. 939–978.

[25] G. M. Amdahl. "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities". *Proc. AFIPS*, 30:483–485, 1967.

[26] D. L. Eager, J. Zahorjan, and E. D. Lazowska. "Speedup Versus Efficiency in Parallel Systems". *IEEE Transactions on Computers*, vol. 38, March 1989, pp. 408–423.

[27] J. L. Gustafson. "Reevaluating Amdahl's Law". *Communications of the ACM*, vol. 31, no. 5, May 1988, pp. 532–533.

[28] T. Feng. "A Survey of Interconnection Networks". *Computer*, vol. 14, no. 12, December 1981, pp. 12–27.

[29] J. Wexler and D. Prior. "Solving Problems with Transputers: Background and Experience". *Microprocessors and Microsystems*, vol. 13, no. 2, March 1989, pp. 67–78.

[30] D. Pountain. "Occam II". *BYTE*, vol. 14, no. 10, October 1989, pp. 279–284.

[31] C.A.R. Hoare. "Communicating Sequential Processes". *Communications of the ACM*, vol. 21, no. 8, August 1978, pp. 666–677.

[32] K. S. Miller and M. M. Rochwarger. "A Covariance Approach to Spectral Moment Estimation". *IEEE Transactions on Information Theory*, vol. 18, 1972, pp. 588–596.

[33] D. S. Zrnić. "Estimation of Spectral Moments for Weather Echoes". *IEEE Trans. Geosci. Electron.*, vol. GE-17, no. 4, October 1979, pp. 113–128.

[34] The Digital Signal Processing Committee, IEEE Acoustics, Speech, and Signal Processing Society, editor. *"Programs for Digital Signal Processing"*. IEEE Press, New York, NY, 1979.

[35] D. M. W. Evans. "An Improved Digit-Reversal Permutation Algorithm for the Fast Fourier and Hartley Transforms". *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, no. 8, August 1987, pp. 1120–1125.

[36] J. S. Walker. "A New Bit Reversal Algorithm". *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-38, no. 8, August 1990, pp. 1472–1473.

[37] F. J. Harris. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform". *Proceedings of the IEEE*, vol. 66, no. 1, January 1978, pp. 51–83.

[38] S. M. Kay and S. L. Marple. "Spectrum Analysis—A Modern Perspective". *Proceedings of the IEEE*, vol. 69, no. 11, November 1981, pp. 1380–1419.

[39] J. Gibson, S. S. Haykin, and S. B. Kesler. "Maximum Entropy (Adaptive) Filtering Applied to Radar Clutter". In *Rec. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 1979, pp. 166–169.

[40] S. S. Haykin, editor. *Nonlinear Methods of Spectral Analysis.* Springer-Verlag, New York, 1979.

[41] F. M. Hsu and A. A. Giordano. "Line Tracking Using Autoregressive Spectral Estimates". *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, December 1977, pp. 510–519.

[42] M. Kaveh and G. R. Cooper. "An Empirical investigation of the Properties of the Autoregressive Spectral Estimator". *IEEE Transactions on Information Theory*, vol. 22, May 1976, pp. 313–323.

[43] S. B. Kesler and S. S. Haykin. "The Maximum Entropy Method Applied to the Spectral Analysis of Radar Clutter". *IEEE Transactions on Information Theory*, vol. 24, March 1978, pp. 269–272.

[44] S. B. Kesler and S. S. Haykin. "Maximum Entropy Estimation of Radar Clutter Spectra". In *Natl. Telecommunications Conf. Rec.*, Birmingham, AL, December 3–6 1978, pp. 18.5.1–18.5.5.

[45] J. H. Sawyers. "Applying the Maximum Entropy Method to Adaptive Digital Filtering". *Conference Record — Twelfth Asilomar Conf. on Circuits, Systems, & Computers*, 78CHI 369–8, IEEE, November 1978, pp. 198–202.

