Space Time Adaptive Processing and Clutter Classification Integration and Evaluation

THESIS

Nathan A. Jensen, Second Lieutenant, USAF

AFIT/GCS/ENG/02M-05

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
26 Mar 02	Final	Mar 01 - Mar 02

Title and Subtitle Space Time Adaptive Processing and Clutter Classification Integration and Evaluation	**Contract Number**
	Grant Number
	Program Element Number
Author(s) 2ndLt Nathan A. Jensen, USAF	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 WPAFB OH 45433-7765	**Performing Organization Report Number** AFIT/GCS/ENG/02M-05
Sponsoring/Monitoring Agency Name(s) and Address(es) AFRL/IFTC ATTN: Zenon Pryk 26 Electronics Parkway Rome, NY 13441-4514	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)

Distribution/Availability Statement
Approved for public release, distribution unlimited

Supplementary Notes
The original document contains color images.

Abstract
Radar is a fundamental technology in today's military and civilian environment, and continuing development of this technology is of utmost importance to maintaining technological advantages this realm. Current radar technologies suffer from jamming and clutter limitations. STAP is a statistical method to remove this noise, however it is extremely computationally intensive, and presents several real time processing hurdles. Clutter Classification is another method to classify the radar returns that are found according to the best fit statistical distribution that the return follows. This research investigation attempts to use this clutter classification technology to aid in the detection of targets by filtering the radar returns and then passing only the target rich data the computationally complex STAP application. This research effort also attempts to optimize the STAP application through this integration to provide real time STAP radar processing power to current platforms with minimal hardware requirements.

Subject Terms Space Time Adaptive Processing, Clutter Classification, PDF Approximation, Non-homogeneity Detector, Target Detector, High Performance Computing	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 167	

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/GCS/ENG/02M-05

Space Time Adaptive Processing and Clutter Classification Integration and Evaluation

THESIS

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Nathan A. Jensen, B.S.

Second Lieutenant, USAF

March, 2002

AFIT/GCS/ENG/02M-05

Space Time Adaptive Processing and Clutter Classification Integration and Evaluation

Nathan A. Jensen, B.S.

Second Lieutenant, USAF

Approved:

Professor Gary B. Lamont Thesis Advisor	27 FEB '02 Date
Professor Eric P. Magee Committee Member	27 FEB 02 Date
Professor Steven C. Gustafson Committee Member	4 Mar 02 Date

Acknowledgements

I would like to thank God, my advisor for dealing with me and all the help and guidance he provided, my wife for tolerating all the late nights and time away from home. To all the freinds I've made here, you guys are the greatest; may your future offer the brightest of opportunity.

Nathan A. Jensen

Table of Contents

List of Figures

List of Tables

Abstract

Radar is a fundamental technology in today's military and civilian environment, and continuing development of this technology is of utmost importance to maintaining a technological advantage this realm. Current radar technologies suffer from jamming and clutter limitations. STAP is a statistical method to remove this noise, however it is extremely computationally intensive, and presents several real time processing hurdles.

Clutter Classification is another method to classify the radar returns that are found according to the best fit statistical distribution that the return follows. This research attempts to use this clutter classification technology to aid in the detection of targets by filtering the radar returns and then passing only the target rich data the the computationally complex STAP application. This research attempts to optimize the STAP application through this integration to provide real time STAP radar processing power to current platforms with minimal hardware requirements.

Space Time Adaptive Processing and Clutter Classification Integration and Evaluation

I. Introduction

Radar is a technology that is integral to many operations in both civilian and military environments. Because of great reliance on this technology, there is intense interest in expanding its efficiency and effectiveness. Since it was first effectively implemented, around the start of WWII, the military has become increasingly reliant on radar technology, and continually demands more functionality and applicability to new platforms. The purpose of this research is to illustrate one method that may enable new and useful performance from the radar platform [DR00].

1.1 Radar Technology

In essence, radar emits electromagnetic radiation waves, and then detects echoes when these waves bounce off some object to indicate its presence [HSW01]. The military constantly pursues new technology to make use of radar. For example, the advent of Doppler radar went beyond indicating only target existence and location, and enabled the return of velocity and motion information about the target [UTK01]. However, with almost every new advance, countermeasures have been developed, including technologies such as jamming, stealth. This technology race continually adds new perfomance and effectiveness, and then negates that greater performance with countermeasures. A fundamental premise is that it is necessary to remain far enough ahead of one's opponent so that current or cutting edge radar technologies remain useful.

One of the major difficulties facing radar capabilities overcomeing jamming, ground clutter, and high noise environments, especially for airborne radar platforms [NRL00]. All of these factors can hamper the operation of typical radar. Even though these difficulties exist, there are several experimental methods to mitigate them. Each has its strengths and weaknesses. One of the methods that may alleviate the effects of jamming and ground

clutter is Space Time Adaptive Processing (STAP). This is a computationally intensive method that statistically removes undesired noise, clutter, and jamming by comparing multiple returns in different locations over a period of time. Although still experimental, it is well known and has produced quality results [AC99a], [AC99b], [CA96], [ML97], [WL99], [WL99a], [WL99b], [ML98], [MW00], [CP00], [TKS01], [YS96], and [FS99]. A downside to this process is its complexity [JW94]. To process STAP in real time applications requires hardware capabilities of contemporary parallel high performance computers. For applications on board an aircraft for example, this hardware requirement severely limits capability [ML97].

Another capability that helps overcome jamming, noise, and clutter is Clutter Classification, which is a less complicated statistical analysis technique. It considers what a typical clutter return may look like, and once a typical clutter distribution is known, future returns may be compared to locally adjacent returns to see if they are significantly different from the surrounding ones. If they are different there may be something of interest in that radar return. The major strength of this method is that it is significantly less complicated than STAP; however, it is not nearly as powerful as the full STAP process. For example, it does not have the ability to inject nulls into the system in directions of strong noise; it simply looks at the distribution of return values. It also is limited in the returns it creates, because it signifies nothing about the strengths of the returns, or their locations. It is merely a look at how the raw data is statistically distributed.

1.2 Research Objective

The purpose of this research is to study and implement the coordinated use of the STAP and Clutter Classification technologies. By using these two methods in conjunction, it may be possible to bolster and highlight the strengths of both methods, while eliminating or reducing the weaknesses of each. The end goal is to create a system that can reliably acquire targets in a noisy, cluttered environment in real time with minimal hardware requirements. If this is accomplished, it would be of great value to the Air Force and the Department of Defense. It may also tend to render current jamming and countermeasures obsolete, and give friendly forces the upper hand in almost any situation.

1.3 Approach and Assumptions

The fundamental purpose of this research is to integrate the STAP application with the Clutter Classification package in order to increase efficiency and effectiveness. Both applications are initially analyzed separately in an attempt to optimize the efficiency and effectiveness of each. Upon completion of application optimization, an integration is completed to achieve the previous objectives. This new application is tested and described with qualitative and quantitative metrics. Throughout this process, focus is maintained on parallel high performance computing. This effort is conducted in close cooperation with AFRL/IF, Rome Labs, NY. The original code for both applications was received from Rome Labs, along with many applicable user manuals and background texts. Upon completion, the final product, along with any optimizations made to the original applications, are to be returned to this sponsor for augmentation into the complete code.

There are several assumptions made in this work, mostly dealing with the applicability of statistical methods and theories. As mentioned earlier, the focus of this work emphasizes the parallel aspects of the two applications, rather than a complete statistical analysis of the problem domain. This statistical analysis would be important, but is clearly beyond the scope of this research. An interesting statistical study of how the addition of clutter and targets change statistical distributions is not discussed, but assumptions are made about these aspects based on empirical evidence. There are relatively few risks associated with this project. Since it is an engineering effort, this research could be useful in many applications, and may offer significant improvements in mission critical technology.

1.4 Thesis Layout

This thesis is organized is seven basic chapters followed by three appendices. The first chapter is an introduction and overview of the process in general. The second is a background chapter that addresses each component application and past optimizations. The third contains the design of the integrated application and improvements made to the component products. The fourth is the actual implementation of the integrated product. The fifth is the design of experiments that are conducted to test the new integrated application, and the sixth contains the results of the application testing described in Chapter five.

The seventh chapter addresses general conclusions drawn from results that were found, as well as possible future work. The first of the three appendices is a simple discussion of radar fundamentals. The second is a genetic algorithm approach to parameter optimization of the STAP appliction, and the third provides a evolutionary algorithm background for the genetic algorithm.

II. Background

This chapter highlights contemporary background research in Space Time Adaptive Processing and Clutter Classification. This serves as a foundation for explanation of the integration of the Clutter Classification and STAP technologies. The chapter begins with discussion of STAP and clutter classification in isolation. Once one understands how each of these components work, the integration of the two is much clearer. It is also helpful to understand some basic radar background information.

2.1 Technique Background and Methodology

First and foremost, before any complicated integration may be undertaken, it makes sense to understand the intricacies of the current processes that are engaged in cooperation. Therefore, STAP is examined, followed by an explanation of the Clutter Classification method in use. Code has already been developed that accomplishes both STAP and Clutter Classification. These efforts are conducted in coordination with efforts currently undertaken at Rome Labs, NY [AC99b]. A mathematical description of the STAP algorithm is the first item discussed. This is followed by a mathematical depiction of the Clutter Classification method in general. After this, each application developed at Rome Labs is discussed and analyzed as a specific instantiation of the mathematical method. Finally, a discussion of previous optimizations, implementations, and enhancements concludes the section.

2.2 Fully Adaptive STAP

Space Time Adaptive Processing, STAP, is a method to increase the effectiveness of a radar system using statistical analysis. Through the use of this process, radar may more successfully identify the existence of a target in a cluttered noisy environment. This particular name is given to the process, because it uses information collected in both the spatial and temporal domains to analyze a return, and then process it to find an anomaly that could possibly be a target [JW94].

2.2.1 The Spatial Domain. The STAP process uses data that is collected from a number of different localities to determine whether a particular return contains a target, or simply jamming and other clutter. To accomplish this, arrays of antennae are placed at a distance from each other. Since a return from a particular object returns differently to each of the separate antenna based on their locality, it is possible to isolate the direction from which the return came. Returns that are not from a desired "look" direction are essentially disregarded. This in effect removes a vast amount of distortion and clutter that the radar may have detected [NAV01]. This process also relies on the statistical appearance of clutter in the desired look direction. If a return is caused by clutter, chances are the locally adjacent returns also display the same type of clutter. Not only does this process look at the strength of the return, but also the Doppler shift of the return as well. This method specifically takes advantage of motion, either by target movement or host platform movement. If no target is present, Doppler shift returns should be nearly the same from spatially local returns. If they are not, this is a clear indication of an anomaly that may be a target. In this manner, it is possible to isolate returns that are statistically different than their surroundings [YS96].

2.2.2 The Time Domain. Not only are several returns collected in the spatial domain, but they are also collected in the temporal domain. Several temporally adjacent returns are collected and analyzed statistically just as were the spatial domains. If no target lies in a specified look direction, statistically, clutter and jamming should be similarly distributed across time as well. If there is a specific strength of Doppler return in one instant, there should not be a drastic change in the return an instant later. In this manner, it is possible to account for changes in both the spatial and temporal domains [FS99].

2.2.3 How STAP Works. As previously mentioned, the process makes use of both the spatial and temporal domains to make a statistical analysis of a set of radar returns. However, there must be a certain organization to the data to allow mathematical processing. The data are therefore arranged in a matrix. These matrices are collected for every "range" that the radar receives a return from. In this manner, that data is formed into a data cube. This cube is subsequently be known as a Coherent Processing

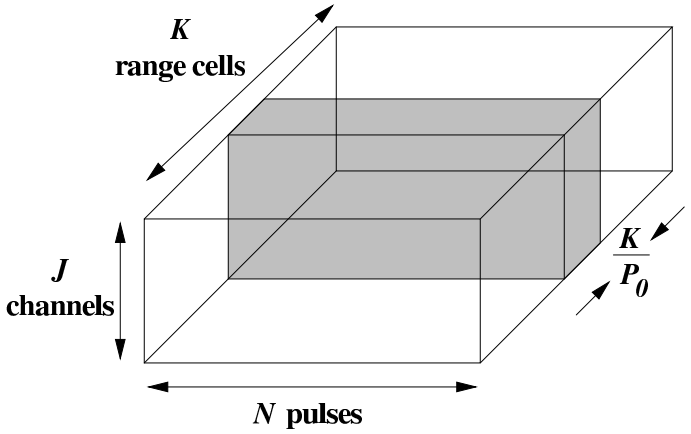

Figure 2.1 Coherent Processing Interval Depiction [AC99b]

Interval, CPI, shown in Figure 2.1. Each location in the cube has an associated magnitude and frequency. These are the attributes that are statistically evaluated. The covariance is calculated between series of elements in the data cube. If there are no targets present in the data cube, one would expect that the returns would have a significantly high covariance. If there is a significant anomaly in the return, for example a target, one would expect that the covariance would be lower. Therefore, the process weights all of the returns with the inverse of the covariance. In this manner, returns that are similar to their surrounding returns are devaluated, while returns that differ significantly from surrounding returns remain strong. This allows targets to be "found" in a cluttered noisy environment [JW94].

2.2.4 Technical/Mathematical Specification. The previous description serves well for a high-level abstraction and overview. However, it is wholly inadequate to describe explicitly and unambiguously the actual specific process of fully adaptive STAP. Therefore, a more complete symbolic representation follows. Even though this discussion is much more complete and specific, it is not all encompassing. Further reading is encouraged to familiarize oneself with the mathematical notation, as well as the fine granularity radar details [JW94]. First and foremost, the actual design of the radar system must be illustrated. Clearly, there must be an antennae array involved here. This allows for multiple spatial returns to be collected. Furthermore, this process is specifically geared to take advantage of Doppler shifting, and is highly advantageous in mobile platform. The use of

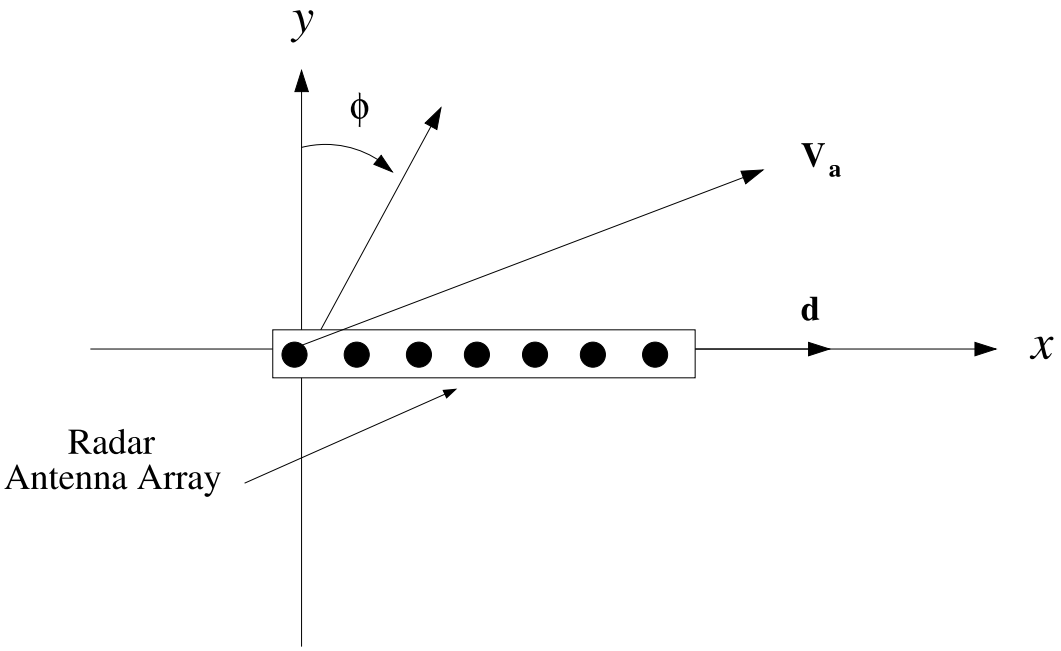

Figure 2.2 Antenna Array Top View [JW94]

the array and mathematical processes actually allow the "movement and positioning" of the main beam without any physical movement of the antennae array [NAV01]. Figure 2.2 and 2.3 illustrates the naming convention that is used to denote the location of the array, target, look direction, angles, and relative velocities. These are necessary to define points of reference used in the detailed STAP explanation [JW94].

The STAP process assumes that the transmit pattern is fixed at a specific angle ϕ and θ. Transmissions are made with a constant pulse repetition frequency, PRF. This is simply the rate at which pulses are emitted from the transmission source. M pulses are transmitted consecutively as previously defined. Given the fact that the PRF is known and constant, one may determine the length of any one pulse. This is known as pulse repetition interval or PRI. Clearly the PRF, denoted as F_r, is related to the PRI, denoted as T_r by equation 2.1 [JW94].

$$F_r = \frac{1}{T_r} \tag{2.1}$$

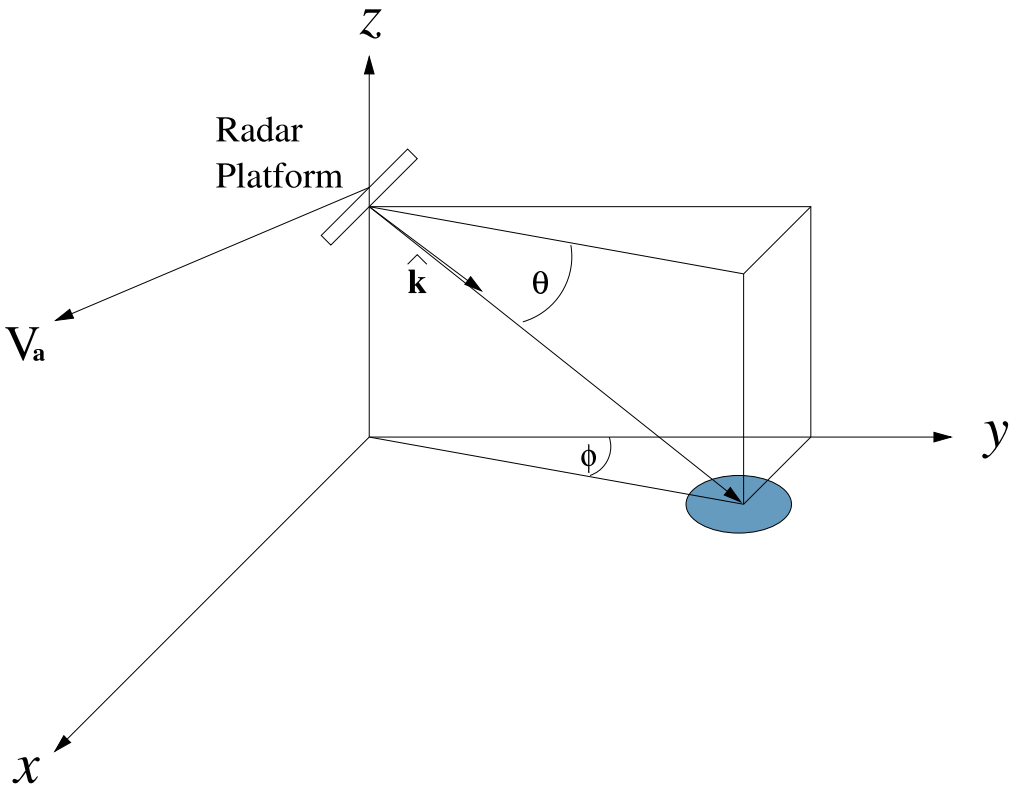

Figure 2.3 Airborne Array Naming Conventions [JW94]

It is also clear from this derivation that there are M pulses in all coherent processing intervals, and the length of those intervals are known. Therefore, the length of any coherent processing interval may be determined with equation 2.2 [FS99].

$$CPI_{length} = MT_r \tag{2.2}$$

Using this representation, it is known exactly where every element of the CPI comes from, and what their respective values represent. It is oftentimes convenient to reference a certain subsection of the data cube. Using traditional matrix notation indicates what subsection of the data cube is in question. For example, x_{nml} specifically denotes the one value in the data cube at location nml. If one wished to denote all of the values as a specific range of interest, it may be explicitly denoted as X_l, where X represents the whole data cube [JW94].

There are two possible options for any range in the data cube. Either it contains a certain amount of clutter in addition to a target, or it contains only clutter and noise. The term X_u is used to represent the portion of the values in a given range cell that may be attributed to noise, jamming, and clutter. This leaves two possibilities, and generates a hypothesis test, shown in Equation 2.3, that determines if a range cell does or does not contain a target with a given confidence [YS96].

$$\begin{aligned} X &= X_u & H_0&\text{: Target Absent} \\ X &= \alpha_t v_t + X_u & H_1&\text{: Target Present} \end{aligned} \tag{2.3}$$

The $\alpha_t v_t$ in the second alternative represents the portion of the range cell that may be attributed to the existence of a target. This provides the mathematical foundation for the STAP process, and frames the task as a statistical hypothesis test to determine whether the returns at a range of interest could have reasonably been formed from clutter, or whether there is a significant shift in the mean to warrant the existence of a target. The v_t term here represents the space time beam steering vector for the specific look direction, while the α_t represents a scaling factor or amplitude of the return. The steering vector is made up of a spatial vector and a temporal vector, represented in Equations 2.4 and 2.5 respectively.

The Kronecker product of the temporal steering vector and the spatial steering vector construct this total steering vector as noted in Equation 2.6, and specifically illustrated in Equation 2.7 [JW94].

$$a(v) = [1; e^{j2\pi v}; \ldots; e^{j(N-1)2\pi v}] \tag{2.4}$$

$$b(\omega) = [1; e^{j2\pi \omega}; \ldots; e^{j(M-1)2\pi \omega}] \tag{2.5}$$

$$v_t(v_t, \omega_t) = b(\omega_t) \otimes a(v_t) \tag{2.6}$$

$$b(\omega_t) \otimes a(v_t) = \begin{bmatrix} 1 \times 1, & 1 \times e^{j2\pi v}, & \ldots, & 1 \times e^{j(N-1)2\pi v} \\ e^{j2\pi \omega} \times 1, & e^{j2\pi \omega} \times e^{j2\pi v}, & \ldots, & e^{j2\pi \omega} \times e^{j(N-1)2\pi v} \\ \ldots, & \ldots, & \ldots, & \ldots \\ e^{j(M-1)2\pi \omega} \times 1, & e^{j(M-1)2\pi \omega} \times e^{j2\pi v}, & \ldots, & e^{j(M-1)2\pi \omega} \times e^{j(N-1)2\pi v} \end{bmatrix} \tag{2.7}$$

This space time beam steering vector is what allows the movement of the main beam without any actual movement of the physical radar array [NAV01], as well as the addition of delays in the correct locations to provide a coherent "snapshot" in time. Given a desired look direction, delays must be injected into the signal based upon known delays in returns from that specified angle. This allows the different returns from the array elements to be shifted such that all returns from this "look direction" arrive at the same time. This is known as the spatial steering vector, or $a(v_t)$ as noted in Equation 2.6. The second steering vector needed is the temporal steering vector, $b(\omega_t)$. As previously discussed, returns are collected over a period of time. Again, delays are added to these returns such that a coherent "snapshot" in time is returned. In this manner, it is possible to ensure that a return from a specified look direction that occurred at a specific point in time

is contained in the same range cell. This temporal steering vector is the $b(\omega_t)$ term in Equation 2.6 [JW94].

The Kronecker product of the two vectors as shown in Equation 2.6 results in a larger $NM \times 1$ vector. This essentially creates M new $N \times 1$ spatial steering vectors, where each vector has additional delay as needed to account for the fact that returns are collected over M different PRI. In this manner, a clear spatially and temporally consistent return may be achieved for further analysis [RB89].

As previously discussed, the fundamental concept behind STAP hinges on the calculation of a covariance matrix. Using this covariance matrix and the steering vector, a weighting matrix may be calculated that properly weights the elements in a given range cell. This covariance matrix is calculated using the expected value of the clutter, jamming, and noise. The expected value is typically derived from the range cells surrounding the range if interest. However, oftentimes the most adjacent cells are disregarded in case the target may overlap range cells. If they were included, the target strength may be artificially deflated, and not be visible. This also relies on the assumption that there are very few targets in any one data cube. If there were many targets, the calculation of the expected value of the clutter would be inaccurate and skew proper weighting. The mathematical relationships for the covariance matrix are given in Equation 2.8 [JW94].

$$R \underline{\Delta} E\{X_u X_u^H\} \tag{2.8}$$

This covariance matrix is impossible to determine perfectly, because only a sample of the return population is known. Therefore, an estimate is created using real time data. This estimate is simply an average of the desired local range cells to determine what values are most typically found in each location. Using this method, it may be possible to find a very good estimate for the true covariance matrix. This estimate calculation is shown in equation 2.9.

$$R' = \frac{1}{K_e} \sum_{l=1}^{K_e} X_l X_l^H \tag{2.9}$$

Using this covariance matrix, a weight matrix can be derived using the inverse of this matrix and the steering vector. This weighting is then applied to the particular range cell in question [RB89]. Upon weighting, the expected value of the clutter in that particular return is removed to reveal a possible target. The mathematical method for determining this weight matrix is given in Equation 2.10 [JW94].

$$w = R^{-1} v_t \tag{2.10}$$

2.3 Partially Adaptive STAP

The fully adaptive STAP algorithm algorithm is basically beyond the capability of current computer technologies. Therefore, researchers have developed several methods to reduce the overall complexity of the algorithm. This allows realistic computation times to be achieved on current high performance computing platforms. This is done by reducing the degrees of freedom that may be encountered in one of the dimensions of the STAP application. There are four main types of reduced dimension STAP:

1. Element Space Pre-Doppler – In this reduction a Doppler filter processes the data after adaptive processing. The data from a few local pulses are adaptively combined, and then a non adaptive Doppler filter returns integration of the CPI with velocity information
2. Element Space Post-Doppler – In this reduction of dimensionality, a Doppler filter processes the entire range of pulses, and produces Doppler bins, rather than pulses. The elements are then processed adaptively. This is not a true space-time adaptive process, because it is only adaptively processed in element space.
3. Beam Space Pre-Doppler – The beam space dimensionality reduction uses a discrete Fourier transform to reduce the dimensionality of the elements. In this manner, the full range of angles is categorized into discrete groups for processing. This data is then adaptively processed a few elements at a time with the pulses, and then passes through a Doppler filter. This method is much more effective if the direction of clutter

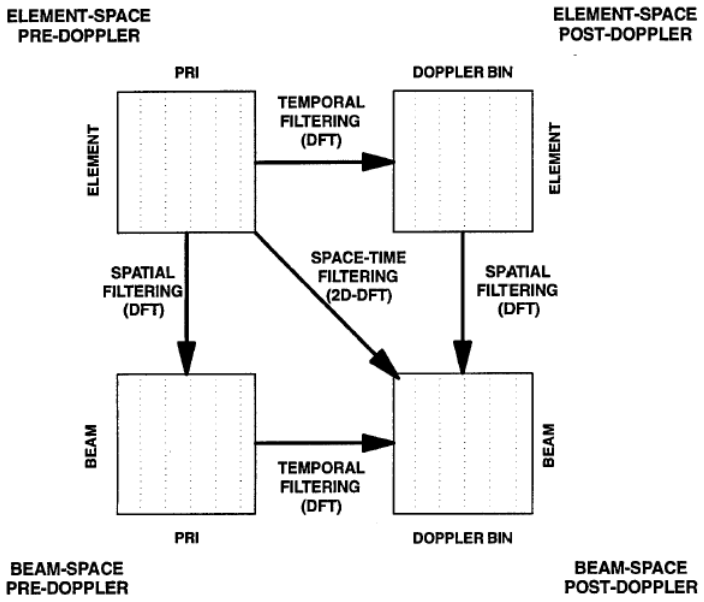

Figure 2.4 Reduced Dimension STAP Taxonomy

and jamming is known a priori, because the beams may be directed in directions of known interest.

4. Beam Space Post-Doppler – This dimensionality reduction reduces the degrees of freedom in both the pulses and elements dimension. A two dimensional discrete Fourier transform is used to discretize the elements into a set of beams and the pulses into a set of Doppler bins. This, combined with some environment knowledge may be used to significantly reduce clutter and jamming by itself. These discrete groups are then adaptively processed.

A taxonomy of these different dimensionality reductions is contained in Figure 2.4. This Figure shows how different filters are employed in what order for each particular dimensionality reduction. The particular implementation that is used in this research is the Element Space Post-Doppler. This is not because the solution is better than others, it is simply the case the application already provided by Rome Labs uses this method of processing. Therefore, the focus of this research is directed toward this reduced dimensionality version of STAP.

2.4 Clutter Classification Background

The clutter classification process is not as well understood as the STAP process. It relies on a statistical hypothesis test to complete a goodness of fit analysis. This is accomplished through a statistical process known as the Ozturk method [US98]. This method in isolation is well understood. It is a method to evaluate a data set. It allows one to discover what distribution a particular sample may follow, as well as how well the particular data fits that distribution. A series of radar returns is analyzed to see if they too fit the same distribution as the training data. If indeed they do, it is reasonable to believe that there is nothing of interest in the return. If the return is distributed differently than the training data, it is reasonable to assume that there may be something of interest in the range cell. It is not clear that this is always the case, but it is used as a hypothesis for testing and creation of the application.

2.4.1 Ozturk Method Background/Mathematical Definition. The Ozturk method is a process that allows one to determine which base distribution a test set may follow. Most hypothesis tests in this area are geared toward determining if a test set follows a specific distribution. The Ozturk method is much more powerful than many other options, because it not only can determine if a set follows a distribution, but it also tests to see what other distributions it may follow, as well as the goodness of fit of those base distributions. This process is also computationally simple. It does not require a large portion of the data set, as most other goodness of fit tests do [AO93].

The Ozturk method's fundamental premise is the representation of order statistics by linked vectors on a two dimensional plane. Suppose that n sample observations are taken. These samples are sorted in increasing order such that $X_{1:n} < X_{2:n} < ... < X_{n:n}$. It is also necessary to have an initial null distribution as well. Even though this method identifies the best fit distribution, it still starts with a known base distribution, $F_0(x, \mu, \sigma)$ with location μ and scale σ, such that: [AO91]

$$\begin{aligned} &F_0(x, \mu, \sigma) = F_0((x - \mu)/\sigma) \\ &\text{where } \mu = 0 \text{ and } \sigma = 1 \end{aligned} \tag{2.11}$$

It is also necessary to know the expected value of the sorted observations from the null distribution denoted by $m_{1:n}, m_{2:n}, ..., m_{n:n}$. These expected values are typically determined by Monte Carlo estimates. Many samples of size N are created from the known null distribution. These samples are sorted and then averaged according to their element location. The standardized sample order statistics, $Y_{1:n}, Y_{2:n}, \ldots, Y_{n:n}$ are also needed for the process. This array is calculated from the unidentified test where: $x_{i:n}$ is ordered sample observation i, $\bar{x}$ is the sample mean, and s is the sample standard deviation, set such that: [AO92a]

$$Y_{i:n} = |x_{i:n} - \bar{x}|/s \quad i = 1, 2, ..., n \tag{2.12}$$

The cornerstone of the Ozturk algorithm focuses on a Q statistic that is derived from a U and V statistic. This allows the sample order statistics to be represented by linked vectors in a two dimensional plane. These statistics are defined in Equations 2.13, 2.14, and 2.15 [SL94].

$$Q_{i:n} = (U_{i:n}, V_{i:n}) \quad \text{where } i = 1, 2, ..., n \tag{2.13}$$

$$U_{i:n} = \frac{1}{n} \sum_{j=1}^{i} Cos\left[\pi F_o(m_{j:n})\right] Y_{i:n} \tag{2.14}$$

$$V_{i:n} = \frac{1}{n} \sum_{j=1}^{i} Sin\left[\pi F_o(m_{j:n})\right] Y_{i:n} \tag{2.15}$$

The fundamental premise of the Ozturk method relies on these $Q_{i:n}$ statistics. It is crucial to note that these statistics are scale and location invariant, which allows a determination of what distribution the test set may follow. Each Q is plotted two dimensionally as a linked vector. They are plotted in order, with the tail of the next statistic starting at the head of the previous statistic vector. In this manner, the final statistic, $Q_{n:n}$, may be compared across test sets to determine which base distribution the test set most closely follows.

The technical explanation in section 2.2.4 is accurate, however it is not simple. There are several processes that must take place to allow this to happen, and many of these processes are extremely computationally intensive when performed numerous times for different locations in the data cube. There are many different variations on how this process is accomplished. However, the exact process used in this project is the algorithm developed at Rome Labs. The process that must be followed to allow these calculations follows; afterwhich, an explanation of the how these processes are accomplished in the Rome application is presented [AC99a]. First one must note that Rome Labs has deviated in their naming convention from other standard forms. In this section, the channels are denoted by a J, rather than an N. Pulses are named with N, rather than M, and range cells are denoted with K instead of L. Other than this minor change, the Rome application closely mirrors the same process described in the mathematical model in this section.

2.5.1 Doppler Filter Processing. Doppler Filter Processing is the first stage in the implementation algorithm. The particular algorithm at hand actually staggers several, (usually two), CPI together. This is known as PRI staggering. Even though this amount is variable, it is most often done in sets of two. Melding these two CPI result in a data set twice as large as the original. The data set is passed through a hamming or windowing function first. Once completed, an FFT is performed at every channel, $2J$, and range cell, K. The FFT is a pulses, N, point FFT [AC99b].

The output of the Doppler filtering task is a complex $K \times 2J \times N$ datacube. This may then be passed to the next computational component. Rather than pulses, the N bins now contain Doppler data. Since this process is an Element Space post Doppler STAP, it is not a true STAP algorithm. True STAP is adaptive in both Space and Time. This algorithm is only adaptive in the space domain. This is why the resultant matrix has several Doppler values at every range cell, rather than one single value. Once this data cube has been created and processed, it is passed to the weight computation stages.

In the given implementation, three data cubes are immediately read in during this phase. They are processed serially and then passed to the different weights and beam

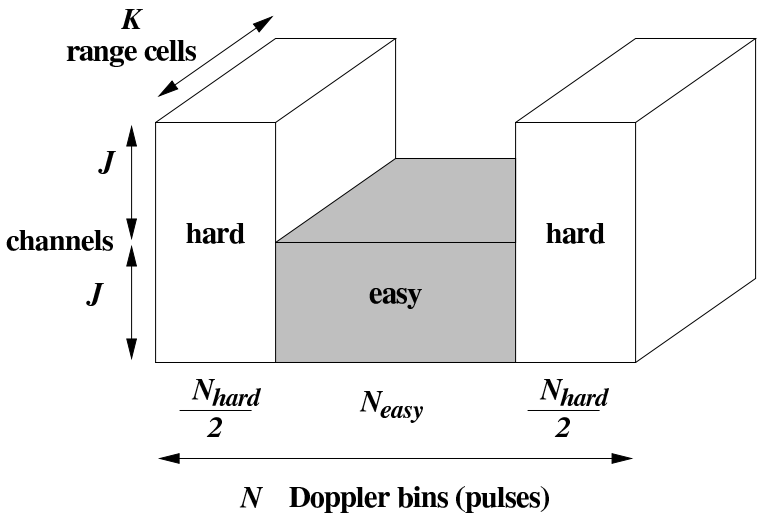

Figure 2.5 Doppler Filter Stage Output [AC99b]

forming elements. After the three initial "pipe fillers", the filter group either creates fake data, or it continues to read real data cubes depending on pre compiler directives. This is repeated a number of times depending upon a setting in the parameter file. This allows the timing of throughput once the data pipe has been filled.

The next stages in the pipeline all need different amounts and sections of the data cube. The Doppler filter group passes a $N_{easy} \times J \times \frac{K}{2}$ sized matrix to the easy weight computation. It passes a $N_{hard} \times 2J \times K$ sized matrix to the hard weight computation. Both beam forming stages need the entire data cube. The output produced by this stage is illustrated in Figure 2.5 [AC99b]

2.5.2 Easy Weight Computation. Easy weight computation is one method to reduce the amount of work that is involved with STAP processing. Easy weight computation is accomplished on areas of the return signal that are not likely to have a significant amount of ground clutter in the return, (Doppler bins that are not close to the main beam). This particular algorithm does not use the upper half of the staggered data, only half of the range bins, and it does not use the sides of the data cube. Therefore, the actual processing takes place on a cube much smaller than the original. It is also interesting to note that the weight matrix for the CPI is formed using data from the three previous CPI's. This

insures an independent data set from which to collect a clutter crossection. A depiction of this data decomposition is given in Figure 2.6 [AC99b].

Again, this process is only adaptive in the space domain. Therefore, adaptive weights are only determined for N matrices. A specified amount of range cells are selected for use in this analysis. This specified amount of data is taken from each of the three previous CPI. This results in N $3(Easyrangesamples) \times J$ matrices. These matrices are solved for a weighting vector.

The weighting matrix is derived using a QR factorization to make the process more efficient [RB89]. This changes the matrix into one diagonal matrix, and one rectangular matrix, Q and R respectively; the product of the two result in the original matrix. Once this has been completed, the smaller R matrix may be manipulated. Upon manipulation, the resultant matrix may be back solved for a weight matrix. In the case of easy weight computation, the resulting matrix is of the form $N \times M \times J$, where M is the number of look directions in the steering vector. This computation is graphically depicted in Figure 2.7 [AC99b].

2.5.3 Hard Weight Computation. The hard weighting is not substantially different from the easy weighting. The only difference is that it is much more computationally intensive. This single stage usually makes up for over 40% of the entire work performed in the STAP algorithm. Rather than only using part of the staggered data cube, it makes use of the entire channel set. It also uses a past condition to augment the current state. The matrix also requires the use of more range cells to come up with accurate results. The current matrix undergoes a QR factorization [RB89]. This factorization is then augmented with the previous R, that has been "forgot" over time. This augmented R is then updated with current clutter samples, weight constraints, and then back solved for the weight matrix.

To make this process more accurate, the data cube is divided into six separate segments of range cells. From the first group of range cells, a number of these are used as sample cells. They are placed in a matrix within their respective channels. The QR factorization and weighting is accomplished for this subsection of the data [RB89]. This

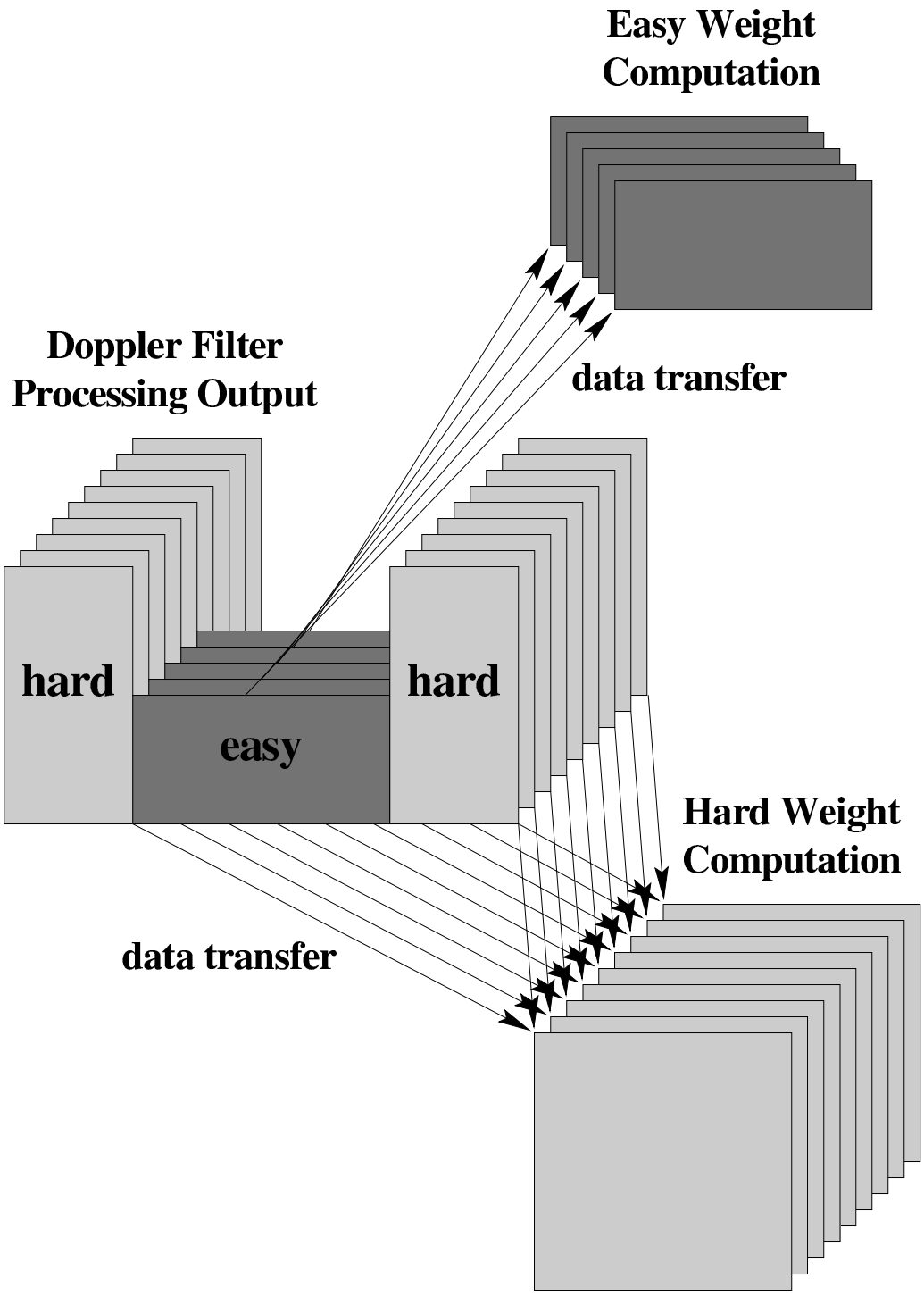

Figure 2.6 Decomposition of CPI [AC99b]

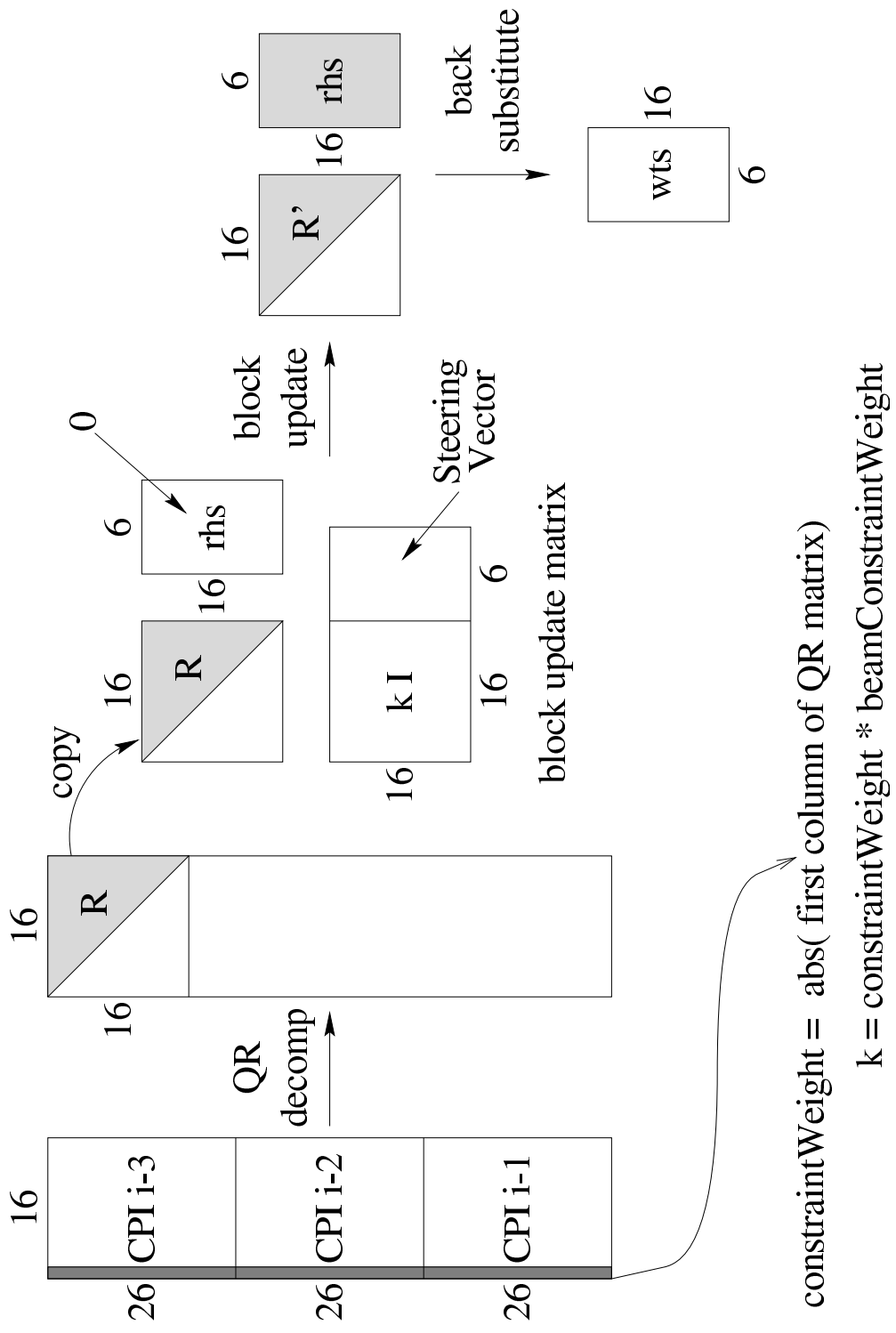

Figure 2.7 Easy Weight Matrix Computation [AC99b]

process is repeated for the five remaining data range segments. This is done to increase the accuracy of clutter nulling. If there is a great amount of clutter (which there probably is in the main beam) that clutter may vary significantly over the entire range of the data cube. This process breaks up the data cube and looks for statistical anomalies in local regions where clutter is much more likely to be homogeneous [AC99b].

The result of the hard weight calculation is a $6N \times M \times 2J$ matrix. M represents the number of beams in the steering vector (if there was only one desired "look direction" M would be one). A graphical depiction of this process is inclueded in Figure 2.8 [AC99b].

2.5.4 Easy Beam Forming. The beam-forming step is when the weights are actually applied to the data cubes. This is nothing more than matrix multiplication. There are N different weight matrices that are multiplied by the N different Doppler bin matrices. Obviously this results in N– $M \times K$ matrices (elimination of the different channels). This leaves the result of Doppler bins associated only with range cells for every intended beam [AC99b]. If this were a true fully adaptive STAP process, the Doppler returns would have also been condensed into one return [JW94].

2.5.5 Hard Beam Forming. The hard beam forming is exactly like the easy beam forming, however there is more computation to be done. In this case, the weighting consists of $6N$– $M \times 2J$ matrices. The Doppler filtered data cube is divided into 6 sections of range cells, corresponding to the matrices of the weighting cube. By accomplishing 6N matrix multiplications, the results become an $N \times M \times K$ matrix just as it did in the easy beam forming. The combination of these two resultant cubes is now ready for analysis and target location [AC99b].

2.5.6 Pulse Compression. This task involves the convolution of the resultant matrix with the original transmit wave pulse. This is accomplished by performing a K point FFT on the data set. Once this has been accomplished, a simple point wise multiplication occurs, after which, an inverse FFT can again return the resultant $N \times M \times K$ matrix [AC99b].

Figure 2.8 Hard Weight Matrix Computation [AC99b]

2.5.7 Constant False Alarm Rate (CFAR). Constant False Alarm Rate is one method to statistically look at the processed sample and determine if the return is created by clutter, or if it is in fact a target. There is a constant probability of a false alarm, and if a certain test cell is different enough from its neighboring range cells, then it sets off an alert that signifies the finding of a target [AC99b].

2.6 Rome Lab's Parallel Pipeline

First and foremost, the objective in this realm is the ability to process STAP information real time for use in a radar system. Even though this is unrealistic in this laboratory environment, the optimization and theory explored in this environment is equally applicable to a high performance machine that may indeed be able to process this data real time. Therefore, the exercise becomes a task to complete the STAP process as quickly as possible while still returning viable results. One possible solution to implement the previously described processes in an efficient manner is offered in Rome Lab's Parallel Pipelined STAP application. The parallel pipeline is illustrated in Figure 2.9 [AC99a].

The general STAP process as described above does not lend itself well to parallelism. Every state is dependent upon the previous state. Therefore, a different process must be found to improve performance. The answer is a pipeline. Each of the above processes may be pipelined to improve throughput. Although this does not improve latency (it actually increases if only one processor is used per group due to communications overhead) it significantly improves throughput. The second optimization that is easily implemented is the parallelization of the actual pipeline stages. The computation accomplished in these stages may indeed be highly parallel. However, the stages of the pipeline require vastly different amounts of work. For example, the hard weighting contains just less than half of the overall computational work. Therefore, it makes sense to apply a higher degree of parallelism to the particularly difficult pipeline stages. Just in familiarization runs, the execution time varied from 1 minute to process 29 data cubes to 25 seconds to process 29 data cubes by adding a few processors in key locations. Although this example is certainly not scientific, it does lead one to believe that optimization of resources is crucial in this problem.

Figure 2.9 Parallel Pipeline Architecture [AC99a]

This pipeline implementation allows the user to set up a cluster of nodes into processing groups that are used in the different pipeline stages. This allows the user to isolate where any particular bottleneck may exist, and then more processing power to that area to alleviate the bottleneck. Even though every pipeline stage does something different, every machine is still running an exact copy of the same binary. Using rank to distinguish different processes in the run there are different areas of the binary that deal with different stages in the pipeline [PP96]. The results of a run are a textual output detailing the computation, send, and receive times in different stages. It also produces an analysis of where targets may exist.