[46] R. Nitzberg. "Spectral Estimation: An Impossibility?". *Proceedings of the IEEE*, vol. 67, March 1979, pp. 437–439.

[47] J. Park. "Spectral Analysis of Time Series Using Periodogram and Maximum Entropy Methods". Master's thesis, Clemson University, Clemson, SC, August 1983.

[48] S. L. Marple. *Digital Spectral Analysis with Applications.* Prentice–Hall, Englewood Cliffs, NJ, 1987.

[49] H. Akaike. "Power Spectrum Estimation through Autoregression Model Fitting". *Ann. Inst. Stat. Math.*, vol. 21, 1969, pp. 407–419.

[50] H. Akaike. "A New Look at the Statistical Model Identification". *IEEE Trans. Autom. Control*, vol. 19, December 1974, pp. 716–723.

[51] E. Parzen. "Some Recent Advances in Time Series Modeling". *IEEE Trans. Autom. Control*, vol. 19, December 1974, pp. 723–730.

[52] T. J. Ulrych and R. W. Clayton. "Time Series Modeling and Maximum Entropy". *Phys. Earth Planet. Inter.*, vol. 12, August 1976, pp. 188–200.

[53] P. R. Mahapatra and D. S. Zrnić. "Practical Algorithms for Mean Velocity Estimation in Doppler Weather Radars Using a Small Number of Samples". *IEEE Trans. Geosci. Electron.*, vol. GE-21, October 1983, pp. 491–501.

[54] E. G. Baxa, Jr. "Signal Processing Techniques for Clutter Filter and Windshear Detection". *NASA Conference Publication 10060, DOT/FAA/RD-91/2-II*, January 1991, pp. 870–886.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1992	Contractor Report

4. TITLE AND SUBTITLE
Real-Time Processing of Radar Return on a Parallel Computer

5. FUNDING NUMBERS
G NGT-50414
WU 505-64-12-02

6. AUTHOR(S)
David D. Aalfs

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Radar Systems Laboratory
Electrical and Computer Engineering Department
Clemson University
Clemson, SC 29634-0915

8. PERFORMING ORGANIZATION REPORT NUMBER
TR 14

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-4456
DOT/FAA/RD-92/20

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Anne I. Mackenzie
Final Report

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 03

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

NASA is working with the FAA to demonstrate the feasibility of pulse Doppler radar as a candidate airborne sensor to detect low altitude windshears. The need to provide the pilot with timely information about possible hazards has motivated a demand for real-time processing of a radar return. This thesis investigates parallel processing as a means of accommodating the high data rates required. A PC based parallel computer, called the transputer, is used to investigate issues in real time concurrent processing of radar signals. A transputer network is made up of an array of single instruction stream processors that can be networked in a variety of ways. They are easily reconfigured and software development is largely independent of the particular network topology. The performance of the transputer is evaluated in light of the computational requirements. A number of algorithms have been implemented on the transputers in OCCAM, a language specially designed for parallel processing. These include signal processing algorithms such as the Fast Fourier Transform (FFT), pulse-pair, and autoregressive modeling, as well as routing software to support concurrency. The most computationally intensive task is estimating the spectrum. Two approaches have been taken on this problem, the first and most conventional of which is to use the FFT. By using table look-ups for the basis function and other optimizing techniques, an algorithm has been developed that is sufficient for real time. The other approach is to model the signal as an autoregressive process and estimate the spectrum based on the model coefficients. This technique is attractive because it does not suffer from the spectral leakage problem inherent in the FFT. Benchmark tests indicate that autoregressive modeling is feasible in real time.

14. SUBJECT TERMS	15. NUMBER OF PAGES
windshear detection, pulsed Doppler radar, real-time processing, parallel processing, transputers, OCCAM language, signal processing requirements and performance	84
	16. PRICE CODE A05

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified		

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std Z39-18
298-102

NASA-Langley, 1992