2.7 Clutter Classification Implementation Discussion

Unfortunately, the given implementation of the clutter classification code is not well documented. There are both FORTRAN77 and C portions of the code, and the "brunt" of the code is in one very large file. This seriously hampers the efforts to run and analyze the code. Currently the code does not run on the AFIT Heterogenous Cluster or at Rome Labs, there are some minor details to work out in compiling and linking for a normal cluster of workstations. Furthermore, it is not particularly geared toward use on STAP radar data sets in its current form, it merely reads a set of numbers and then calculates different statistical traits that the set portrays. There are many syntactic and symantic issues to be addressed on this code to say the least, and it makes a coherent discussion about the applications' intricacies difficult. However, the discussion is necessary and one must start somewhere. A discussion of the mathematics behind the Ozturk algorithm is given, followed by a code walk through for the parallel implementation of the Ozturk method implemented by Rome Labs [FC01].

2.7.1 Initialization and Setup. The first thing that is done in the Ozturk code is the initialization of all the processes and Message Passing Interface MPI overhead. In this software architecture, there is one master task; all of the other processors are simply slaves that accept work and then return the processed data when called upon. The slaves only function during two functions. When the master process reaches one of the

functions, it passes data to slaves. All of the processes then run in parallel to complete the procedure. Once the procedure is done, the master process collects the data, and progression continues in a serial manner on the master. After the MPI initialization, the master reads some initialization data from a file specified on the command line, or from a default initialization file, (ozturk.init). The initial data is also read in as specified in the initiation file [FC01].

2.7.2 Serial Processing Stage. The first real processing that the master process completes is the calculation of sample statistics on the data set. These statistics are the mean, variance, and correlation coefficient. However, one must keep in mind that if this is complex data that there is both a real and imaginary portion of the value. Therefore, there are two means, two variances, and one correlation coefficient. If there is only one single column of numbers, as is the case with the current configuration, it appears that the numbers are considered as real, and the imaginary portion is disregarded. From here there are two options: use the same input set for the raw data set, or read in another raw data set. Currently, the code simply uses the same set for both data sets. Just as for the input data, the sample statistics for the raw data set are calculated as well. Next, the master process averages the input and raw data set according to initialization options. One of three options is chosen: no average, subtract mean, or two pulse. The actions that are accomplished are quite simple. In the subtract mean procedure, the predetermined mean is subtracted from every data element. If the two pulse option is chosen, every element is decreased by the amount of the next element in the series. Once this "average" is accomplished, the sample statistics are again calculated for the sets. Now the actual Ozturk method is used to determine the distribution identification [FC01].

2.7.3 Ozturk Method. This procedure is the heart of the clutter classification system. It allows the determination of a probability density function for a data set, as well as a goodness of fit metric for that set. Initially, the main Ozturk method does some initialization concerning the standard normal distribution, and the number of samples. The first thing that is called in the main Ozturk method is the MSTAR procedure. [FC01]

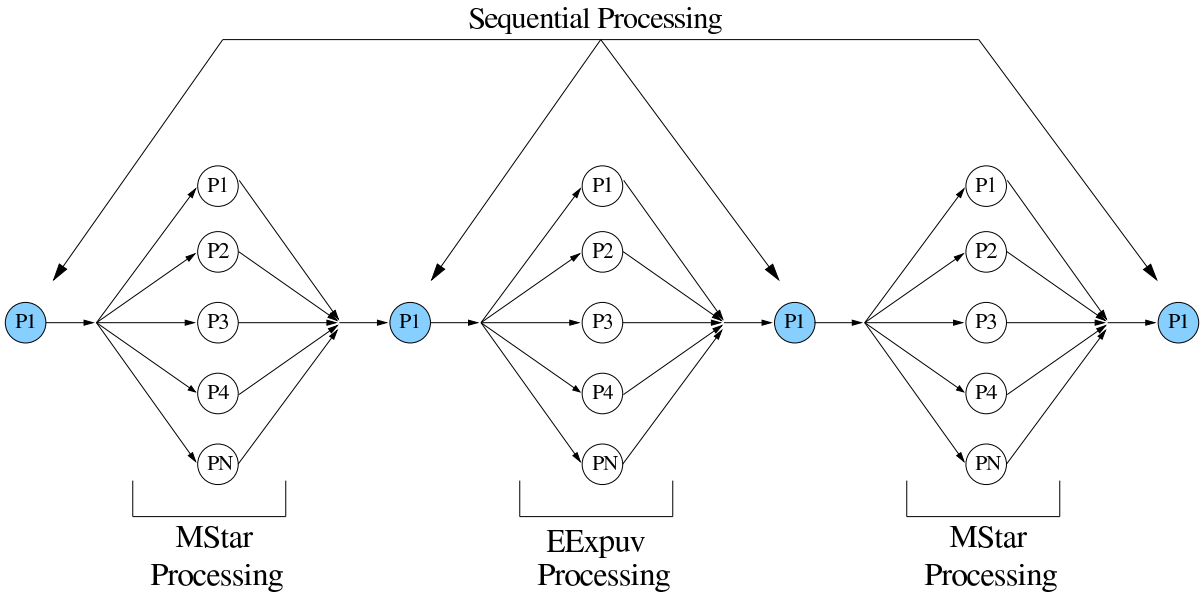

Figure 2.10 Parallelization Clutter Classification [US98]

2.7.4 MSTAR Procedure. The MSTAR method calls a C function that packs data that is passed to the slave processes into a continuous buffer. The packed buffer is then sent via an MPI broadcast to all of the other processes that are currently waiting in the slave loop. Once the data has been broadcast, the par_mstar routine is called. Inside of this procedure is where the brunt of the Ozturk work is completed in this particular version. This is also where the program becomes parallel. The slave loop receives data from the master, processes some setup information, and then commences the par_mstar procedure as well. This behavior is shown in Figure 2.10 [FC01]. Note the decomposition illustrated in this figure carefully, because it has been changed drastically in the new implementation [CC01].

2.7.5 Par_MSTAR Procdure. The purpose of the Par_MSTAR procedure is to obtain the expected values of specific elements in the sample array, Equations 2.16 and 2.17, where x is the sample point, $\bar{x}$ is the mean, and S is the sample standard deviation. It is clearly extremely useful if one knows how many standard deviations away from the mean a certain element in the array is expected to be based on the index in that array [AO92b].

$$\frac{(x - \bar{x})}{S} \tag{2.16}$$

$$\frac{|x - \bar{x}|}{S} \tag{2.17}$$

This first thing that is done in the par_mstar procedure is the decomposition of computation done by each of the processes. Each process calls gen_my_rep. In this procedure, the number of replications is divided by the number of processes. The ceiling of this answer is the number of replications that must be performed by each process. Each process then creates a distribution that conforms to the specified input distribution. This set is then sorted. Each element is added into a collecting array. Next, a procedure is called that sets each element in the data set array to its distance in standard deviation away from the mean. These are also added to a different collection array. This process is repeated the number of replication times. The end result is two arrays. The first contains the sums of all the sorted elements, the first element would be the number of replication smallest elements summed, and the second contains the sum of the sorted distances from the mean in standard deviations. Once all of the processes have completed processing their particular partition of data, all of the local summations are passed back the master controlling procedure, and summed using an MPI_Reduce [FC01].

When MSTAR regains the summed arrays, it divides each element in the array by the number of replications. This basically determines the expected value of each element in the array by creating a large quantity of distributions, and then calculating the average value of each element in the array. This serves as a measure against which the incoming data set may be compared.

2.7.6 COEFF Procedure. The coeff procedure produces coefficients that are used for the U and V statistics calculation. This is the process that assists in dissecting the single value into the two dimensional linked vector. Given a particular distribution, an angle, theta, is produced. This angle can then be used to return either portion of the original signal [FC01].

2.7.7 EEXPUV Procedure. This is another work intensive procedure in the process. It is the only other procedure that takes advantage of parallelism. Just as is the case with the MSTAR procedure, there are some bookkeeping details taken care of at first. Second, the data is packed by a C function, and then passed to the slave processes that are waiting in the slave loop procedure. Once the data has been passed, the procedure par_eexpuv is called. This is very similar to the same way that the par_mstar procedure was controlled as well [FC01]. Again, this particular operation does not appear to serve any purpose in the original FORTRAN code, and has been omitted in the new implementation [CC01].

2.7.8 Par_EEXPUV Procedure. Just as was the case with the par_mstar procedure, the first thing that this procedure accomplishes is determining how much work each slave must accomplish. Just as before, there is a pre-determined number of replications that must be performed. Each slave performs an equal number, or as close to equal as possible, of these replications. Once this is determined, each process creates a data set according to the specified distribution. Once created, the U and V statistics are determined for each element. What they represent mathematically it is not extremely complicated. Each U is dependent upon the previous U's as well as standard deviation of the current sample multiplied by the coefficient for U divided by the number of samples. It is identical for V, except previous V's and V coefficients are used [FC01]. The U and V that are at the end, as well as half way through, are collected, and then passed back to the calling procedure. Once the master procedure has all upper U's and V's from each slave, these are used to calculate the correlation coefficient for the bivariate data, as well as several Johnson distribution statistics. In the same manner, the correlation coefficient for the half U's and V's are calculated [AO91].

2.7.9 DisID Procedure. Once the data statistics have been determined, a call to DisID is made. This procedure cycles through all of the known distributions that may be created. For each distribution, the subroutine Dis01 is called. Inside this procedure, number of replications distributions are created of that type. Statistics are calculated on these distributions given the coefficients that were previously determined. Once this has

been done, the average of the last U and V statistic is returned. Again, these are certainly useful statistics, but they do not seem entirely relevant to the task at hand considering the ability to obtain these values from given tables. One also must question why new distributions are continually generated. Once a good expected value for any statistic is known, there should be little reason to recalculate this data [FC01].

2.7.10 Parest Procedure. Up to this point, all the work has been focused toward creating a data set from which to draw conclusions. Now, the task remains to actually draw some conclusion about the input raw data. First, the data is either reread from the disk, or it is created using the *nlsamp* procedure. It is also copied into another identical array. Both of these arrays are sorted according to size, and the *statuv* procedure processed the first with the coefficients that were derived from the earlier calculations. The last elements of the U and V arrays are saved. Once completed, the expected value of U and V are calculated with the coefficients and the expected values of the standardized statistics array (emstar). After completion of the previous stages, the algorithm actually begins to estimate the desired parameters of the distribution. This is the part of the Ozturk method that is extremely insightful. This process uses a table of values to determine the statistics of a given distribution. It then evaluates the unknown against these table statistics. It keeps track of how far away each distribution's statistics are from the unknown's statistics. In this manner, all the distributions are ranked according to which ones are the closest to the unknown. The five closest distributions are returned to the user in order from best to worst. This is the desired output that is later integrated into the STAP procedure [FC01].

2.8 Motivation for a C Clutter Classification Port

Unfortunately, the clutter classification code, especially in its current form, is not well tested, or documented. The code as a whole is not organized well. There are only two actual files. The first file that contains nearly all of the code is a large conglomeration of different efforts. Little regard was given to modularity and functional division of code. Many of the functions and procedures are not documented well. The code is also written in FORTRAN. There are many scoping issues with the code style used; it is extremely

difficult to follow the many "go to" statements, as well as where they return to on the run time stack. This code was also written for FORTRAN with extensions. The compiler that exists on the AFIT Heterogeneous Cluster does not support extensions. This also complicates a coherent analysis and evaluation of the code [FC01].

Efforts at Rome are currently producing a non-extended version of the FORTRAN code. This code is usable with the g77 non-extended FORTRAN compiler that exists on many Linux clusters [JB98]. Current efforts have produced a version of the code that compiles on the cluster, however it does not produce correct results, especially when run in parallel. Also, this code does compile on the cluster of workstations, (see Appendix E for complete hardware specifications). However, it does not perform properly and it fails to complete. It crashes during an MPI reduce, even though there is no apparent reason why there should be a problem in that location. These are all problems that were dealt with during this research effort.

In order to produce a more robust and effective solution to the Clutter Classification problem, as well as achieve a more seamless integration with current STAP procedure, the current implementation of Clutter Classification was ported to the C language. During this port, each of procedures in the original Clutter Classification code were evaluated to ensure that each is needed in the final implementation, as well as whether each portion has been implemented in the most efficient manner possible. Upon a close look at the current implementation of the code there are several items that one notices. The current implementation is entirely inefficient, it does a vast amount of computation that is just not needed for distribution identification, and the entire prospect of running the code in parallel becomes moot. A proper implementation of this technology may be accomplished that is so efficient that there is no need to parallelize the actual identification of a distribution. However, given the number of identifications that must be made, as well as the independent nature of those identifications, it may be profitable to execute many identifications in parallel.

The new port of the Clutter Classification code is much slimmer and efficient than the original implementation. This implementation starts by creating set of distributions that are used for statistical analysis later. This is done through the use of the MSTAR proce-

dure. It is not any different than the original implementation in FORTRAN [US98], except there is no parallel computation whatsoever. These distributions are created, summed, and averaged before any analysis of the unknown distribution is accomplished. The results of this MSTAR run then serve as the input for a COEFF call. An array of coefficient statistics is created from the expected value distribution that was created with the MSTAR call. These statistics are then used for every subsequent identification of unknowns. There is no need to recreate the MSTAR call. This results in extensive time saving when considering the application of the Clutter Classification code in real time. The other EEXPUV and DISID calls that were in the original are also useless in the new implementation. There is no need for them, because the end goal is simply a rank ordering of possible source distributions

When considering the possible optimizations, the Ozturk method simply becomes a search to see how far away the statistics of the unknown distribution are from the statistics of a known distribution. These known values are determined from a table lookup. Since there are 27 known distributions, listed in Appendix A, that are searched, every element in the unknown must be examined 27 times. The unknowns are ranked according to its fitness. This identification phase has a complexity of $27n$ where n is the sample size. This process is repeated for following unknowns. There is no need to recalculate thesource statistics, which results in a speedup of many orders of magnitude.

2.9 Background Summary

In conclusion, this chapter is the background behind the problem. Fortunately, the STAP technology has been rigorously examined in past research. There is much information to serve as a foundation for present and future research. However, the direction that was chosen for this research is new terrain entirely. The Clutter Classification does not have such a solid background, however, there is enough information on the topic to serve as a baseline model. According to literature reviews, the Ozturk method, Clutter Classification, has never been used to determine the presence or absence of targets in STAP returns, and should provide an extremely promising research area. In fact, even the Clutter Classification application in isolation has never been used to detect targets. This chapter is

organized with two main sections. The first is a mathematical and theoreticl background, while the second is a look at a particular instantiation of the mathematical method.

III. Design of Integrated Application

Based upon Chapter II background knowledge, the design model for the Clutter Classification and Space Time Processing integration can be created. The following sections discuss adaptations of the Clutter Classifications, modification to the STAP program, and compiling the two applications into one. As indicated in chapter I, the main purpose of this research effort is to combine the STAP and the Clutter Classification programs into a product that has the strengths of both products, and the weaknesses of neither.

3.1 Initial Integration Concerns

The clutter classification fits into the parallel pipeline architecture quite well. From preliminary observations, it appears that clutter classification has the ability to look at one range cell at a time, and then predict if there is anything interesting residing in that range cell. If something interesting is found in one or more range cells from a data cube, then that data cube is passed to the STAP procedure for a further more thorough analysis. Clutter classification simply becomes the first stage in the parallel pipeline. It serves as a method to separate the interesting and non-interesting radar returns from one another. In this manner, the only time that the computationally intensive STAP is invoked is when there is a high probability that a target may be found in a given data cube. The new pipeline is shown in Figure 3.1.

Even though it does appear that the Clutter Classification does integrate well, this does bring light to some complications that were encountered that have not been addressed in previous chapters. Specifically, when using STAP, the weighting vectors are reliant upon past data cubes. If adjacent data cubes are dropped from the input stream, how does that effect the creation of a reliable weighting matrix? Furthermore, the removal of data cubes from the input stream may increase the throughput of the overall system. However, it does not help the latency issues with the overall pipeline. In fact, the addition of another stage at the front of the pipeline may even increase the overall latency when the data cube must pass through the entire pipeline structure. Both of these items are serious issues that must be addressed. There are some very significant changes that are made to the separate

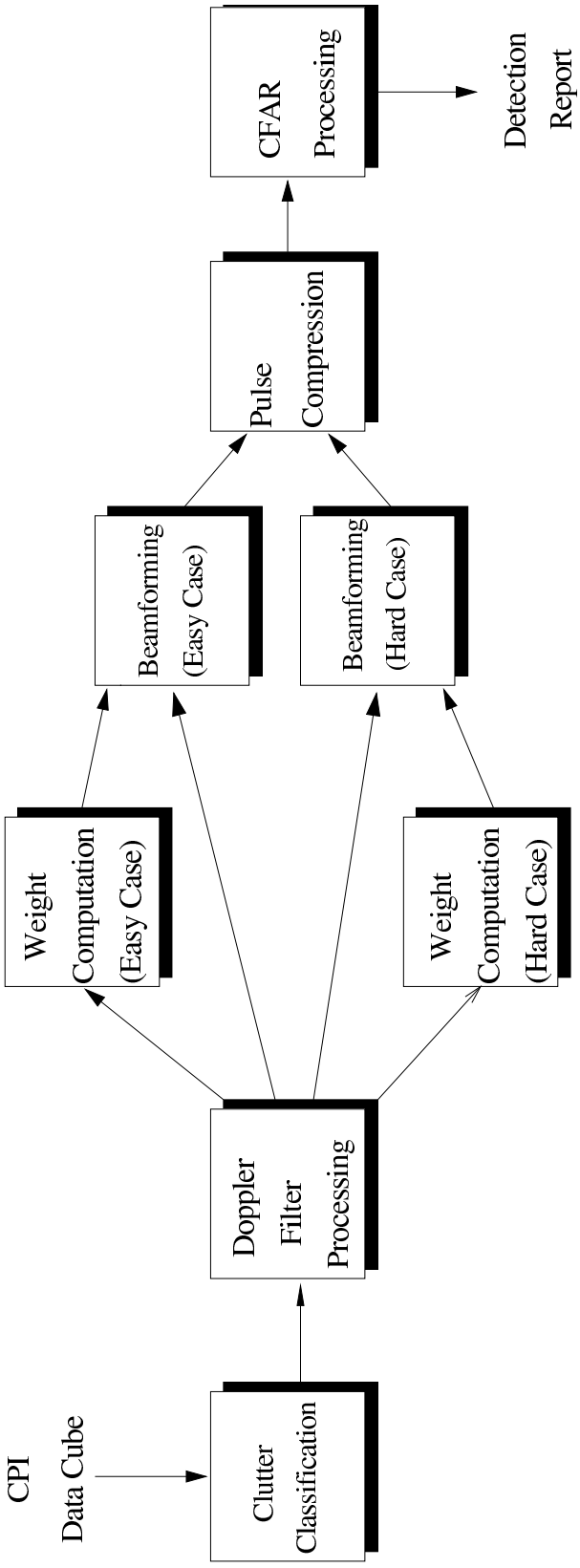

Figure 3.1 STAP Clutter Classification Integration

products to complete the interface. The direct port from Rome Lab's FORTRAN code to a *C* version of the Clutter Classification is completely incapable of coherent use on a STAP radar data set. Drastic changes to the current first stage of the STAP pipeline are also required. Upon completion of these modifications, the integration of the applications is discussed. Each of these items are addressed in the following sections.

3.2 Clutter Classification Modifications

Rome Lab's version of the Clutter Classification code is not implemented for taking a stream of distributions as input, corresponding to range cells in this case, and then processing them in a meaningful manner. Rome Lab's implementation of STAP also expects CPI to reside in binary Matlab file [AC99b]. The Clutter Classification does not have the ability to process data in this format. The first thing that is accomplished is to adapt the *C* version of Rome Lab's Clutter Classification code so that it may be used to process the binary Matlab CPI file.

The Rome Lab's STAP implementation expects the data to be stored in a row stride that crosses the PRI of the first antenna first. Then the PRI of each consecutive antenna, until the entire range cell is exhausted. Then the next range cell is processed in the same manner. Using this method, the entire three dimensional CPI data cube can be stored in a contiguous single dimension array. Each return element has a real and imaginary portion. These are stored separately using rectangular coordinates that are made up of short integers. Each short integer requires two bytes of storage, which in turn implies four bytes per element, 8192 bytes per range cell and 4.19 MB per CPI in the default data configuration. Using short integers significantly reduces the amount of storage needed, as well as mathematical complexity. However, one must also note that there may be serious accuracy issues involved in the truncation process [AC99a].

The real strength of the improved Clutter Classification port is exploited when multiple distributions are identified using the same setup data that was determined a priori. This part of the Clutter Classification is particularly efficient when processing the STAP data cube. Each range cell is considered a distribution. All of the range cells that make up the CPIs may use the same null distributions that were determined at startup. This

results in much greater runtime efficiency. The Clutter Classification was modified so that it would read in each range cell, identify the top five best fitting distributions and how well each identified distribution fit the expected value of that distribution.

The modified Clutter Classification code was then executed on CPI that accompanied Rome Lab's distribution of the parallel pipelined STAP. After minor debugging, the process seemed to work flawlessly. As one would expect, range cells that were near each other were very similarly distributed, and the identified distributions of range cells slowly changed as the Clutter Classification code progressed through the range cells. However, it is impossible to make any real statements as to whether this test is a success or failure. It seems that no one really knows what is inside of the few CPIs that were given to AFIT as part of the parallel pipeline STAP [AC99b]. It is not known what targets exist where in the data, nor is there a description of the clutter and terrain that produced the CPI in the first place. Therefore, the results of this test are at best inconclusive. However, it certainly seems to indicate that there is definite potential in this product.

3.2.1 Parallel Decomposition of Clutter Classification. As previously discussed, each stage of the pipeline is parallel in itself. To insure a seamless integration, as well as to explore the parallel nature of the Clutter Classification code itself, this product is ported to a parallel structure in isolation, and then integrated into the parallel pipeline as a new first filter stage. This parallelization is written to closely mirror the style that is used in the parallel pipeline application. All of the constants files and nomenclature used in the parallel pipeline are also used in the parallel Clutter Classification. It is also specifically engineered to accept MPI groupings and processor assignments in a similar manner to the STAP pipeline application [AC99a].

Each stage of the STAP pipeline is a task decomposition. Each stage accomplishes a part of the process, and then the data is passed to the next stage. Each stage is also parallel in nature. This is a data decomposition type parallelization. Here, multiple processors may be used for each stage to exploit the parallel nature of that stage to decrease the processing time incurred by that computation. In keeping with that design, the parallel Clutter Classification code is developed in the same manner. Specifically, it exploits a

data decomposition parallelism to decrease the latency of the computation. As shown later, there are very few communications necessary for this stage, and thus significant performance improvements may be achieved through this parallelization [VK94].

The first thing that is accomplished is the calculation of expected values for points in the distribution. This would be equivalent to the MSTAR procedure from the previous FORTRAN implementation. However, these Monte Carlo sets need only be created once. This is the fundamental difference between the new C implementation and the previous implementation provided by Rome Labs [US98]. They may be created upon startup, and then used for the entire duration of processing. Since this is the case, this part of the procedure does not influence performance of the algorithm in any fashion. It is not included in any run times, nor does it effect the integration of the two products. For these reasons, this process in not parallel. The master machine computes the entire Monte Carlo set itself one time at startup. This is vastly different from the previous implementation. In Rome Lab's Fortran code, these Monte Carlo estimates were the only parts of the algorithm that were parallel. However, running this section in serial, and only doing it once results in dramatic increases in performance [US98].

Once these Monte Carlo sets are used to calculate an expected value of their respective locations in the distribution, these expected values are passed to all of the slave processors active in the run. Even though it was quite a computationally complex process to calculate these values, the results are quite small and easy to distribute. Even if they were large and complex this would be irrelevant, because it is only done once during startup and never repeated. It is also important that each machine use the same set of expected values. Since these values are only good guesses at the true expected values, using the same values insures consistent results across the machines. It is not a problem if a machine incorrectly identifies a distribution. However, it is extremely important that all machines incorrectly identify like distributions in the same manner. Consistent results are much more important than accurate identifications.

Once these values have been passed, the real parallel execution of the program may begin. The parallelization relies on the fact that the distribution identification of any range cell is completely independent of the identification of any other range cell. Therefore, there

is no reason that any of the range cell distribution identifications be accomplished local to any other range cell. Clearly this attribute lends itself to the parallelization of this process. The size of the range cells and the number of range cells in a CPI is known a priori. Therefore, each processor calculates its share of the range cells to be processed. If the number of range cells to be calculated is not evenly divisible by the number of processors, the processors with the lowest rank each take an additional range cell, until all range cells are accounted for. Now, based on processor rank [PP96] as well as the knowledge of how many range cells each processor must calculate, every processor may determine exactly which cells they must calculate. This is shown graphically in Figure 3.2.

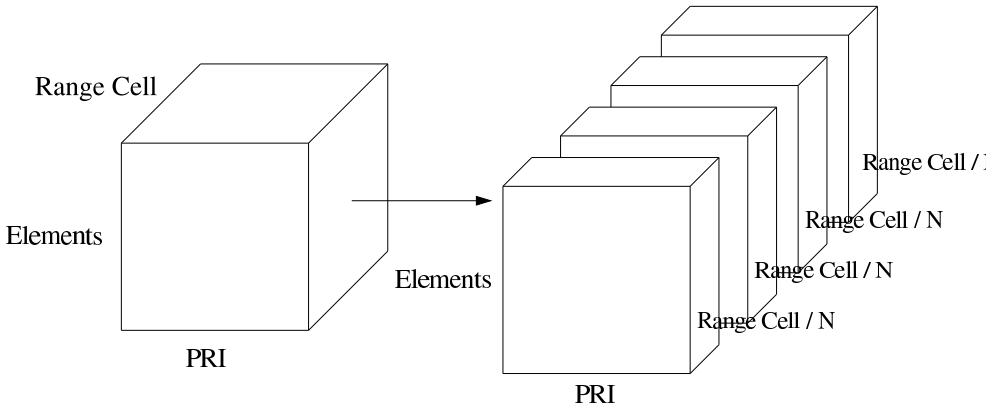

Figure 3.2 Data Decomposition in Parallel Clutter Classification

Now that each process knows which range cells it must calculate, it retrieves these range cells from disk. There are several different formats that this product may read, including binary, Matlab, and ASCII text. Each process only reads the range cells from the file that are needed at that location. This attribute is also a useful feature in the light of parallel design. Once all of the locally pertinent data has been retrieved, the actual computation phase begins. Each processor calculates the mean of all the data in each of its range cells. This is stored and used later by the root process. Each processor also runs the PAREST procedure and calculates the five best fitting distributions according the the Monte Carlo estimates calculated by the root processor [PP96]. In this manner all of the range cells are identified by distribution and are processed for mean value. This is the majority of the computation involved in the algorithm. It is apparent by design that this system is inherently parallel and does scale well with additional processors.

Once the needed calculations are accomplished for each range cell, the results are passed back to the root processor. The root processor uses these results to determine some final characteristics of the range cell set. These attributes are dependent upon adjacent range cells, and thus are much more easily executed serially by the root processor [JG95]. However these computations are not intensive, and should only impact the scalability and overall performance of the parallel program minimally. It is extremely important to note that regardless of how many processors are used, the amount of data passed over communications lines remains constant. It may take somewhat longer, because the master process must communicate with more processors, however, each processor will send smaller data sets. The root processor determines how many of the five best fit distributions each range cell has in common with its adjacent neighboring range cells. It also compares the means of each range cell with neighbors. These results are written to an output file for analysis. However, in the next iteration of the program, they serve as a basis to determine which data cubes should be sent to the STAP pipeline, and which range cells should be disregarded entirely. This entire process is depicted in figure 3.3.

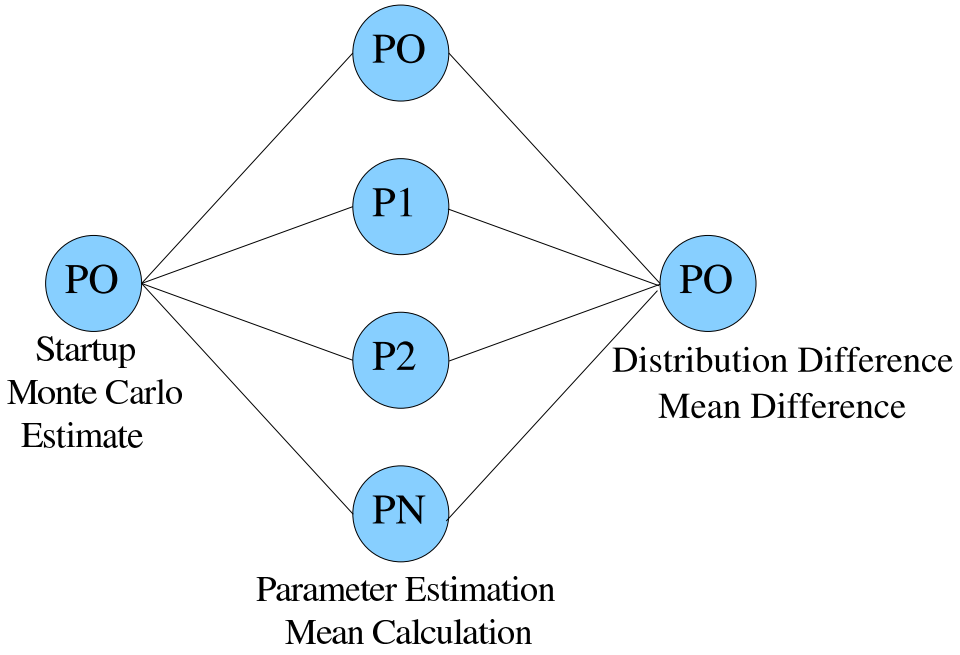

Figure 3.3 Parallel Clutter Classification Runtime Structure

3.2.2 Clutter Classification as a Non-Homogeneity Detector. One of the most promising aspects of the Clutter Classification code is its ability to detect changes between range cells. As previously discussed, a target is nothing more than a shift in the value of the mean of that range cell. Clutter Classification may offer more to that original hypothesis test than one ever expected. STAP relies on the assumption that every range cell is independent and identically distributed. In the real world, this is clearly not the case. If this were the case, there would be no need to determine which PDF a range cell followed, because it would always be the same as the next. This is not the case in reality, because of the changing landscape from range cell to range cell. This is what is known as non-homogeneous clutter. STAP works by determining the expected value of clutter at each element in the range cell. If data from a differently distributed range cell is used to calculate this expected value, it skews that value in error. For example, if the first half of the range cells come from over water, and the second half of the range cells in a CPI come from rugged mountains along the coast line, this manner of evaluation may become particularly erroneous. Expected values of the clutter over the sea may be determined by using clutter returns from the mountains. This clearly adds more clutter to the sea that should be expected. Furthermore, the clutter from the sea is used to determine the expected clutter over the mountains. This may weaken the expected value of the clutter and may results in insufficient suppression of that clutter [MW00].

These STAP weaknesses leads to the new technology of a non-homogeneity detector. In the past, non-homogeneity detectors have been used to isolate range cells that are particularly different than the other range cells in the CPI. These range cells were essentially discarded when calculating the weighting matrix [RM01] [MW96]. Therefore, these range cells that were non-homogeneous would not improperly skew the weights for the range cells. This has been met with mixed results. In some cases it improved the effectiveness of STAP, while in other cases it actually decreased. This method also has the potential to produce false positives. For example, if there is a large shift in the mean in one location, it is regarded by STAP as a target, regardless of what actually caused that shift in the mean. This shift in the mean may have actually been caused by a change in clutter rather than the addition of a target. This shift in clutter may also be associated with a shift

in the underlying distribution. Any time this assumption of IID is broken, STAP suffers and possibly produces erroneous and unreliable results. However, by using the Clutter Classification methods, it may be possible to determine if the there is really a shift in the mean caused by the addition of a target, or if there is merely a shift in the mean cause by changing clutter patterns. Again, a non-homogeneity detector does not aim to detect a target, rather it aims to discover locations in the data cube that are significantly different from their surrounding range cells

To further pursue this possibility, the Clutter Classification is further modified to test this hypothesis. Clutter Classification may be used as a non-homogeneity detector. The new Clutter Classification software reads in range cells just as it had done previously. Each range cell is assigned the top five best fit distributions. These distributions are saved for each range cell. Once completed, a comparison is done to see how many best fit distributions in common each range cell has with the previous range cell. If the returns share many distributions in common, it may be a good assumption that those range cells are indeed distributed alike, and have similar clutter returns. However, if they are not, that may signify that there has been a significant change in the they way the clutter is shaped in those range cells, or the addition of a target in that range cell changed the shape of the returns significantly enough to change the best fit distributions. In this case, a change in the mean of those range cells may not be a clear indication of a target, just a shift in the clutter return. In either case, it is clear that this location may now be labeled an area of interest, because of the anomaly found there. The number of distributions that a range cell has alike with previous range cells serves as a metric for homogeneity. This is a scale from zero to five, zero meaning that there were no distributions in common with the previous adjacent range cells, and five meaning that all of the best fit distributions for both of the adjacent range cells are exactly the same as the current range cell. It is also useful to have a scalable metric for measure of non-homogeneity, or heterogeneity. For this, a bias is added to best fit distributions in common count. The inverse of this sum serves as a scalable metric for homogeneity that returns values from $\frac{1}{bias}$ to $\frac{1}{bias+5}$.

To create a measurable test for this non-homogeneity detector, a different test data set is used. The MCARM data set is used, because it is very well known. The clutter

that exists in this set has been studied thoroughly. This allows a test to be constructed that may yield measurable results. The data set that is used is also substantially different than the data arriving with the parallel pipeline STAP application. This required further adaptation of the Clutter Classification code. The data cubes from the MCARM data set were shaped differently. They were dimensioned $Antenna \times PRI \times RangeCells$, with $N = 30$, $M = 128$, and $L = 630$. Not only were the data sets different sizes, they also consisted of different elements. Each element consisted of two double floating point numbers, the real part and the imaginary part. Each of these elements contained values on the order of 1×10^{-5} to 1×10^{-1} [TH01]. This is a stark contrast to the range of values, an excerpt is contained in Appendix B, that were in the parallel pipeline specific data sets [AC99b].

Another major difference between this non-homogeneity detector and other previous implementations is the manner in which it determines what a non-homogeneity is in general. Most non-homogeneity detectors attempt to construct an expected value for all locations in the data cube. Then a comparison is made to see which range cells deviate significantly from that expected value. Not only is this extremely computationally intensive, but it is also does not account for shifting homogeneous clutter. For example, the scenario of a land and sea transition is used again. The clutter over the sea is very homogeneous, and the clutter over the land is homogeneous as well. If a traditional approach were used, the total expected value would be calculated, and then applied to range cells in some manner. Clearly in some locations expected values derived from land are used on sea range cells and vice versa [MW96]. This has the potential to produce horrendous end results. The new implementation of the non-homogeneity detector merely looks for running changes in clutter, rather than calculating expected values over areas. In the land and sea case, there would be one range cell tagged as non-homogeneous directly at the land sea convergence. The other range cells would all be considered normal. This detector is geared toward detecting location of clutter shifts and transitions rather than non-homogeneities over an area.

Preliminary trial runs indicate that the Clutter Classification product is simply too sensitive to minor changes in distribution. Therefore, before serious test suites are estab-

lished, it may be useful to find a way to "turn down" the sensitivity of the application. Methods must be devised in order to highlight only the more egregious changes in distribution. Otherwise, it would be nearly impossible to determine which range cells were truly much different than the previous range cells, and which ones were only slightly different than adjacent returns. This is accomplished through a windowing adaptation to the Clutter Classification product. In order to ensure that only vastly different range cells are be flagged, changes are made that force the data to appear more homogeneous to the distribution identification system. A scale factor is added to the parameters that allow a fraction of the adjacent range cells to be used in the calculation of the distribution for the current range cell. Depending upon this scale factor, it is now possible to ensure that a portion of the data set is exactly the same as in the previous distribution identification procedure. This overlap results in much clearer delineation between severely non-homogeneous range cells. This windowing procedure is depicted in Figure 3.4.

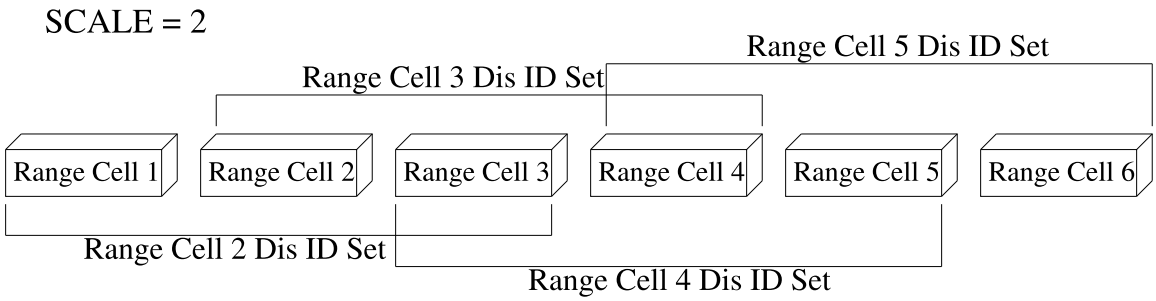

Figure 3.4 Clutter Classification Windowing Option

This windowing adaptation alleviates the effects of minor changes in distribution. This window is also adjustable, it is read in as one of the parameters in the startup file. Not only does it decrease the unreliable output, but there are also several very interesting side effects to expanding the window to overlapping range cells. This is discussed in the next section. This new change is much more useful in exposing areas of clutter that do not resemble their surroundings. In the general case with the scale factor set to two, the distribution identification of a range cell contains $\frac{2}{3}$ of the same data points as the previous distribution identification. Therefore, the new $\frac{1}{3}$ of the elements must be drastically different from the elements that were replaced in order to result in a change of

best fit distribution. This scale may be adjusted to increase or decrease the sensitivity of the product.

3.2.3 Clutter Classification as a Target Detector. It is also possible to mutate the non-homogeneity detector in the previous section in an attempt to detect target presence. According to STAP theory, a target may be interpreted as a shift in the mean of returns across range cells. This theory relies on a somewhat flawed assumption that every range cell is independently identically distributed. Clearly this is not the case in every circumstance. It may be much more accurate to say that a shift in the mean of like distributed cells is an indication of target presence. Using the Clutter Classification method it may be possible to detect mean shifts and changes in distribution caused by the addition of a target rather than a change in clutter.

The previously discussed scaling property is very useful in creating this target detection system. By no means is this target detection system perfect, there are severe limitations. However, it is extremely promising and clearly deserves much more thorough research. Because of the scaling of this product, when the scaling factor is set to two, only $\frac{1}{3}$ of the range cells change for every range cell evaluation. Therefore, it requires a significant change induced by these new elements to change the underlying distribution. For ease of explanation, a trivial example is illustrated. Assume that every range cell is identically distributed. The clutter is perfectly homogeneous, and there are no changes with range. A target is injected in a particular range cell. After the target has been injected into the system, the data set is analyzed with the clutter classification product with the scale set to two.

The first time that the different target values are encountered is on the evaluation for range cell directly before the target range cell. The addition of these target values may change the distribution of the range cell evaluation. Hopefully it is differently distributed than the range cells preceding it. Next, the target range cell is evaluated. It has a high homogeneity reading, this is because the new $\frac{1}{3}$ elements are nearly identical to the $\frac{1}{3}$ that were discarded. The range cell after the target range cell also looks the same, because the different target values are still in the test set. Only on the $target + 2$ range cell is another

shift be observed when the target values are again eliminated from the test set. Therefore, it may be possible to detect targets by looking for a shift in the entering and exiting of target values for each range cell. This is graphically shown in figure 3.5.

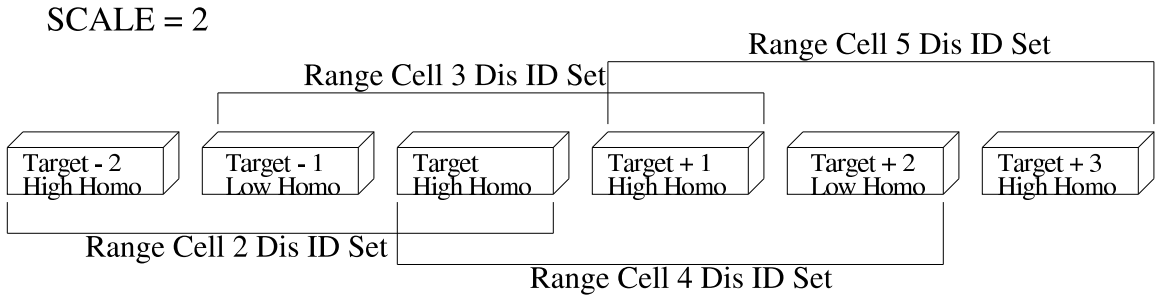

Figure 3.5 Target Detection Using Clutter Classification

Again, it is desirable to create some metric that exposes these scales distribution changes more effectively. According the the scale parameter, proper entry and exit points of the data set are determined. These critical points are summed, and biased to prevent division by zero errors. The reciprocal of this value serves as a metric where higher values indicate a higher probability of target existence. Using this metric, it is possible to analyze the results of the scaled clutter classification product.

3.2.4 Clutter Classification in the STAP Pipeline. Once the Clutter Classification had passed out of proof of concept type testing, the task of integrating it into the parallel pipeline was started. This was the most time intensive phase of the integration process. Since the STAP pipeline is so well tested and understood, it is clearly a better decision to modify the clutter classification code to the standards of the STAP code rather than the converse. The STAP parallel pipeline may be thought of as both a task decomposition and a data decomposition [VK94]. This is because the pipeline model decomposes the process into different tasks, where each task is handled by a different stage of the pipeline. However, it may also be thought of data decomposition inside each parallel stage of the pipeline. This is because the actual data that is being manipulated is spread across a group of processors that make up a certain stage. The processors in that stage are completing the same operations on differing subsets of the data. The STAP product communicates via standard MPI communication software. This method of parallelization requires that

each processor in the cluster be running the same binary. Therefore, there must be control structures in the binary that allow each node to know which subsection of which stage for which it is responsible. Keeping all of this overhead strait proved to be the most difficult portion of the product integration.

The STAP process is controlled by a case statement in the main binary. When this binary is executed, each node starts executing as a certain subsection stage node based on the identification it is assigned at runtime. As previously discussed, the original first stage in the pipeline is the Doppler filter. There are several things that occur in this stage that are not specifically related to the Doppler filter. For example, this is the stage where the CPI are read in from disk or created randomly. This may no longer occur in this stage; it must be migrated to the new first Clutter Classification stage. The Clutter Classification algorithm is added to the large case statement, and begins executing its code with its own smaller group of processors. Because of the temporal dependency in the weight calculation, it is necessary to save a few CPI locally in the Clutter Classification group. If this new first stage finds something of interest in any range cell, it is sent to the rest of the STAP pipeline, along with temporally local data sets in order to calculate proper weighting matrices. This actual integration is discussed in detail in Chapter IV.

3.3 Design Summary

In conclusion, this chapter describes the integrated product. Unfortunately, neither application was initially well suited for integration. The STAP application is well written and seems to behave as expected at first glance. Therefore, the Clutter Classification is integrated into the STAP pipeline, rather than vice versa. The Clutter Classification was initially written in FORTRAN rather than C, it was poorly written, and it did not produce correct results. Therefore, most of the changes are made in this application in preparation for integration. This chapter also covers the design of the final product and some of the basic issues that are addressed in the integration, which are more fully discussed in the implementation chapter. It has provides a framework and foundation for the actual low-level implementation of the integrated application, and it covers some changes and optimizations that are made to each application alone, as well as integration

issues in general. This chapter has also covered some other possible applications that may be formed from the new Clutter Classification technology.

IV. Implementation

Upon completion of separate product optimization and testing, and integration design, the implementation and completion of the STAP/Clutter Classification can be completed. These two products integrate well, but they are significantly different in enough areas to make this an extremely complex task. This task can be completed in increments, such that functionality is steadily added until the entire integration is complete. In this manner, it is possible to maintain a correct functioning executable through the entire process. This is extremely useful for debugging and management reasons [RP01]. This chapter is a discussion of the difficulties encountered in the integration and how they were overcome. It also discusses several major incremental steps, as well as what factors influenced particular implementation specifics to accomplish the integration.

4.1 Integrating the Binaries

The first step of the integration completed is the integration of the Clutter Classification code to the STAP code. Initially, there is no communication between the two. MPI utilizes the Unix "R" [DG95] services to start many instantiations of the same binary on different machines. Each of these particular instantiations are given rank numbers upon startup [PP96]. As discussed previously, the STAP pipeline partitions the set of processors into groups and then each group executes a function in parallel. The initial step consists of adding a new call in the case statement that directs groups of processors to execute a specific function. This increment sets up an additional group of processors that are used for the execution of the Ozturk code. The number of processors allocated to these groups is determined from a startup file called proc.dat. Rather than having seven pipeline stages, the improved product now has eight stages that must be accomplished. However, other than operating in the same binary these two products are still entirely separate at this point in the build; they do not collude in any manner. This increment is shown in Figure 4.1.

Both of the original products require a substantial amount of startup parameters that are provided in a text file. These parameters are for the actual requirements in the

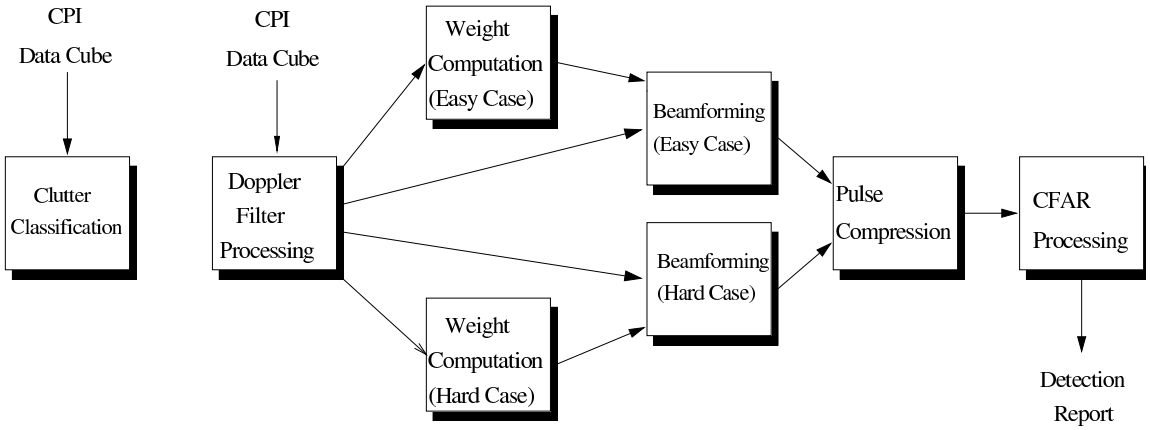

Figure 4.1 Application Integration – Increment 1

mathematical computations, rather than setup in the parallel environment. Previously, each product had its own startup files. Some of the parameters were the same, and some of them were not. The next stage in the integration was to create one universal startup file that would contain all of the parameters that were in use in the entire process, contained in Appendix C. When parameters were read in, they were stored in a parameters structure. This structure was changed in such a manner that it contained all of the parameters needed by both products. The parameters that were common to both were given common names, and the code of the Clutter Classification product was changed to match those of the STAP application where differences were encountered.

4.2 Passing the Data Cube

Once the two separate binaries were combined to one, the brunt of the work began. Passing the data cube from one set of processors to another set of processors turned out to be a non-trivial task in itself. The Clutter Classification code would initially process the data, and then pass the data to the rest of the pipeline. There was no effort to decrease the workload at first, the object was merely to integrate the Ozturk code into the rest of the pipeline and validate that it did indeed behave as expected. The first iteration of this data passing alleviated some of the complications of the parallel pipeline. However, even at first sight, it was clearly a severe bottleneck. Each stage may have different numbers of processors allocated to them. Therefore, the transfer of data from one processor to another

processor is not a one-to-one and onto [KG96] straight across transfer. One processor may have data that needs to be delivered to several different processors; in the same token, one processor may need to receive data from several different processors. The first solution to this problem was to forgo any calculation, collect the data, send the data, and then redistribute the data. This is illustrated in Figure 4.2. However, this solution was clearly inadequate and immediately abandoned.

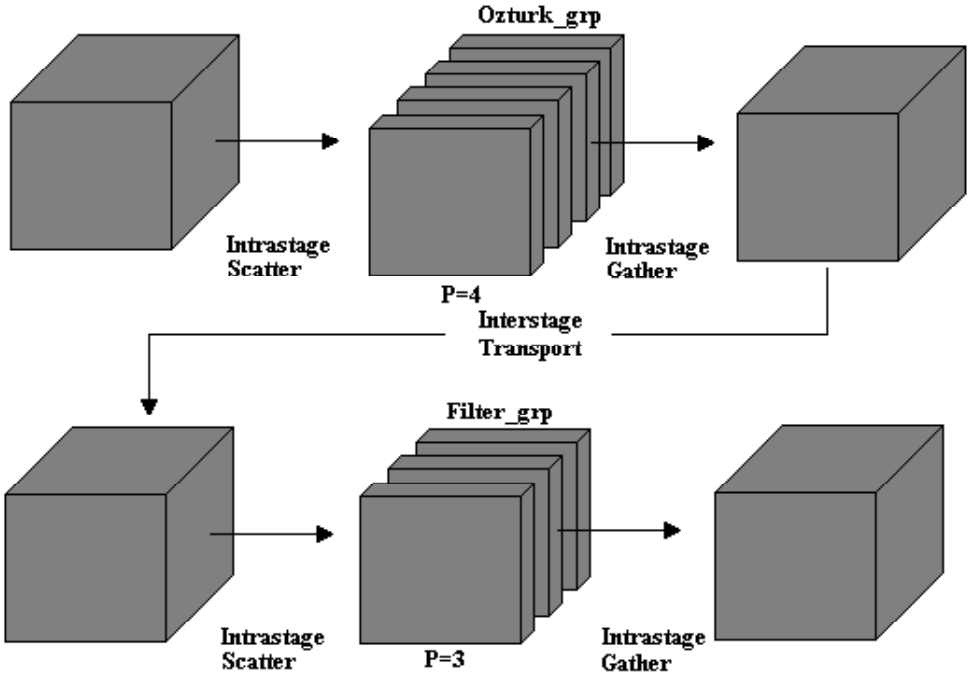

Figure 4.2 Non-Distributed Data Passing Method

One must also expect a very large disparity in time between the computation phase of the Ozturk filter and the other running stages. There are several reasons that this may be the case, and they may be adjusted accordingly later in development. One must keep in mind that the Ozturk program was designed with double precision floating point numbers, 64 bits, where as the rest of the STAP pipeline uses short integers, 16 bits. Not only are the data significantly shorter in the STAP pipeline, the data types are also much different. Floating point operations are much more complicated than integer operations. These and other issues are addressed in following increments in the process. The immediate task is simply to test the concept, and then incrementally optimize the product.

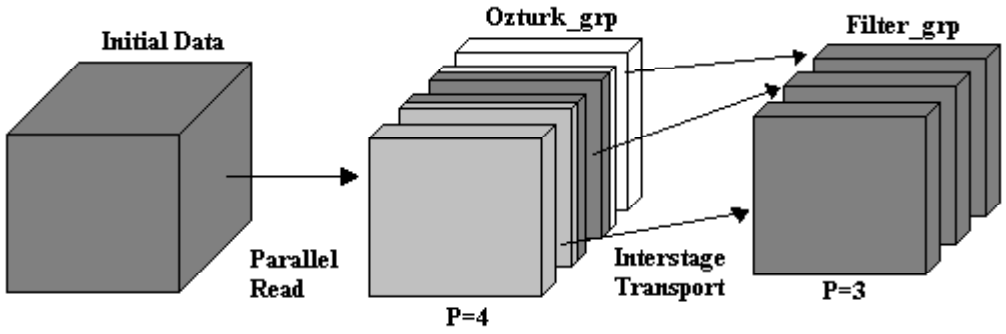

Figure 4.3 Distributed Data Passing Model

The next solution was much more complicated. A method was created in which every processor in the Ozturk phase had the ability to determine where the data it possessed needed to be sent in the next stage of the pipeline. In the same manner, each processor in the Doppler filter group calculated how much data it needed to receive from each processor in the Ozturk stage. In this manner. there was no need for a centralized control, nor for the collection and redistribution of data. Initial empirical evidence suggests that this is a much more efficient viable solution. It is clearly a usable correct implementation that serves as a model for further development in the integration process. In this implementation, the sending processor iterates through all of the range cells that it controls. It calculates which processor in the next stage needs that particular range cell and then sends it. Each range cell is sent with a separate send, and received by the Doppler filter group by a separate receive.

The third and final approach that was implemented was very similar to the second approach. However, rather than iterating through all of the range cells, and then passing them to the correct corresponding processor in the next stage, each processor would calculate which range cells needed to be passed, and then send them in one large send. This should reduce the overhead that is encountered with 512 different sends. A graphical depiction of this method shows this distributed approach to passing data between the two stages in figure 4.3.

4.3 Ensuring Viability After Integration

Once this data passage scheme was completed, It is now possible to ensure the viability of the system in general. Several test runs were completed to validate that the system still operates as expected. The STAP results should be exactly the same as before the Clutter Classification addition. This is because there is really no change in the system. Even though the data was evaluated with the Ozturk method, no data was omitted or altered. Therefore, everything should still function as normal. The performance measurements however, do change significantly. This is because of the additional overhead injected by the integration of an eighth stage into the pipeline. These performance returns are discussed in later sections. This validation is a qualitative look at the STAP output, rather than analysis of throughput and latency which has been the main focus throughout this research. However, if the actual qualitative output of the product is poor, the throughput and latency of the product is meaningless.

Initially there were some issues involved with the PRI stagger approach. The Clutter Classification does not only need to identify and pass data cubes, it must stack one data cube atop the next, and then pass the larger data structure to the Filter Group Stage. This was discovered during the qualitative analysis of the integrated product, because the results of the analysis were not the same as the non-adapted product. Half of the passed structure consisted of random initialization data rather than the needed radar data. However, after this was resolved, the results of the modified product with the additional stage were qualitatively identical.

4.4 Optimizing the Integrated Product

The fundamental premise of this application and research is the ability to use the Ozturk method as a front end filter for the STAP application. It is desirable to have the ability to discriminate between data sets that may have something interesting in them and data sets that do not have something interesting in them. In this manner, data cubes that clearly do not have any interesting return contained in them may be discarded on the front end, and thus save a substantial amount of processing that is required in the STAP pipeline. Therefore, now that the two products have been integrated, the task

remains to intelligently pass only the needed data cubes to the STAP pipeline. First, the Clutter Classification code was altered to determine if a target may be present. Before this modification, the application only produced a series of qualitative results. There was no method to determine target presence. That task was left to the operator to read and analyze results. However, after this modification to the application, the output of the application after processing each data cube was a boolean value. This boolean value provided an answer to the question: "Does this data have high probability to contain targets?" If the boolean value was yes, then the clutter classification would pass the data to the STAP application. If not, then the clutter classification would simply discard the data as non-useful.

In the STAP application, the weighting matrix is calculated using data from previous data cubes that were passed through the pipeline. This relies on the assumption that data cubes that are adjacent to one another nearly always have the same clutter and noise characteristics. In general this is a safe assumption. However, in practicality it is not always the case. Furthermore, the addition of the clutter classification to the STAP pipeline has severely aggravated this situation. Supposing that several data cubes in a row are discarded, (a very common occurrence), the weight matrix for the next target rich data cube has been calculated from a previous data cube that is very distant temporally and spatially from the current data cube. This can severely hamper the quality of the noise nullification through the use of an incorrect weighting matrix. Therefore, the pipeline and clutter classification code was modified to account for this. Whenever a target is encountered, not only must that data cube be passed, but the two data cubes before it must also be passed. In this manner, one may ensure that the correct data cubes are used to calculate a relevant weighting matrix.

Logic was also added to the Clutter Classification stage to ensure that redundant data cubes are not passed to the STAP pipeline. Suppose that there were two adjacent data cubes that both contained targets, (likely a very common occurrence). Without some logic, the first cube, along with the previous two are passed to the STAP application. At the next CPI, which is also target positive, it is passed, along with the previous two, to the STAP pipeline. This results in sending the same cubes multiple times. This is clearly

an undesirable trait. Therefore, a window of data cubes is maintained at the clutter classification level. Information on when these cubes have or have not been passed is also maintained. When a cube needs to be passed to the STAP pipeline, a check is made to ensure that the proper cubes have been sent. If they have, there is no need to re-send; if they have not, they are sent to calculate correct weights. The STAP application must also be adapted to dismiss the actual results of the STAP pipeline on these filler cubes. These filler cubes are incorrectly processed, and then the results of these cubes discarded. This is not wasting time and computation power, because of the pipeline structure. Therefore, a boolean is associated with each cube that is passed. This boolean represents whether the data cube in question is to return viable results, or whether it is to be discarded upon completion of the weight computation.

It appears that the application is now completed, even if there are many actual coding optimization that need be completed. The application does appear to function well, and serves as a baseline for this proof of concept. One of the biggest weaknesses of this implementation is the fact that the speedup achieved by this application is now completely dependent upon target density in the scanned region. This is one characteristic that is not desirable in a real time system, but it may have to be accepted. One would certainly like a product that behaves the same regardless of conditions. However, if one considers the most cluttered/target rich environment that may realistically be encountered in operation, and then tunes the application to handle that many targets, this application may still prove viable for real time processing. Therefore, it is certainly useful to mathematically model this behavior so this tuning may be accomplished. The processing power of the unoptimized STAP application is given in Equations 4.1and 4.2. These are given in terms of latency and throughput, the most critical performance metrics of this real time system.

$$Latency = Stages \times Max\{cost(stage1), cost(stage2), \ldots, cost(stageN)\} \tag{4.1}$$

$$Throughput = Max\left\{cost(stage1), cost(stage2), \ldots, cost(stageN)\right\} \quad \text{When pipeline is full} \tag{4.2}$$

These metrics are simple in the original STAP application, and they are consistent, as long as the number of stages and the cost of these stages remain constant. However, these metrics become significantly more complicated after the product integration. As shown in Equations 4.3, 4.4, and 4.5, the latency of any specific data cube is now dependent upon whether that data cube contains any possible targets. The actual throughput of the application has changed even more dramatically than the latency metric. Also, note that both are now dependent upon the characteristics of the data that is encountered in the actual operating environment.

$$Latency = \begin{cases} Stages \times Max\left\{cost(stage1), cost(stage2), \ldots, cost(stageN)\right\} \\ \quad \text{If target found} \\ cost(ozturkfilter) \\ \quad \text{If no target found} \end{cases} \tag{4.3}$$

$$\begin{aligned} AverageLatency = \; & DiscardRatio \times Stages \times \\ & Max\left\{cost(stage1), cost(stage2), \ldots, cost(stageN)\right\} \\ & +(1 - DiscardRatio) \times cost(ozturkfilter) \end{aligned} \tag{4.4}$$

$$\begin{aligned} Throughput = \; & DiscardRatio \times \\ & Max\left\{cost(stage1), cost(stage2), \ldots, cost(stageN)\right\} \\ & +(1 - DiscardRatio) \times cost(ozturkfilter) \end{aligned} \tag{4.5}$$

The throughput of the new integrated application is very close to Equation 4.5, however this is an approximation. There are some complexities involved in the discarding of data cubes. The first part of the equation accounts for the cubes that must pass through the entire pipeline. The throughput calculation assumes the the pipeline is entirely full.

This is certainly not the case in reality, (if it were, the Ozturk filter would be worthless). However, the bubbles that are injected into the pipeline to replace dropped data cubes are accounted for by the complete processing of other data cubes in the Ozturk stage. Therefore, to make this application even remotely efficient, there are two issues that must be addressed:

- Latency – To improve latency, the higher the discard ratio, the lower the latency for a greater section of input data. Even if the Ozturk stage is not significantly shorter than the longest stage, latency is dramatically improved for the data sets that do not contain targets.
- Throughput – To improve throughput, the cost of the Ozturk filter stage must be significantly smaller than the cost of the longest stage. Also, the discard ratio has to be high, such that many data cubes are allowed to complete in the smaller cost of the Ozturk filter, rather than the high cost of the longest pipeline stage.

4.5 Optimizing Code for the Integrated Product

Once a functional model application has been created, the goal is to optimize that proof of concept platform to increase its efficiency and effectiveness. The first issue to be addressed is the size and type of data utilized in the STAP application. Currently, the STAP pipeline functions on short integer types. The Linux clusters, (AFIT Heterogeneous or Homogeneous Cluster), in use consists of Intel or AMD chips that implement a short integer as a 16 bit integer. The Ozturk Clutter Classification application represents its data as 64 bit IEEE floating point numbers. This is a large disparity in size and operational complexity, and it is shown in the baseline results found in chapter VI. In preliminary trials, the clutter classification usually runs an order of magnitude or more slower than the other stages of the pipeline. This is a very undesirable trait, especially when considering the above mathematical representations of latency and throughput. If the clutter classification stage remains considerably longer than the other stages, both throughput and latency are severely hampered.

One solution is to change the data types found in the different applications to match each other. The Ozturk stage is required to work on a much larger and complex data type, convert it, and then pass it to the next stage. Therefore, one must decide whether to make the Ozturk code match the STAP pipeline, or the STAP pipeline match the Ozturk code. Previously, all modification were mainly to the Ozturk algorithm to make it match the more complicated STAP application. However, it may be more intelligent to change the STAP application in this scenario. As explained in the STAP background section, the goal of the parallel pipeline is to create an application capable of processing STAP data cubes real time. The usage of short integers rather than double floating point numbers is clearly an effort to speed up that application as much as possible. However, there is also a degree of accuracy that is lost when this is done, especially when comparing the actual data values of the true MCARM data and the values of the data file that were given to AFIT as part of the STAP application.

When considering the increased performance capabilities of modern computers it may be the case that changing these short integers to double floating point numbers still yield the same real time capability and also offer a much greater degree of quality in the output of the application than was available using supercomputers available during the creation of this application. Therefore, the STAP application is changed to use double floating point numbers. This increases both the computational complexity of the algorithm and the communication time needed between parallel stages of the application. This may also make it possible to use some of the true MCARM data, rather than the small data set that accompanied the STAP application deliverable. The results of this change are contained in Chapter VI, Results and Analysis.

Even though is makes more sense to alter the STAP application for the different data structures, changes were also made to the Ozturk filter to accommodate the smaller data type. This should results in decreased run times associated with the clutter classification stage of the integrated application. However, it is questionable how drastic this speedup may be. Only a portion of the operations in the application are actual operations on the data type changed. Therefore, the speedup only affected a minimal portion of the code. Bookkeeping, communication overhead, and other computations are unaffected by

the alteration. Therefore, it makes sense to keep the changes made in the STAP pipeline, and discard the changes made to the Ozturk filter. With minimal additional costs, it is possible to give the STAP pipeline a much finer grain data set, and create a uniform data type across the entire application.

After this increment is completed, it is necessary to test the optimization to validate whether or not is is a viable front end filter for the integrated application. It is known that the Ozturk filter scales linearly with respect to number of processors. Therefore, a test is conducted that allocates many processors to the Ozturk filter to determine if it runs faster than the other stages in the pipeline. The results of these tests are contained in chapter VI. If this change does not increase efficiency enough, the actual complexity of the filter must be changed to something more congruent with the other stages of the pipeline. Therefore, changes are made to decrease the complexity of the algorithm in general. However, it must then be determined if the results are still qualitatively viable. It appears that there is a trade off between efficiency and effectiveness. The task is to make the application as efficient as possible without rendering the results ineffective. This results in two basic questions, the second dependent on the first:

- Is it possible to decrease the complexity of the filter to give it similar run times as the other pipeline stages? What is the minimum amount of work that the Ozturk filter can be required to do, and how can this reduced complexity be implemented? If it cannot be done in comparable run times, this effort must be abandoned completely.
- If it is possible to decrease the run time of the Ozturk filter within bounds of the other pipeline stages, is the quality of output of the filter sufficiently effective? This is a much more complicated test, because the metrics for quality output are much more subjective than in the previous question.

The fundamental premise for use of the Ozturk filter was its ability to identify distributions base on relatively few observances. As discussed in Chapter III, initially, every single data point in every range cell is used for identification of the data set. After initial testing, it was found that using every point from the single data set alone was too sensitive, and failed to show only great changes in the continuum. To alleviate this, a window was

applied over top of surrounding range cells. In this manner, the sensitivity was decreased, but this lead to an extremely large data set. It may be possible to rethink this implementation of the Ozturk filter in the light in which the method was developed. Two different complexity reductions were conducted and analyzed both quantitatively and qualitatively, to understand which results in better output:

- Remove the Scale – The scale option was reduced so that only data from the original data cube would be used. In reality this results in a reduction in the data size by 66%. However, it is very likely that this optimization also results in a drastic reduction in effectiveness.
- Small Sample Across Range Cells – This option also reduces the size of the data, however, it still incorporates data from different range cells. In this manner, the user requests the percentage of the range and scale to be incorporated in the Ozturk evaluation. That percentage of the data is chosen at random and analyzed. It is hoped that this reduces the complexity of the filter and alleviate some of the ill-effects encountered in the single range cell analysis

4.6 Implementation Summary

In conclusion, the fourth is the actual integration implementation. It discusses how the two prepared applications were integrated into one program. It contains the details of the compilation and intricacies that were overcome during the integration process. It is arranged in an incremental fashion that conveys the software engineering process that is used to systematically build the integration one phase at a time, as well as the testing that ensured the results of this integration were accurate in each increment and after each optimization. Upon completion of a baseline model, this chapter discusses several of the possible optimizations and complexity reductions that may serve to increase the efficiency of the integrated product, while limiting the decrease in effectiveness. The results and analysis of this integration and optimizations follow in chapter VI.

V. Design of Experiments

Chapter V discusses the design of scientific experiments that provide justification and empirical evidence to support the original hypothesis and conjectures. The first and second sets of tests are specifically testing particular parameters for the separate application implementations of STAP and Clutter Classification. These tests are aimed at parallel performance and quantitative results. The theory behind these applications has been well researched and discussed [SL94] [JW94], and is beyond the scope of this research. Therefore, a focus is maintained on the parallel high performance aspects of these particular applications in isolation. The third set of tests are different qualitative tests of the Clutter Classification application. There are several assumptions that were made about the the Clutter Classification application. Most of these assumptions deal with the applicability of certain statistical properties to range cells with and without targets, as well as how these properties change when targets are added to these range cells. Again, this is not an attempt to prove that the application is qualitatively viable, however, they do provide empirical evidence that would seem to support the assumptions made. The fourth set of derived tests are for the integrated STAP Clutter Classification pipeline. These tests deal with both qualitative and quantitative aspects of the new integrated product.

5.1 STAP Experimentation Design

The Design of the STAP experiments are designed to explicitly test quantitative results. This reason for this is two fold: the data sets originally sent with the Rome Labs STAP are not and well known, and the application and qualitative tests have already been conducted in many different research efforts [JW94]. The problem at hand is to implement the well known mathematical algorithm in a more efficient manner. The first set of tests that are conducted are baseline test environments. These test environments serve as a starting line for comparison to measure speedup and throughput after parallelization and optimization. The test environments are created on the heterogeneous AFIT cluster. Three baselines were created:

1. Slow Test Environment – This environment was made with the slowest processors available on the AFIT Heterogeneous Cluster. These machines are Pentium III processors running at 400MHz. This is a worst case run. Assuming that one of these processors is in use, it is highly probable that all of the processors in that stage run as slow as this slowest machine.

2. Medium Test Environment – This environment was created with the medium speed processors available on the AFIT Heterogeneous Cluster. These machines are Pentium III processors running at 600MHz. These machines should provide a good "average case" performance baselines.

3. Fast Test Environment – This environment is made up of strictly the fastest processors that are available in the AFIT Heterogeneous cluster. This serves as a best case scenario. The processors in this benchmark consist of Pentium IV processors running at 1.7GHz.

Once the benchmarks were created, tests were run to exploit the parallel nature of the STAP application. Because of the heterogeneous nature of the AFIT Heterogeneous Cluster, the allocation of processors is extremely relevant. Placing certain processors in certain locations can either produce extremely good results, or it can absolutely destroy the results as a whole. Appendix D contains a genetic algorithm geared toward intelligent processor allocation optimization. To test the parallel nature of this application, processors are continually added to the slowest stages in the parallel pipeline until either the processor pool is exhausted or the addition of a processor results in degraded performance.

Once these tests are conducted on the AFIT Heterogeneous Cluster, the same tests are conducted on the AFIT Polywell Homogeneous Cluster, (a detailed system description is contained in Appendix E). Every node on this cluster is identical, and consists of a 1.2MHz Athalon processor with a 100Mbps Ethernet backbone. The processor allocation on this cluster is much simpler, and these tests highlight the benefits and simplicity of a homogeneous cluster.

All of these environment baselines are conducted a number of times. The reason for this is to expose any variability between runs. It is known that STAP is a very deterministic

process, and there should be very little difference in runs if everything else is held constant. The multiple runs validate this conjecture.

5.2 Clutter Classification Experiment Design

The Clutter Classification experimentation is significantly more involved than the STAP testing. Not only are there quantitative aspects and optimizations to deal with, but other qualitative aspects must also be addressed. Furthermore, this parallel implementation is completely new. It is useful to decompose this experimentation to a much finer granularity to understand which section of the clutter classification are the most intensive, which ones scale well, and which ones do not. Therefore, the quantitative experimentation for the Clutter Classification is decomposed into four main sections. There are three main sections of code in the parallel decomposition of the Clutter Classification that are tested, and one overall measurement. The experiments that are run follow:

Test 1 Parallel Read – This is where all processors read their section of data from the disk. This may be done in parallel and should exhibit substantial speedup. All things considered, however, it is a small portion of the overall computation time.

Test 2 Parallel Computation – This is where each processor actually does the parallel distribution identification of its assigned range cells. This very scalable, because no interprocessor communication is required, and should benefit greatly from additional processors. This section is also the most computationally intensive section of code by far.

Test 3 Serial Section – This is the section of the code that must be ran in serial. This section is the overhead that is encountered when the root processor must gather the information from the slave processors and determine some statistics about that data. This amount of work is not variable, and should remain the same regardless of processor count. However, there may be issues getting all of the processors synchronized to return data in an efficient manner.

Metric 1 Total Parallel Time – This metric is a total of the three main operations comprising the parallel architecture of the application. This included the parallel portions, as well as the serial and overhead. This is the final most important performance metric.

Once the parallel aspects of this application have been explored, the qualitative metrics must also be highlighted. To test the qualitative attributes of the system, empirical evidence is used to suggest that the addition of a target to a range cell does indeed change the best fit distribution of that range cell. This premise may or may not be true, and the statistical analysis needed to determine if this is the case is left for a comprehensive study in that area. These tests simply illustrate that the addition of a target does tend to change the best fit distribution in the specific test cases provided.

To qualitatively test this, targets are injected into well known range cells via a separate injection application. Targets are injected in multiple location with multiple parameters, (shown in Chapter D.4). These CPI were then used as a test set to see if the Clutter Classification application actually identified the specific range cells as possible target locations. When the quality of the results is low, modification are made to the application in an effort to increase the effectiveness of the application.

Just as was the case with the STAP tests, the Clutter Classification tests were also conducted on the AFIT Polywell Homogeneous Cluster. Again, this serves as a validation for the data that was collected from the AFIT Heterogeneous Cluster. There is no reason that the Polywell should come up with results significantly different than the first cluster. In this manner, the Polywell runs serve as both a sanity check and study of the clutter classification application implemented on the heterogeneous cluster vs a homogeneous cluster.

5.3 Clutter Classification as a Non-Homogeneity Detector Experimentation

As proposed in the application design Chapter III, it may be possible to use the Clutter Classification application as a stand alone non-homogeneity detector. Using this application, it may be possible to process a running set of range cells and determine if a particular range cell is significantly different than the adjacent range cells. To do this, an

experiment was created in which the structure of the CPI was known. In this CPI, there were known areas of non-homogeneity, as well as injected targets in specific range cells. This manufactured CPI, and others like it, were processed by the Clutter Classification application in an effort to determine where the non-homogeneities are in the particular CPI. There are several parameter combinations in the application that must also be tested. These parameters, as well as parameters of the injected material are changed to understand how the Clutter Classification application handled different setup parameters and injected non-homogeneities.

Success for this experiment is defined as accurate detection of a certain percentage of the known non-homogeneous range cells. This percentage is dependent upon several parameters and is fully discussed in the results section. Failure of this test can come in several ways as well. If it is found that some of the non-homogeneities cannot be found, or that certain injection parameters do not allow the non-homogeneity to be found, the test fails. This would tend to indicate that the fundamental statistical assumption is flawed, and that the addition of a target or non-homogeneity does not produce a change in the corresponding best fit distribution.

5.4 Clutter Classification as a Target Detector Experimentation

The Clutter Classification as a target detector is very much like the previous example of Clutter Classification as a non-homogeneity detector. However, there are several different search parameters that are now used instead of the ones previously mentioned. In this case, the object of the test is now to find only the targets, rather than the non-homogeneities in general. This complicates the test significantly, and still does not offer proof that this is a viable concept. It is merely evidence that does not contradict the statistical theory.

Just as in the non-homogeneity tests, several target CPI were manufactured bearing different characteristics. It is important to test the application with many different parameters and data sets. A success in this application is the ability to reliably determine which range cells in the CPI actually contain a target, instead of simple clutter, non-interesting returns, or unimportant non-homogeneities.

5.5 Integrated Product Experiment Design

The integrated STAP Clutter Classification algorithm is the main component of this research effort. The main objective of this study is to integrate these two applications in an effort to increase efficiency and effectiveness. To that end, an entire series of experiments were conducted in an incremental optimization approach. The first aspect to be tested was the transmission of data between the new pipeline stage and the rest of the pipeline. Several different methods of passing this data are tested for efficiency and effectiveness. The most efficient method is then augmented into the final application. Once this has been optimized to perform at the same or higher levels that the other data passing models in the pipeline, the focus shifts to the actual runtime aspects of the different pipeline stages in an effort to increase their efficiency and effectiveness.

The first test in this realm is a benchmark to compare the actual computation time involved for each stage. Once this benchmark has been established, testing is geared toward parallel aspects of the pipeline. The fundamental goal of this testing is to validate or invalidate the efficient use of the clutter classification as a first stage filter for the parallel pipelined STAP application. Incremental tests are conducted with the results of each being presented in the testing process. Once it has been shown that it is possible to use the Clutter Classification as a first stage filter, qualitative aspects of that filter are then tested. The quality of the results must be maintained. If the quality of the filter degrades as it is optimized, the quality of the filter must remain high enough to complete the assigned task effectively, otherwise the filter stage is useless, regardless of how fast it executes.

The third set of tests executed on the integrated STAP Clutter Classification application are geared toward exposing the attributes of a pipeline that has the ability to drop unneeded data cubes at run time. It is expected that the ratio of CPI that may be dropped from the processing chain greatly influences the efficiency of the new application. Tests are performed that indicate how the application performs when this metric, subsequently termed the discard ratio, changes in different test data sets. It must be determined if performance is enhanced enough to warrant the use of the Ozturk filter when the discard ratio is at a typical real world value.

During this research effort, many sets of tests are conducted as described in this chapter. To facilitate organization and ease of understanding, each of these test sets have been illustrated and given a specific number, noted in table 5.1 and 5.2. These tables decompose the tests into groups, and then into specific test sets. These identification numbers are used to reference each specific test set throughout this thesis.

5.6 Experiment Design Summary

In conclusion, Chapter V discusses design of experiments. This chapter discusses exactly how a scientific method was employed to design an experiment that tests the fundamental hypothesis of this research effort. These tests are developed in an effort to isolate one specific variable or evaluate one specific metric. These tests are also designed with an effort to concentrate on statistical significance, although in most areas, it is very difficult to imply any statistical significant qualitative results.

Table 5.1 Table of Tests and Experiments

Product	Test Environment	Test Specifics	Metrics	Test #
STAP	Heterogeneous	Slow Processors	Throughput Latency	Test 1
		Medium Processors	Throughput Latency	Test 2
		Fast Processors	Throughput Latency	Test 3
		Parallel Performance	Scalability Speedup Throughput Latency	Test 4
	Homogeneous	Parallel Performance	Scalability Speedup Throughput Latency	Test 5
Clutter Classification	Serial	Comparison with FORTRAN	Throughput Latency	Test 6
	Heterogeneous Cluster	Parallel Read	Speedup Scalability	Test 7
		Parallel Calculation	Speedup Scalability	Test 8
		Serial Section	Speedup Scalability	Test 9
		Total Runtime	Speedup Scalability	Test 10
	Homogeneous Cluster	Parallel Read	Speedup Scalability	Test 11
		Parallel Calculation	Speedup Scalability	Test 12
		Serial Section	Speedup Scalability	Test 13
		Total Runtime	Speedup Scalability	Test 14
	ANY	Inhomogeneity	Qualitative (Subjective)	Test 15
		Target Detection	Qualitative (Subjective)	Test 16

Table 5.2 Table of Tests and Experiments (Cont'd)

Product	Test Environment	Test Specifics	Metrics	Test #
Integrated Application	Heterogeneous Cluster	Transmission1	Wall Time	Test 17
		Transmission2	Wall Time	Test 18
		Transmission3	Wall Time	Test 19
		Transmission4	Wall Time	Test 20
		STAP w/ Long Float	Wall Time	Test 21
		Clutter Class w/ Short In t	Wall Time	Test 22
		Optimization 1	Wall Time	Test 23
		Parallel Optimization 1	Wall Time	Test 24
		Optimization 2	Wall Time	Test 25
		Parallel Optimization 2	Wall Time	Test 26
		w/ Oz Filter w/o Oz Filter	Wall Time Speedup	Test 27

VI. Analysis of Results

Chapter VI analyzes the results that were found in each of the experiments discussed in Chapter V. The organization of these results are in the same order as the tests discussed previously. Analysis is provided based on quantitative and qualitative results of testing designed in chapter V. To be consistent with the nomenclature derived in chapter V, each experiment discussed is designated with a specific test number that correlates the design and motivation for an experiment.

6.1 Original STAP Performance Observations

According to code refinement criteria, the existing STAP code is well written, and it has been tested, optimized, and analyzed in nearly every aspect [RP01]. It has been boarded on many different parallel computers with very positive results [AC99a], [WL99a], [WL99b], [SF99], and [WL99a]. To augment familiarization with the code, as well as its parallel runtime behavior; many experiments have also been conducted on the local AFIT clusters. These tests certainly give insight to the inner workings of the code and methodology. This section examines results obtained from STAP runs on the local AFIT Heterogeneous Cluster. It is also interesting to note the performance characteristics of the STAP application when running on a fully functional heterogeneous cluster vs other homogeneous clusters that are available at AFIT. These metrics serve as a baseline for comparison to the final implementation of the STAP/Clutter Classification integration.

6.1.1 Initial Benchmark Results. To benchmark the system, a one processor per stage run was completed. It was completed on differing sets of computers resident on the AFIT Heterogeneous Cluster to expose the effects of heterogeneity. One run consisted of the fastest machines in the cluster, (Test 3) , the second consisted of medium speed computers, (Test 2), and the third consisted of the slowest, (Test 1). The program behaved in a very deterministic manner. Identical runs varied in execution times by very little, (less than .005 seconds for throughput and latency), if at all. The minimal changes encountered between runs are related to the non-deterministic way in which communications are handled in the MPI environment. There is no method to ensure that messages

Table 6.1 PII 450MHz STAP Benchmark

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler Filter	1	0.0625	0.8816	1.556	2.5002
Easy Weight	1	2.4049	0.0891	0.0065	2.5004
Hard Weight	1	0.339	1.6789	0.4829	2.5007
Easy BF	1	2.2091	0.291	0.0001	2.5003
Hard BF	1	2.1817	0.2935	0.0245	2.4997
Pulse Comp	1	2.0459	0.4024	0.0514	2.4996
CFAR	1	2.4639	0.0357	0	2.4996
Estimated Throughput		.3999			
Estimated Latency		9.9997			
Measured Throughput		.4205			
Measured Latency		7.1035			

Table 6.2 PIII 600MHz STAP Benchmark

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler Filter	1	.0581	0.861	1.6382	2.5573
Easy Weight	1	2.4639	0.0891	0.0041	2.5571
Hard Weight	1	1.3813	1.1243	0.0001	2.5058
Easy BF	1	2.4071	0.162	0.0001	2.5692
Hard BF	1	2.44	0.1298	0.0001	2.5699
Pulse Comp	1	2.1366	0.405	0.0306	2.5723
CFAR	1	2.5358	0.0365	0	2.5722
Estimated Throughput		.3888			
Estimated Latency		10.2717			
Measured Throughput		.3883			
Measured Latency		4.657			

are sent in the exact same manner, that collisions are consistent, and network traffic is always identical. The slowest machines were 400MHz Pentium III's, the medium speed machines were 600MHz Pentium III's, and the fastest machines were 1.7GHz machines. The speeds of these machines alone affect the the performance of the code, but the same growth and scalability trends are observed. The program ran as expected, and scaled very well with additional processors. Table 6.1, 6.2, and 6.3 contain the results of these initial benchmarks.

6.1.2 Initial Parallel Results. The desired outcome of this project is to process data more effectively and efficiently. In that light, it makes sense to complete preliminary tests examining the current parallelization scheme. It is desirable to understand how

Table 6.3 PIV 1.7GHz STAP Benchmark

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler Filter	1	.0152	0.2712	1.5058	1.7922
Easy Weight	1	1.3574	0.0429	0.3943	1.7945
Hard Weight	1	1.3319	.4603	0.0001	1.7923
Easy BF	1	1.702	0.0612	0.0001	1.792
Hard BF	1	1.7308	0.0893	0.0001	1.7914
Pulse Comp	1	1.6578	0.1278	0.0052	1.7909
CFAR	1	1.7795	0.0113	0	1.7908
Estimated Throughput		.5572			
Estimated Latency		10.2717			
Measured Throughput		.3883			
Measured Latency		4.657			

well the current product scales with additional processors, as well as how it performs and behaves on the local AFIT Heterogeneous Cluster. During the preliminary runs, processors are continually added to the run until one of two things occurs: The processor pool is exhausted, or the addition of a processor results in degraded performance.

One must also consider the architecture of a pipeline in general when performing runs in parallel. Due to the nature of a pipeline, every stage in the pipeline must run as slow as the slowest stage. Therefore, to increase the efficiency of a pipeline, one must isolate the slowest stage, and then optimize it. It is also interesting to note that one may not realize linear speedup, even if linear speedup was achieved in the stage of interest. For example, assume the longest stage in a system takes time x to complete. The addition of a processor to that stage results in a runtime of $\frac{x}{2}$. However, the next longest stage in the pipeline, (before optimization), took $.9x$. After all optimization are complete, every stage in the pipeline still must run at $.9x$. Therefore, results from a pipeline optimization may increase in "spurts." The addition of the first three processors may bring very little improvement. However, with the addition of a fourth additional processor, the speedup provided by all additional processors may be observed [JH98].

The initial experiment, (Test 4), consisted of adding a processor to the slowest pipeline stage, executing the code, observing a new bottleneck, and repeating. For the most part, this experiment yielded exactly what was expected. As processors were added, the computation cost for that stage decrease linearly. There was also very little change in

the communications cost. It is certainly refreshing to see the application execute and scale so well. It is also interesting to note that there appears to be no decrease in scalability as the number of processors increase. Clearly, a point or a "knee" in performance may be observed if enough processors are added. However, in the limited environment of the AFIT cluster, it appears that there is almost equal benefit added for each additional processor. Runs were completed with processor count ranging from seven to sixteen. If there were more processors available, it would be interesting to add them and observe scalability at higher processor counts.

The following graphs explicitly show the relationship between additional processors and throughput/latency. In general, one would expect to see latency lowered by shortening the duration of the longest pipeline stage with the addition of processors. Indeed, that is exactly what was observed. The overall latency is the length of the pipeline multiplied by the longest stage. The throughput is also be enhanced by the addition of processors. By adding processors to the longest stage, throughput is increased, because the pipeline produces one result per pipeline cycle. If the cycle is shortened, there are more cycles during any given time, and thus the throughput increases. STAP is nothing more than linear algebra and statistics. It has been shown many times that most linear algebra problems scale nicely with additional processors. Clearly, the operations involved in STAP are no different. The STAP application performed extremely well, scaled well, and produced accurate stable results consistently. This is shown in Figure 6.1 and 6.2.

Upon examination, one may be tempted to assume that the algorithm is not scaling as well with the last few processors. Upon closer examination, however, it appears that this is not the case. Previously, the concept of speedup in a pipeline was addressed. The speedup is very closely related to the cost of the longest pipeline stage. When the throughput is graphed alongside the cost of the longest stage in the pipeline, (Figures 6.1, 6.2, 6.5, and 6.6), one notices that the throughput and the cost of the longest stage is almost a perfect inverse relation.

This was also clearly the case on the slower machines. The first additional processor added resulted in a very large throughput increase. The second processor did not. The algorithm is clearly not experiencing scaling problems, but rather can only advance to

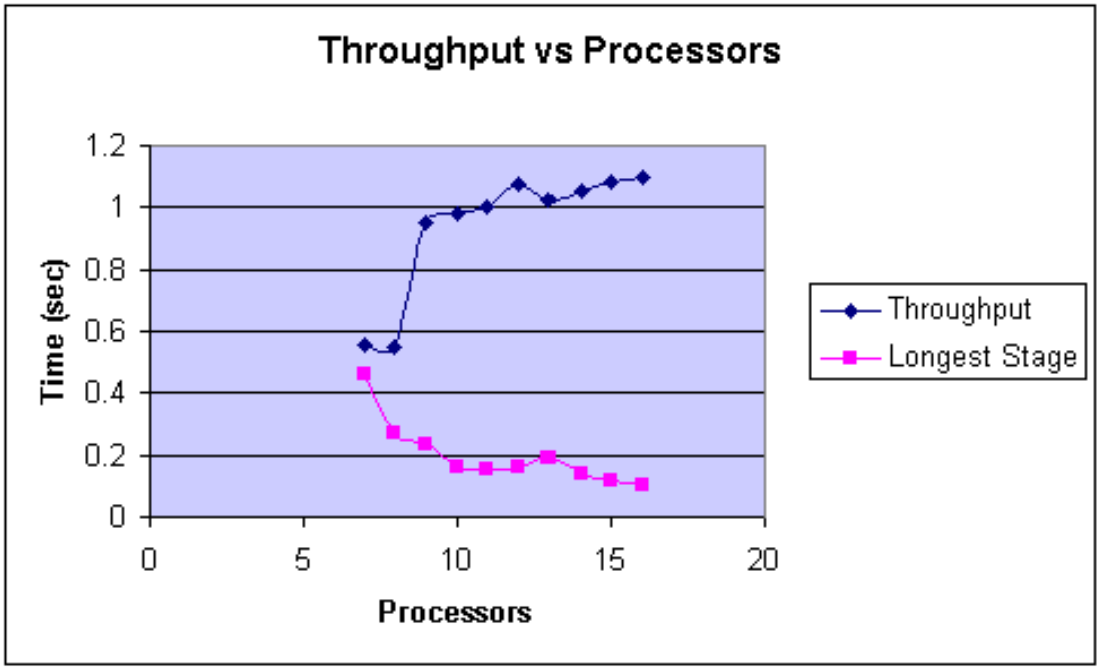

Figure 6.1 Throughput (Faster Machines)

the rate of the next slowest pipeline stage. Just as with throughput, the latency results of the STAP experiment are exactly what was expected. More processors were added, hence decreasing the cycle time of the pipeline, latency was reduced. Again, one notes that as a whole, the algorithm seems to scale linearly, however, that scaling is closely tied to reduction of the longest pipeline stage. Latency is directly related to the length of the longest stage, as well as other factors such as changing communication overhead with additional processors. This is shown for both the fast and slow processor runs in Figure 6.3 and 6.4.

In this section, one must also not overlook certain factors unique to our cluster. The AFIT cluster is very heterogeneous. This makes processor allocation extremely relevant in this application, (please refer to Appendix D for genetic algorithm optimization of processor allocation). For example, suppose that there are five 1.7GHz machines currently processing the hard weight computation, but it is still the slowest stage in the pipeline. Therefore, another processors is added, but it is only a 450MHz machine. Because of the fact that all machines distribute work inside of a stage equally, as well as the fact that

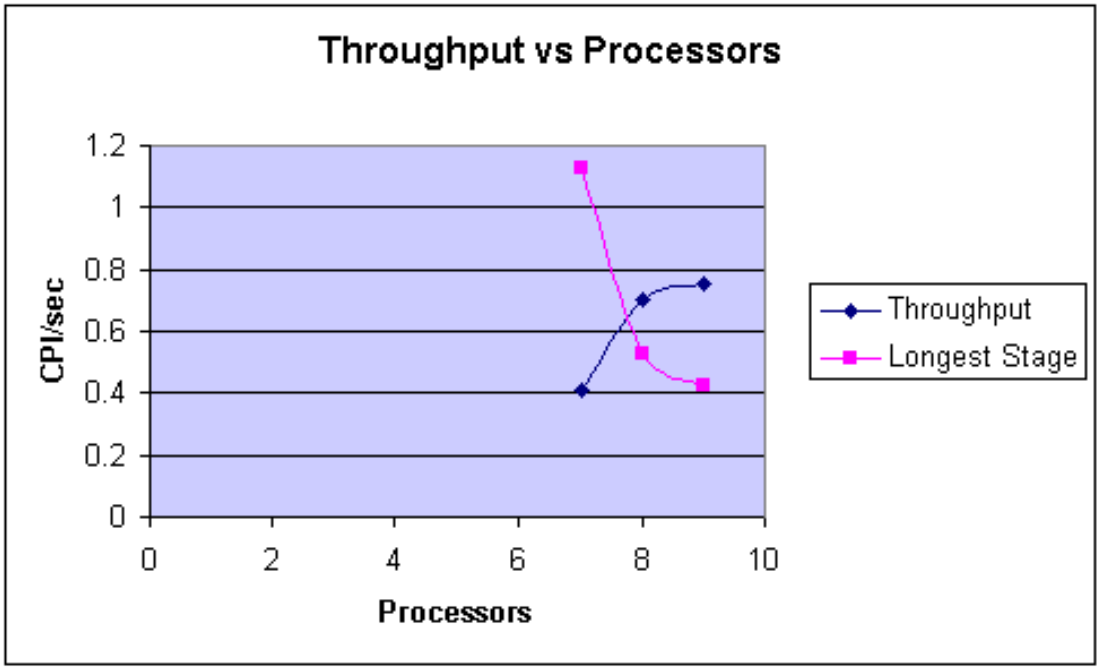

Figure 6.2 Throughput (Slower Machines)

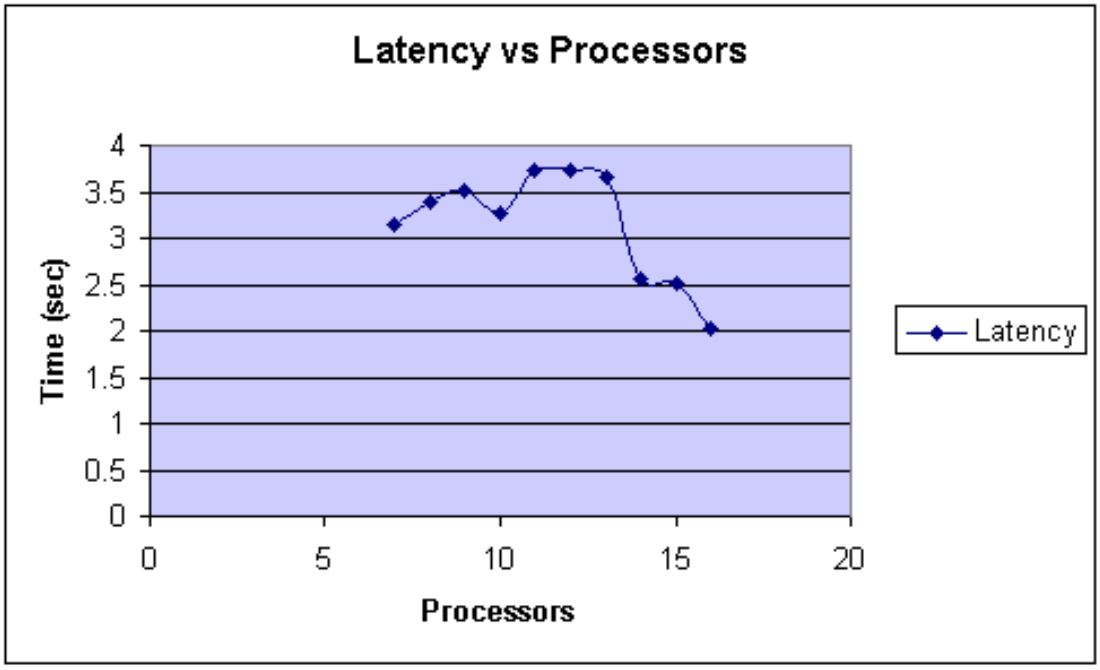

Figure 6.3 Latency (Faster Machines)

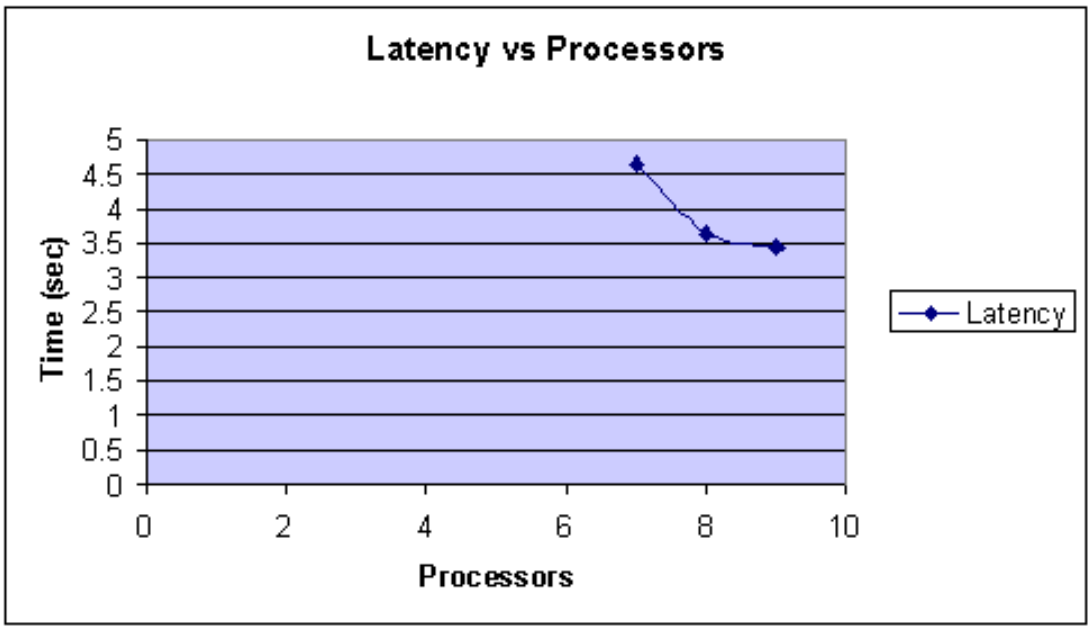

Figure 6.4 Latency (Slower Machines)

they must all complete together and pass the information to the next stage, all of the processors in that stage may now only run at the speed of the 450MHz machine. Clearly in this scenario it is possible to add a machine to a stage and reduce the performance of that stage completely disjoint of any scaling problems.

6.2 STAP Results on Polywell Cluster

After these initial benchmarks, it may be useful to understand STAP performance and capabilities on a newer homogeneous cluster. This cluster consists of 16 identical 1.2 GHz Athalon processors. They are connected with a 100Mbps Ethernet backbone. Again, the same series of tests were conducted. Processors are continually be added to the slowest pipeline stage until the processor pool is exhausted, or there is a decrease in performance, (Test 5). These runs are significantly less complex, because of the homogeneity of the cluster. One need not account for different speeds of processors, and how they may effect the pipeline. Because of this fact, the results are much more predictable and also show great scalability. The actual decomposition of these parallel runs are contained in tables 6.4 - 6.13.

Table 6.4 Incremental Run (Seven Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	1	0.0288	0.3199	1.4940	1.8426
easy weight	1	1.8134	0.0263	0.0030	1.8427
hard weight	1	1.5179	0.3248	0.0001	1.8427
easy BF	1	1.7859	0.0565	0.0001	1.8424
hard BF	1	1.7621	0.0801	0.0001	1.8423
pulse compr	1	1.7203	0.1095	0.0124	1.8422
CFAR	1	1.8293	0.0128	0.0000	1.8421
Estimated Throughput		0.5427			
Estimated Latency		7.3693			
Measured Throughput		0.5426			
Measured Latency		3.2550			

Table 6.5 Incremental Run (Eight Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	1	0.0289	0.3192	1.4970	1.8451
easy weight	1	1.8158	0.0263	0.0030	1.8451
hard weight	2	1.6825	0.1624	0.0001	1.8450
easy BF	1	1.7884	0.0564	0.0001	1.8449
hard BF	1	1.7645	0.0801	0.0001	1.8447
pulse compr	1	1.7226	0.1094	0.0125	1.8445
CFAR	1	1.8316	0.0128	0.0000	1.8444
Estimated Throughput		0.5420			
Estimated Latency		7.3789			
Measured Throughput		0.5418			
Measured Latency		2.9666			

Table 6.6 Incremental Run (Nine Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	2	0.0145	0.1614	0.8135	0.9894
easy weight	1	0.9486	0.0263	0.0031	0.9780
hard weight	2	0.8113	0.1628	0.0019	0.9760
easy BF	1	0.9368	0.0567	0.0001	0.9936
hard BF	1	0.8165	0.0800	0.1062	1.0027
pulse compr	1	0.8569	0.1092	0.0230	0.9890
CFAR	1	0.9603	0.0128	0.0000	0.9731
Estimated Throughput		0.9973			
Estimated Latency		3.9542			
Measured Throughput		1.0338			
Measured Latency		2.3267			

Table 6.7 Incremental Run (Ten Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	2	0.0148	0.1615	0.8304	1.0067
easy weight	1	0.9748	0.0263	0.0031	1.0041
hard weight	3	0.9036	0.1085	0.0007	1.0128
easy BF	1	0.9160	0.0565	0.0001	0.9726
hard BF	1	0.8328	0.0803	0.0928	1.0059
pulse compr	1	0.9285	0.1089	0.0230	1.0603
CFAR	1	1.0708	0.0128	0.0000	1.0837
Estimated Throughput		0.9228			
Estimated Latency		4.1565			
Measured Throughput		0.9119			
Measured Latency		2.0828			

Table 6.8 Incremental Run (Eleven Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	3	0.0118	0.1075	0.7187	0.8381
easy weight	1	0.8559	0.0263	0.0821	0.9643
hard weight	3	0.5384	0.1086	0.5526	1.1996
easy BF	1	0.7931	0.0567	0.1311	0.9809
hard BF	1	0.3955	0.0806	0.5548	1.0308
pulse compr	1	0.8600	0.1091	0.0410	1.0101
CFAR	1	1.0102	0.0128	0.0000	1.0230
Estimated Throughput		0.8336			
Estimated Latency		3.90200			
Measured Throughput		0.9591			
Measured Latency		3.10740			

Table 6.9 Incremental Run (Twelve Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	3	0.0114	0.1074	0.7132	0.8321
easy weight	1	0.8342	0.0263	0.0475	0.9080
hard weight	3	0.4632	0.1084	0.4332	1.0048
easy BF	1	0.8042	0.0568	0.0536	0.9147
hard BF	1	0.5352	0.0802	0.3293	0.9447
pulse compr	2	0.8650	0.0549	0.0910	1.0108
CFAR	1	0.9133	0.0128	0.0000	0.9261
Estimated Throughput		0.9893			
Estimated Latency		3.7138			
Measured Throughput		1.0769			
Measured Latency		2.32570			

Table 6.10 Incremental Run (Thirteen Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	3	0.0118	0.1075	0.7601	0.8794
easy weight	1	0.8346	0.0263	0.0606	0.9215
hard weight	4	0.7076	0.0815	0.2993	1.0884
easy BF	1	0.7383	0.0568	0.1168	0.9119
hard BF	1	0.5776	0.0803	0.2672	0.9250
pulse compr	2	0.8282	0.0550	0.0704	0.9536
CFAR	1	0.9104	0.0128	0.0000	0.9232
Estimated Throughput		0.9188			
Estimated Latency		3.6812			
Measured Throughput		1.0621			
Measured Latency		1.7075			

Table 6.11 Incremental Run (Fourteen Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	4	0.0085	0.0807	0.8387	0.9279
easy weight	1	0.8166	0.0263	0.0487	0.8916
hard weight	4	0.4756	0.0817	0.4128	0.9701
easy BF	1	0.7667	0.0566	0.1352	0.9585
hard BF	1	0.4561	0.0804	0.4257	0.9622
pulse compr	2	0.8929	0.0546	0.0463	0.9937
CFAR	1	0.9501	0.0257	0.0000	0.9758
Estimated Throughput		1.0063			
Estimated Latency		3.8596			
Measured Throughput		1.0808			
Measured Latency		1.8822			

Table 6.12 Incremental Run (Fifteen Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	4	0.0083	0.0807	0.8298	0.9189
easy weight	1	0.8227	0.0263	0.0651	0.9141
hard weight	5	0.7043	0.0661	0.1806	0.9510
easy BF	1	0.8456	0.0564	0.0202	0.9223
hard BF	1	0.6123	0.0812	0.2778	0.9712
pulse compr	2	0.8379	0.0547	0.0192	0.9118
CFAR	1	0.8848	0.0257	0.0000	0.9104
Estimated Throughput		1.0296			
Estimated Latency		3.7123			
Measured Throughput		1.0333			
Measured Latency		1.2886			

Table 6.13 Incremental Run (Sixteen Processors)

Pipeline Stage	Processors	Receive	Computation	Send	Total
Doppler filter	4	0.0152	0.1017	0.5899	0.7069
easy weight	1	0.6445	0.0263	0.1047	0.7755
hard weight	5	0.5692	0.0659	0.1789	0.8141
easy BF	1	0.5844	0.0567	0.1283	0.7694
hard BF	2	0.6577	0.0402	0.0367	0.7346
pulse compr	2	0.5672	0.1398	0.0829	0.7898
CFAR	1	0.7502	0.0196	0.0000	0.7698
Estimated Throughput		1.22841			
Estimated Latency		3.0359			
Measured Throughput		1.3129			
Measured Latency		1.200			

The results of this analysis are represented in figure 6.5. Note that there is an extremely clear inverse correlation between the throughput and the cost of the longest stage. As the cost of the longest stage decreases, (or increases a little in some cases), the throughput varies inversely with the change in cost. The latency in figure 6.6 also depicts a clear direct correlation between the cost of the longest stage and the overall latency of the pipeline. This is for the same reason that the throughput is inversely correlated with the cost of the longest stage. Every stage must operate at the cost of the longest stage. Therefore, the latency is the length of the pipeline times the longest stage in the pipeline. Keeping this is mind makes the runtime behavior of the STAP process much clearer.

6.3 Preliminary Serial Clutter Classification Analysis

The new Clutter Classification code has been tested in isolation to evaluate the performance gains and abilities offered by the new system. However, performance metrics of the old FORTRAN code are somewhat unclear before optimizations were made. The old code does not run on any platform that is currently available. Therefore the results of past runs are shown. Even though the measurements were taken on parallel machines, they offer a general feel for the orders of magnitude of improvement that were gained by the change in implementation. The Figure 6.7 contains the best results that were ever obtained from the original Ozturk code [US98].

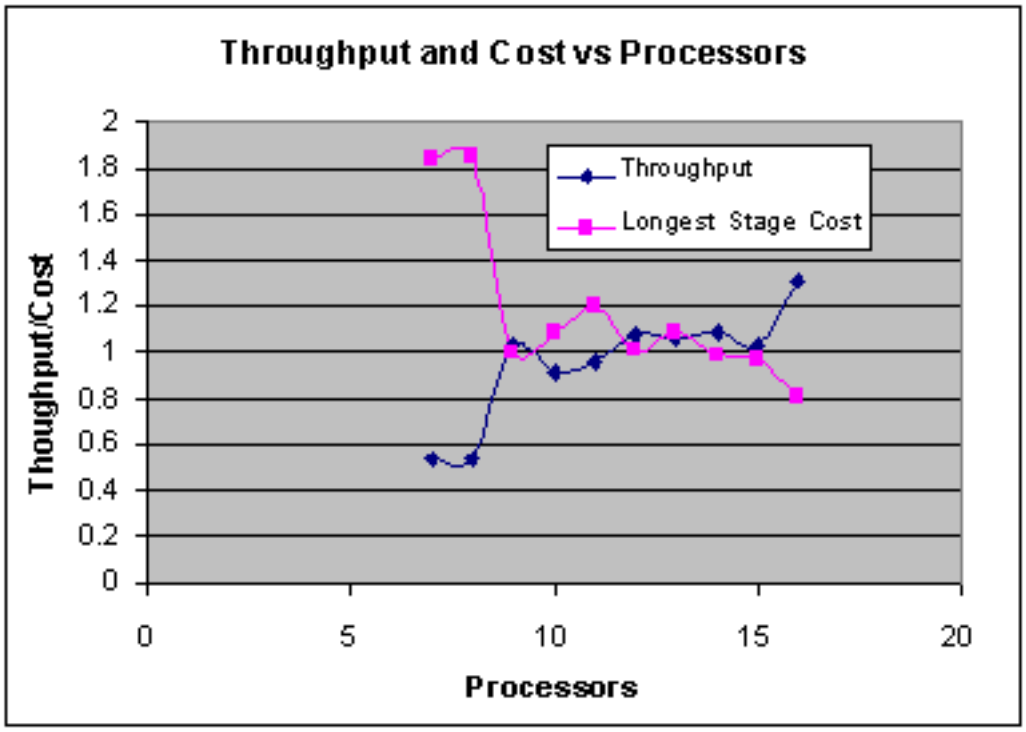

Figure 6.5 Polywell Cluster Scalability Throughput Performance

Once notices that even the absolute best returns on Argonne Lab's SP is around three seconds per input size of 500. Similar tests were conducted on a one-processor Pentium IV machine running at 1700MHz, (Test 6). Even though there is clearly no comparison between the computing power of the SP and the Intel PC, one may note the time taken to process distributions with varying sizes are about three orders of magnitude less than on the SP. Also, to illustrate the scalability of this algorithm, sample sizes of up to 2000 were used. These sample sizes are four times larger than the sample run with the old code, and yet they still complete orders of magnitude quicker. It appears that this implementation scales linearly with sample size, which is also reflects well upon this implementation. Clearly this is a vast improvement to the code, and should prove very useful in the STAP and Clutter Classification integration [CC01].

6.3.1 Parallel Results of Clutter Classification on AFIT Heterogeneous Cluster. With the entire parallelization of the Clutter Classification complete, an analysis of the performance was undertaken. This is necessary to understand the performance charac-

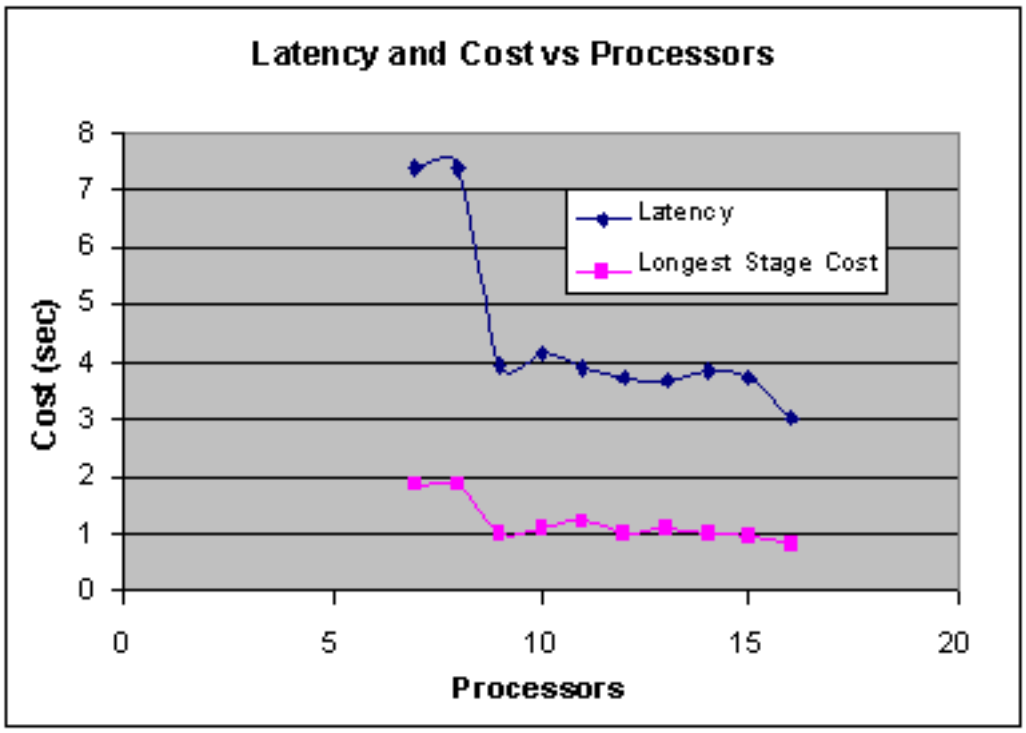

Figure 6.6 Polywell Cluster Scalability Latency Performance

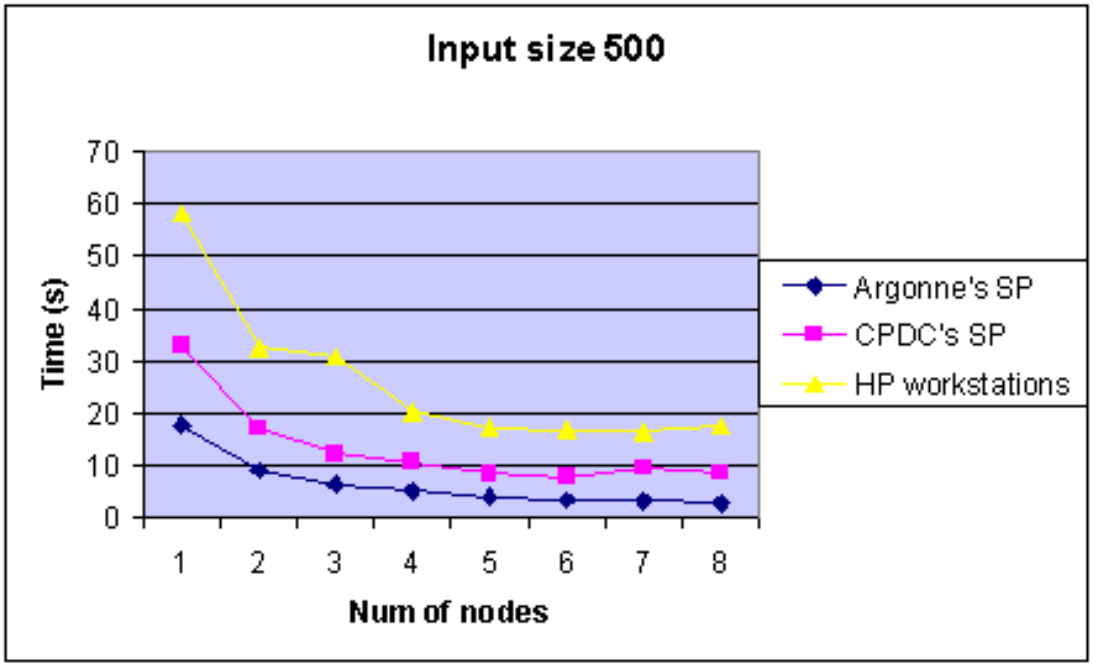

Figure 6.7 Parallel FORTRAN Ozturk Performance [US98]

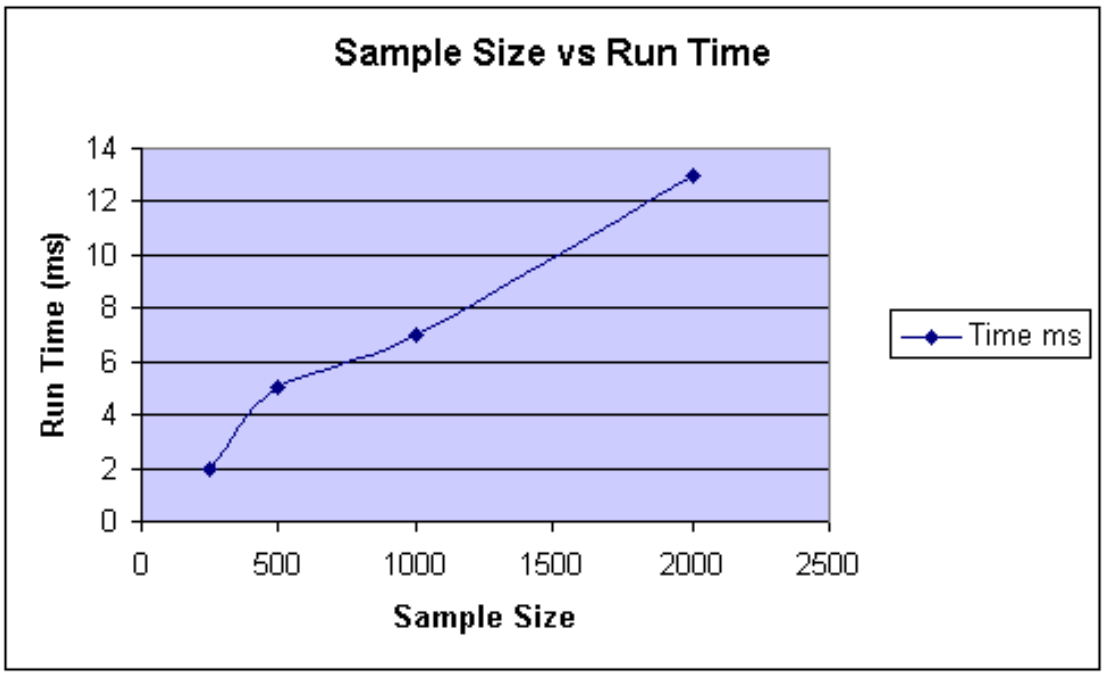

Figure 6.8 Optimized C Ozturk Performance

teristics before augmentation into the STAP parallel pipeline. It provides a feel for the performance metrics that are realized once this product is converted into a pipeline stage. It also validates the hypothesis that this product is very scalable. To get a better understanding of how each phase of the product behaves when implemented in parallel, each section has been been analyzed separately, followed by a complete study of performance for the entire package. These tests were conducted on both the AFIT Heterogeneous Cluster and the AFIT Polywell Cluster for further analysis of the different hardware platform capabilities.

The first part of the Clutter Classification product studied was the parallel read, (Test 7). In the first iterations of this application, each processor read the entire data cube. However, this proved to be a much more time intensive process than initially assumed. Therefore, it became necessary to add additional functionality that would allow each process to only load the regions of the data cube that would be needed for local processing. Since these regions are independent, each processor may load only its section of the data. The actions of one process never supply causation for change in another pro-

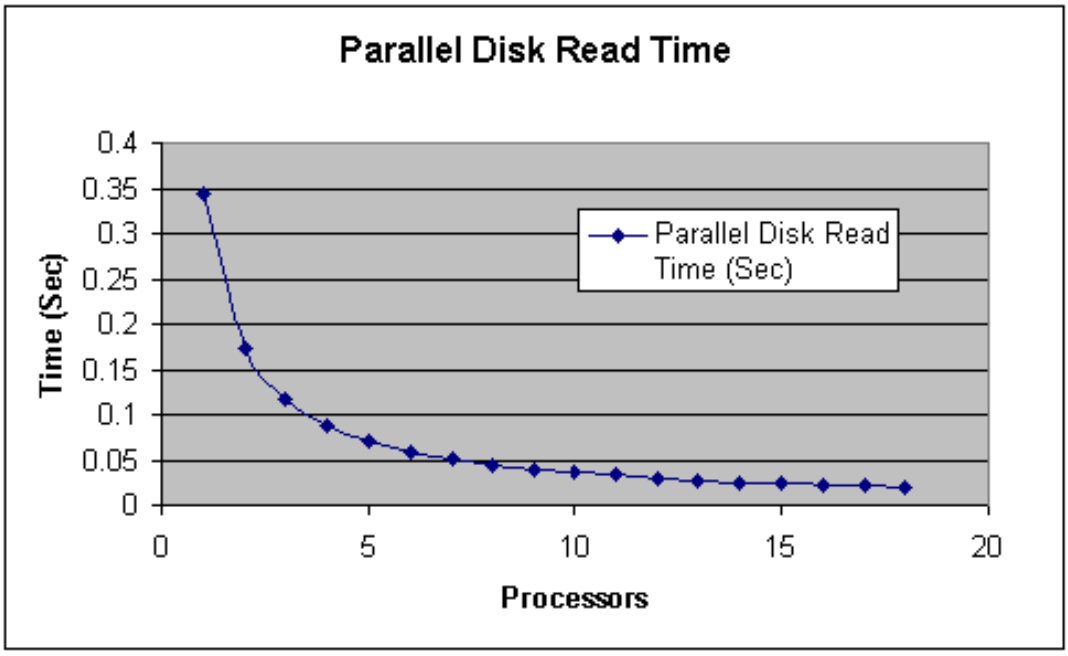

Figure 6.9 Runtime of Parallel Read CPI Data

cess. As expected, these loads scale very well with additional processors. The file is read only; therefore, no blocking or waiting is required of any processor [JH98]. There is some overhead that is observed due to the need for every process to obtain a file handle and open the file [VK94]. As shown in figure 6.9 the time needed for the parallel load decreases with every processor added.

Not only is it clear that definite benefits are gained by this parallel read, but these speedups are very linear in nature. Figure 6.10 shows the parallel disk read speedup plotted against true linear speedup. Note that there is minimal loss due to overhead. However, these losses are miniscule when compared to the benefit that is added with additional processors. When running with 12-18 processors, this additional feature resulted in the time savings of at least $\frac{3}{10}$ of a second. When considering that processing the entire data cube in parallel took around $\frac{5}{10}$ of a second this is drastic improvement.

The next stage of the Clutter Classification that was studied was the actual parallel calculation of best fit distributions, (Test 8). The is essentially the most computationally complex portion of the process. Even though this is the most computationally complex,

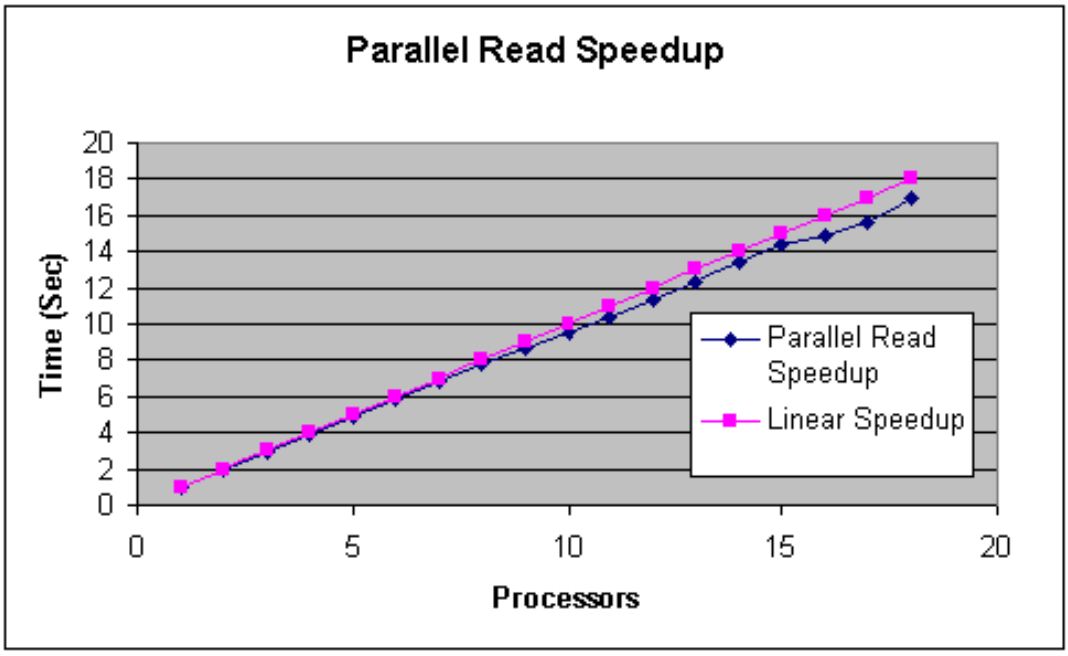

Figure 6.10 Speedup of Parallel CPI Read

it is only on the order of n^3 [TC00]. However, due to the independent nature of the range cell processing, significant speedups are still achieved. For the most part, this entire process consists of several calls to the PAREST function by different processors on different portions of the data cube. Identical parameters are used to process each range cell. This is ensured by forcing one processor to calculate all of these values a priori, and then distributing these values to their remote locations. As observed in figure 6.11, additional processors also result in a decrease in execution time. This is a very valuable trait. This is the portion of the code that takes the most time to execute, and clearly benefits from a parallel implementation. This also serves as a good indication that it integrates nicely into the parallel pipeline STAP process that has already been developed.

Just as the parallel read, the parallel section of the distribution identification also scales very well. Figure 6.12 shows the speedup of the parallel program vs a true linear speedup. Clearly this program scales nicely, even when the number of processors used gets quite high. However, one remember that this is only the parallel portion of the code. There is some overhead that must be accounted for. This parallel exploitation decreases the wall clock time to process 630 range cells from $4\frac{1}{2}$ seconds on average to less than

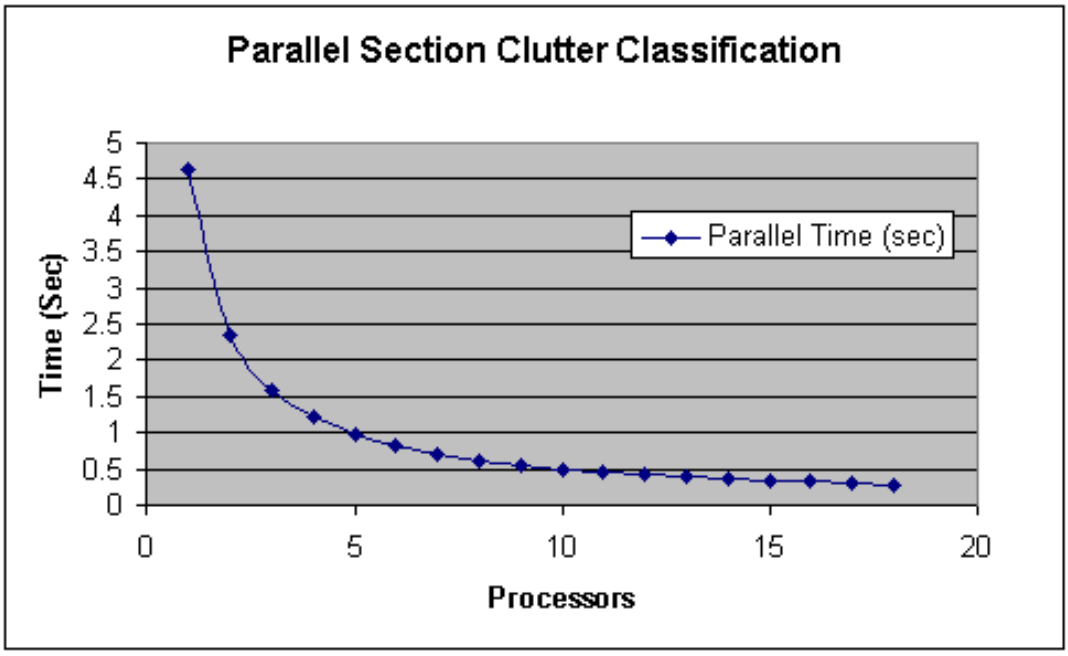

Figure 6.11 Runtime of Parallel Section of Clutter Classification

$\frac{4}{10}$ of a second, again, a drastic improvement. The process is also very predictable. The variation of completion times between runs was so miniscule that it does not appear on the graphical representations.

Even though this process is very parallel in nature, there is still some serial overhead involved that cannot be avoided. There is also some startup overhead that is involved just by the very nature of using MPI programming constructs [JG95]. All of these things must be accounted for. Luckily in this case they are quite small, almost to the point that they may be disregarded in the big picture. This serial section is where all of the parallel results are gathered back to the root processor. Once there, the root processor makes comparisons from the results of all the range cells. It discovers which range cells are not like the surrounding range cells by determining how many of top five best fit distributions each has in common with adjacent range cells. For the smaller processor counts this overhead remained extremely small, especially when only P4 1.7GHz machines were used. Once slower machines were added, this created a very heterogeneous cluster, and the measure of latency was unpredictable yet still quite small, (Test 9). It may also be true that with the addition of so many processors, the major time factor involved was

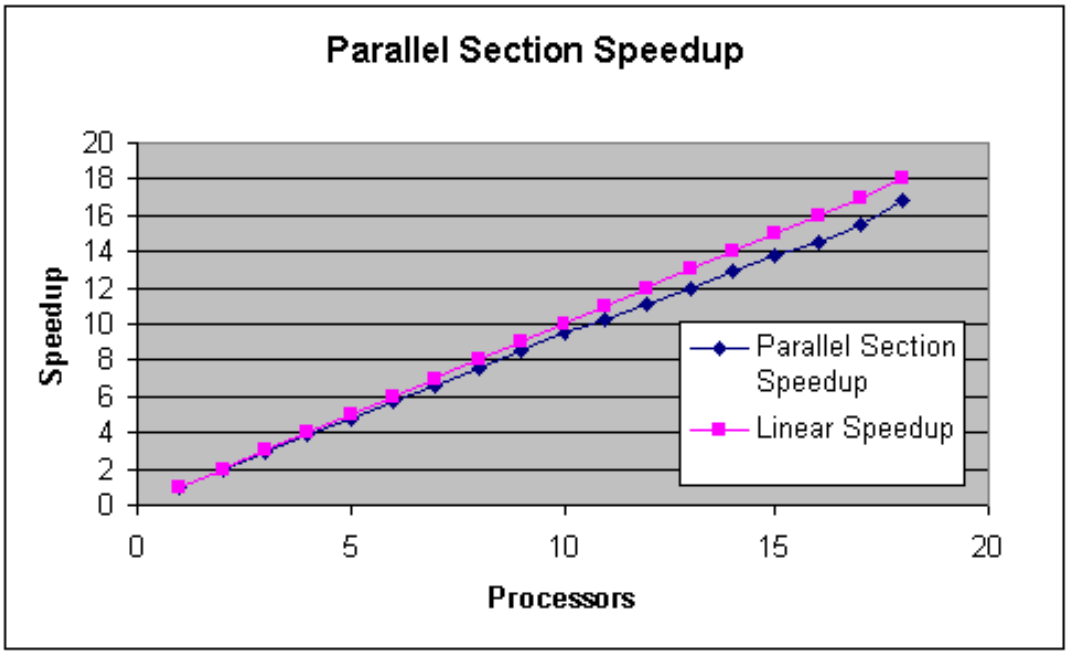

Figure 6.12 Speedup of Parallel Section of Clutter Classification

synchronizing all of the processors, rather than actually passing the data. Also, additional machines may results in greater probability of error and retransmit requirements [PP96]. All of these additional overhead values are accounted for in the serial section of the code. As observed in figure 6.13, the overhead remains extremely low. Once the slower processors were added as high numbered processors, this value becomes unpredictable.

Now that each portion of the Clutter Classification has been discussed, the overall results of the product analysis can be more meaningful. As a whole, the parallel implementation provides drastic performance benefits, and scales well with additional processors, (Test 10). This certainly gives one reason to believe that if implemented on a system with a more consistent communication structure that it would continue to scale well and perform even better. However, due to the heterogeneity of the AFIT cluster, it is apparent that the serial portion of the code behaves radically when there is a significant number of processors added with different performance capabilities. Even though this time is very miniscule even in the worst case, the parallel time to process a range cell is so small that even small amounts of overhead are noticed. In Figure 6.14, the total runtime of the parallel implementation is observed. As expected, additional processors reduce runtime in

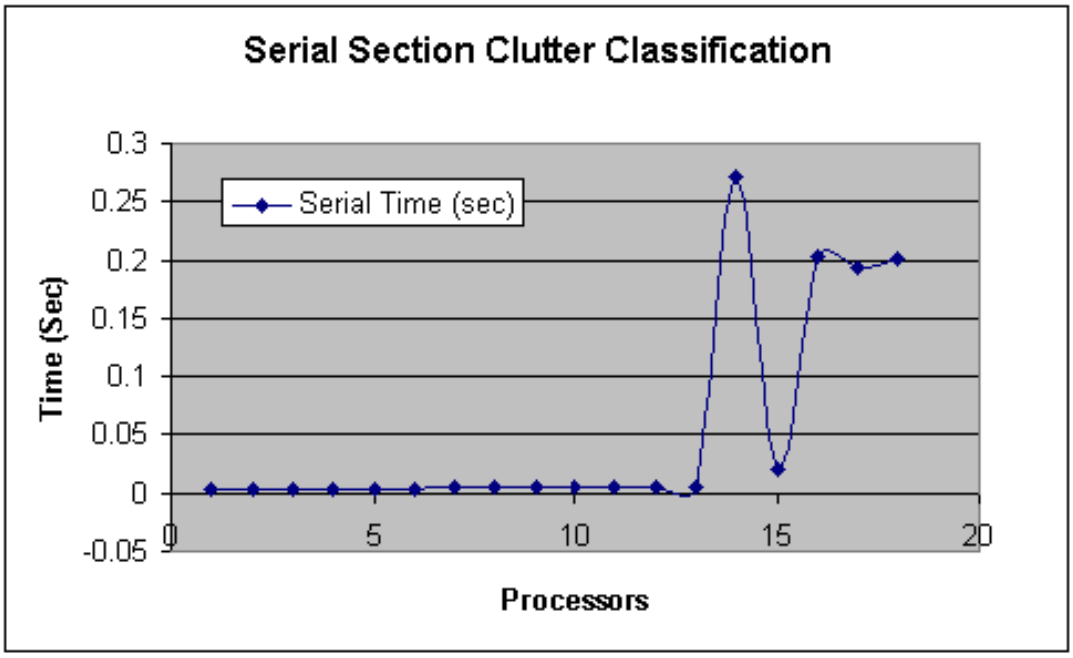

Figure 6.13 Runtime of Serial Section of Clutter Classification

general. However, when the processor count is high, the additional unpredictable overhead skews run times minimally.

The entire process enjoys near linear speedup, except when the processor count is high. Again, this leads one to believe that it is an excellent candidate for a front end filter for the parallel pipeline STAP product. Not only does is appear to adapt to the parallel pipeline well, but it also has high potential for capability in isolation. Even considering just the Clutter Classification product in isolation, these significant enhancements promise to be very valuable to the academic community and the DoD alike.

6.3.2 Parallel Results of Clutter Classification on Polywell Cluster. The parallel aspects of the clutter classification algorithm on the newer homogeneous AFIT Polywell cluster are also relevant to the discussion at hand. The same set of tests that were conducted on the AFIT Heterogeneous Cluster were also conducted on the Polywell cluster. This should expose many of the differences that are encountered when dealing with a homogeneous cluster vs. a heterogeneous cluster. It is shown that speedup, throughput, and latency are much more predictable when dealing with a homogeneous cluster. It is

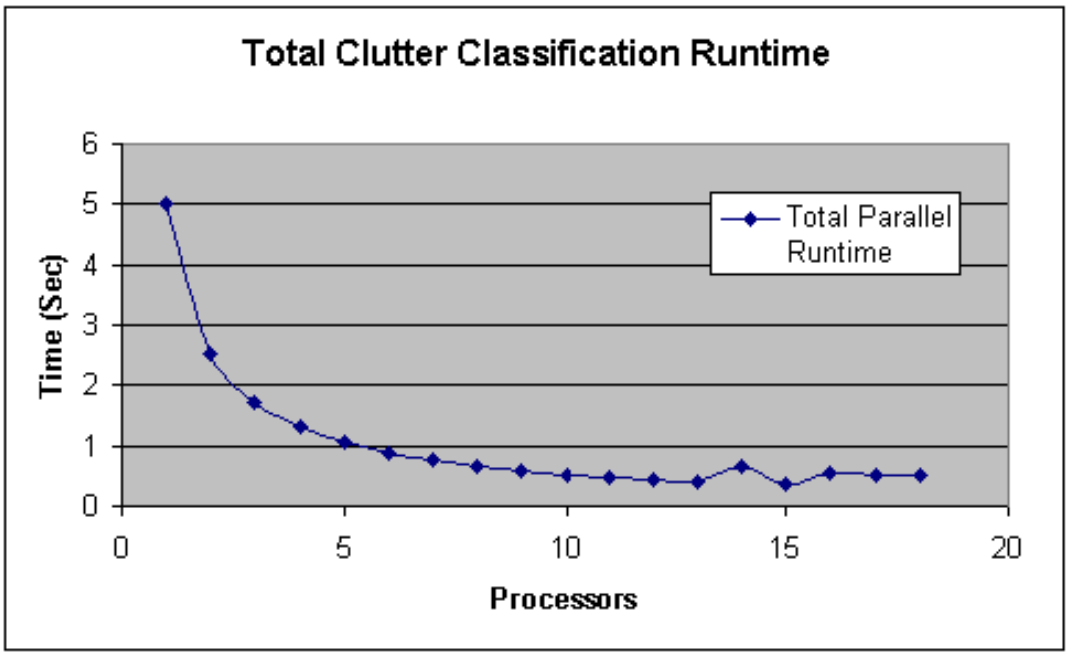

Figure 6.14 Runtime of Total Parallel Clutter Classification

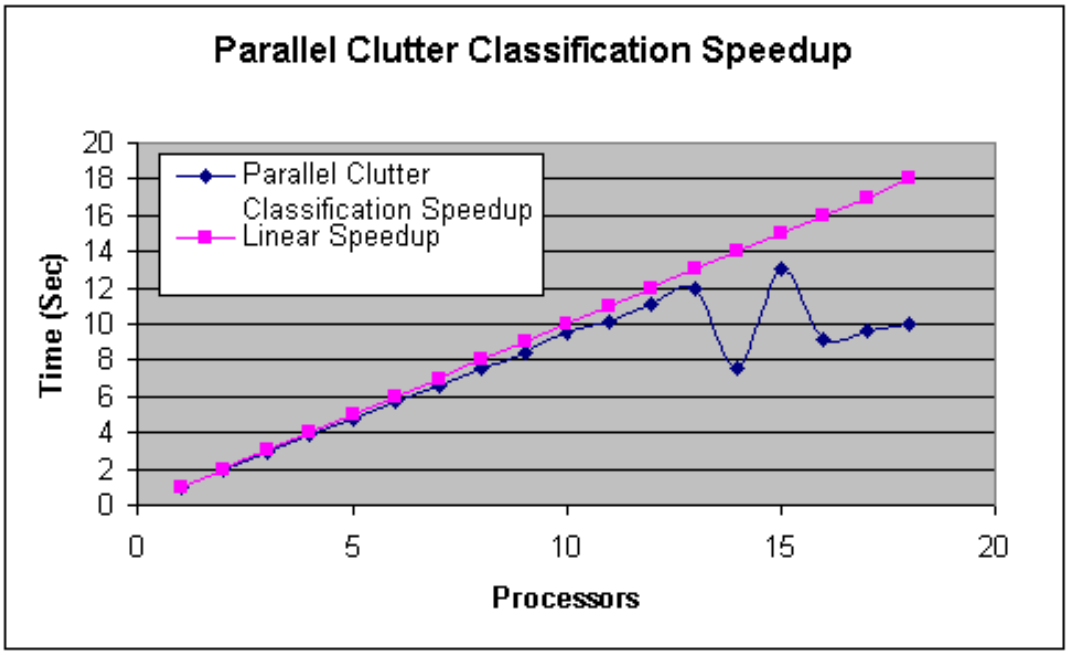

Figure 6.15 Speedup of Total Parallel Clutter Classification

also much simpler to allocate processors in an environment where each node has the same capability as all the other nodes on the cluster.

Just as was the case with the AFIT Heterogeneous Cluster, the parallel read cost and speedup with additional processors explicitly highlight the great scalability of the read in general, (Test 11). This is shown for latency and speedup in Figure 6.16 and Figure 6.17 respectively. The second set of figures deal with the parallel computation costs associated with the identification of the actual data sets, (Test 12). Figure 6.18 and 6.19 show the cost and speedup respectively for this section of the code. The third measurements taken were for the serial section of the code, (Test 13). This is the portion of the code that cannot be done in parallel. It may simply be understood as overhead that must be dealt with regardless of the processors involved in the computation. The serial section results are contained in Figure 6.20 and 6.21. One should also note the extreme spike in cost at the end of the serial time. It is hypothesized that this is encountered due to overhead encountered in synchronizing the many different systems involved. The final measurements taken were the cost and speedup of the entire system, (Test 14). It is clear that this application scales well with additional processors, and may thus be a viable front end for the STAP application. This data is graphically shown in figure 6.22 and figure 6.23.

This entire test set was conducted ten times each. However, it was discovered that there was very little variability between runs, as long as processor allocation remained constant. The results remained so constant that error bars on the above charts are not even visible. However, runs with the same processors in differing orders produced vastly different results on the heterogenous system. This was expected and dealt with by using the results that appeared to offer the greatest efficiency.

6.4 Clutter Classification as a Non-Homogeneity Detector Results

As discussed in Chapter III, it may be possible to use Clutter Classification as a Non-Homogeneity detector. Therefore, test sets were created to test this hypothesis, (Test 15). Known MCARM data cubes were processed to determine if known characteristics found in these data sets were identified by the Clutter Classification application. This section

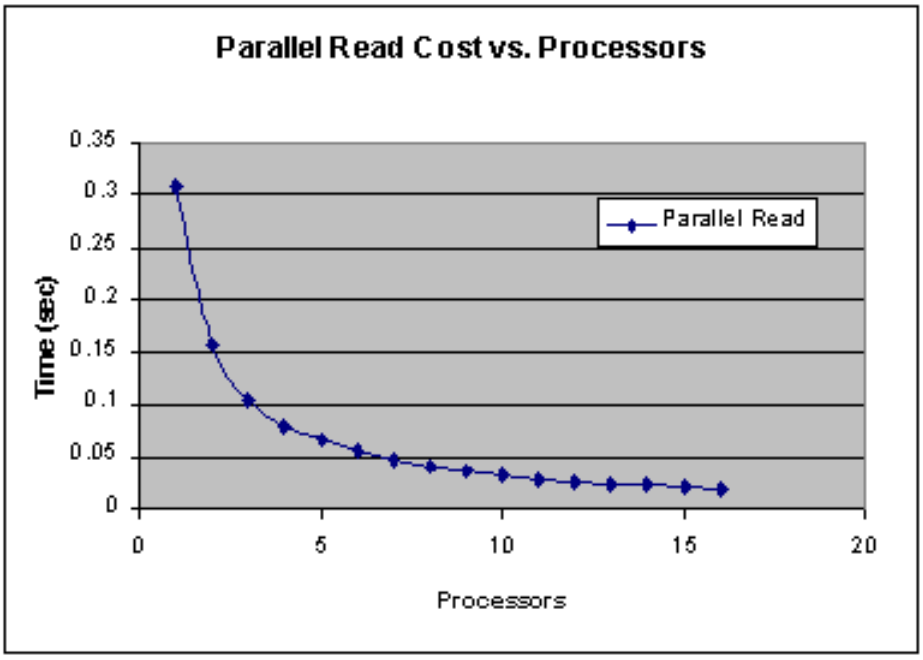

Figure 6.16 Parallel Read Cost

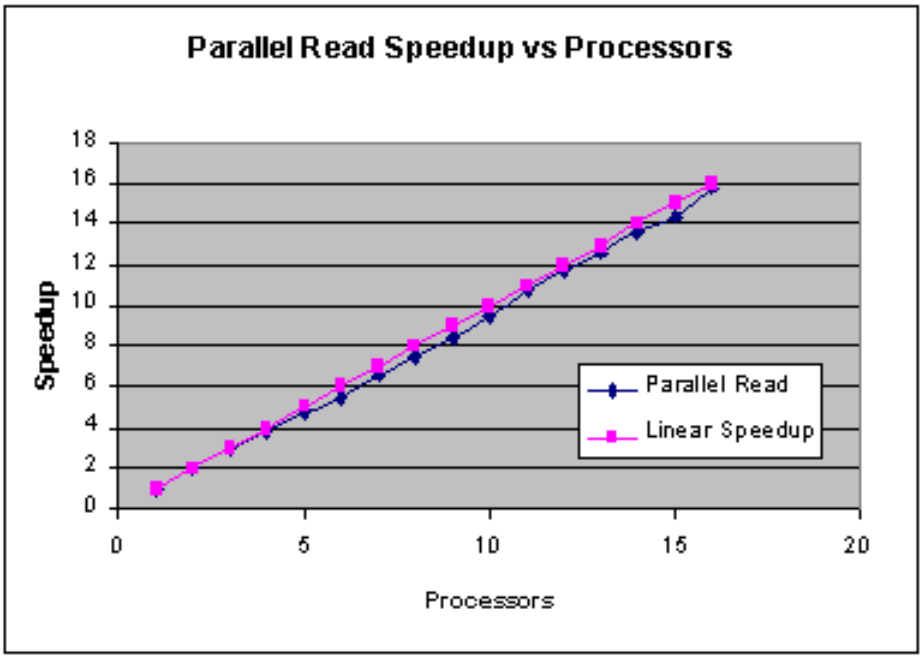

Figure 6.17 Parallel Read Speedup

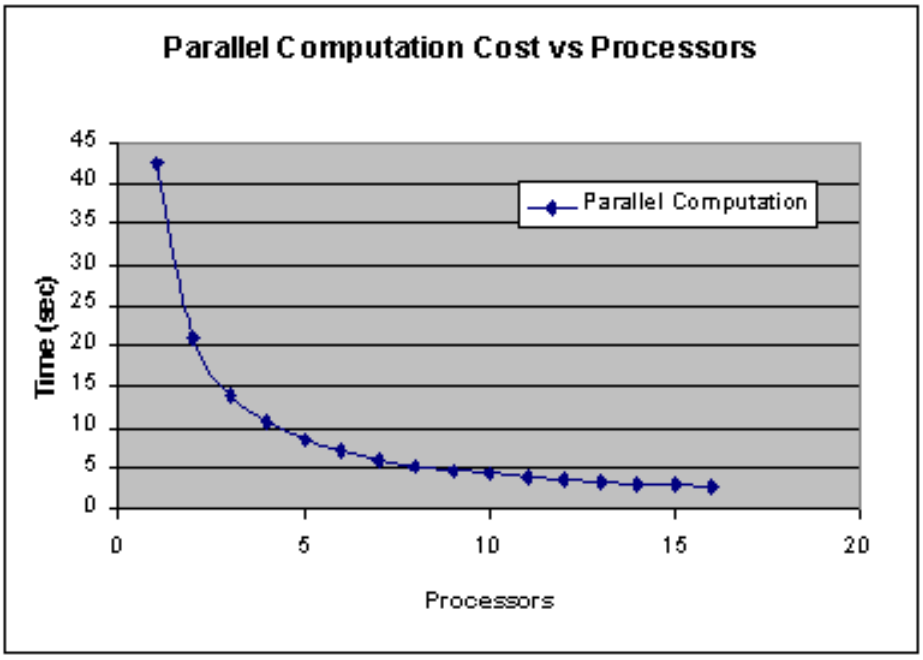

Figure 6.18 Parallel Computation Cost

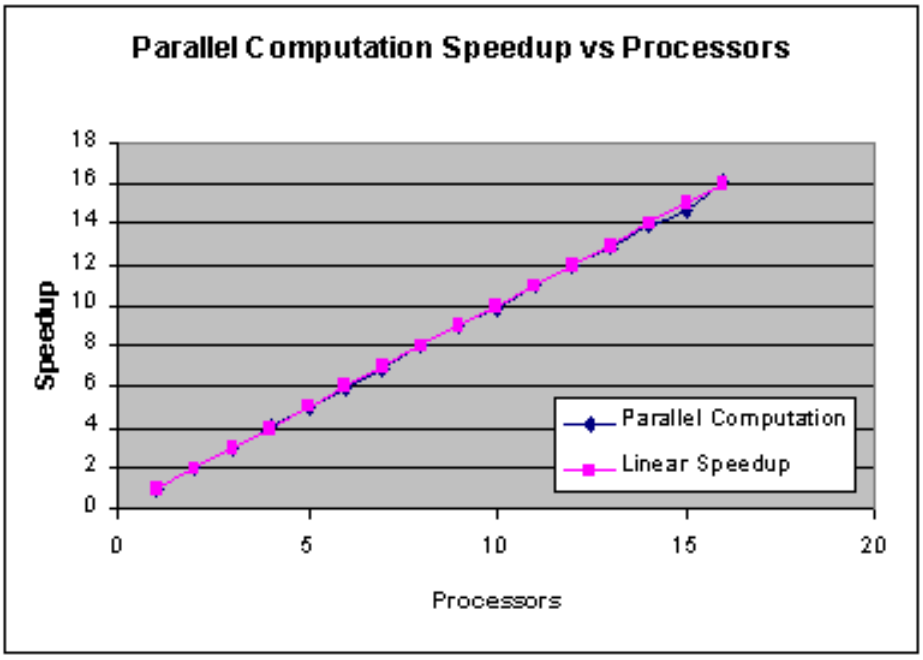

Figure 6.19 Parallel Computation Speedup

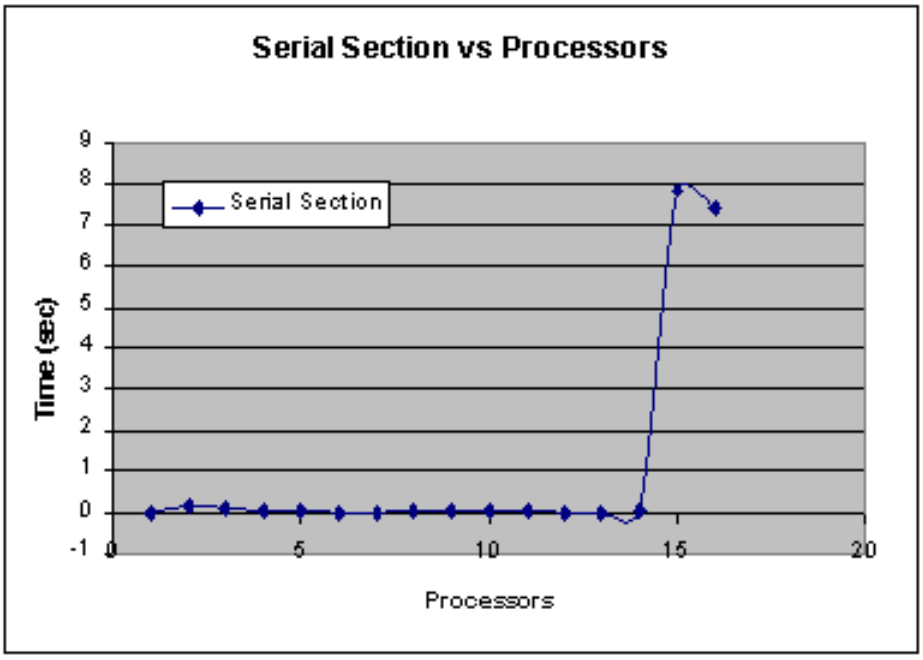

Figure 6.20 Serial Section Cost

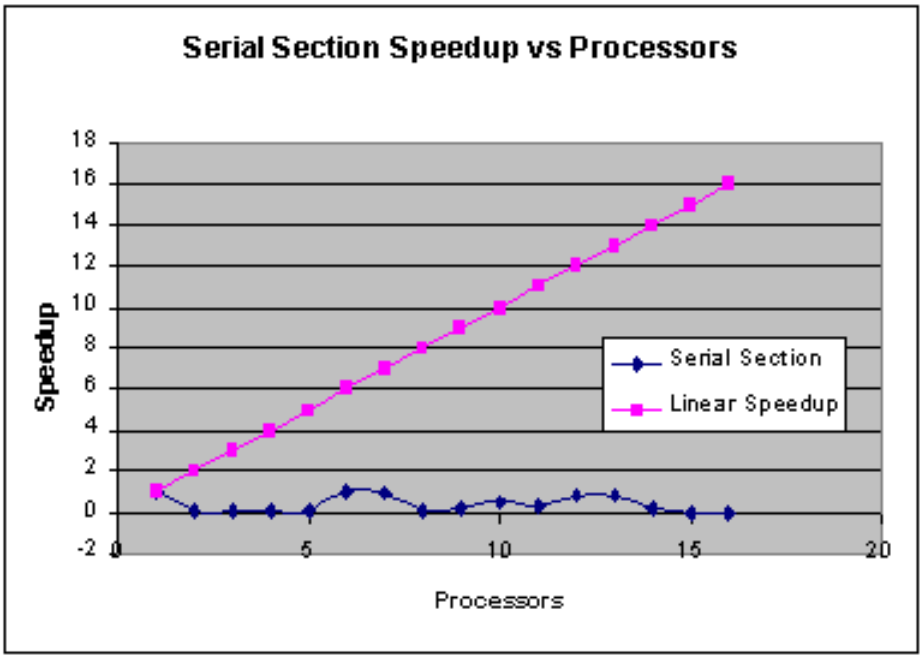

Figure 6.21 Serial Section Speedup

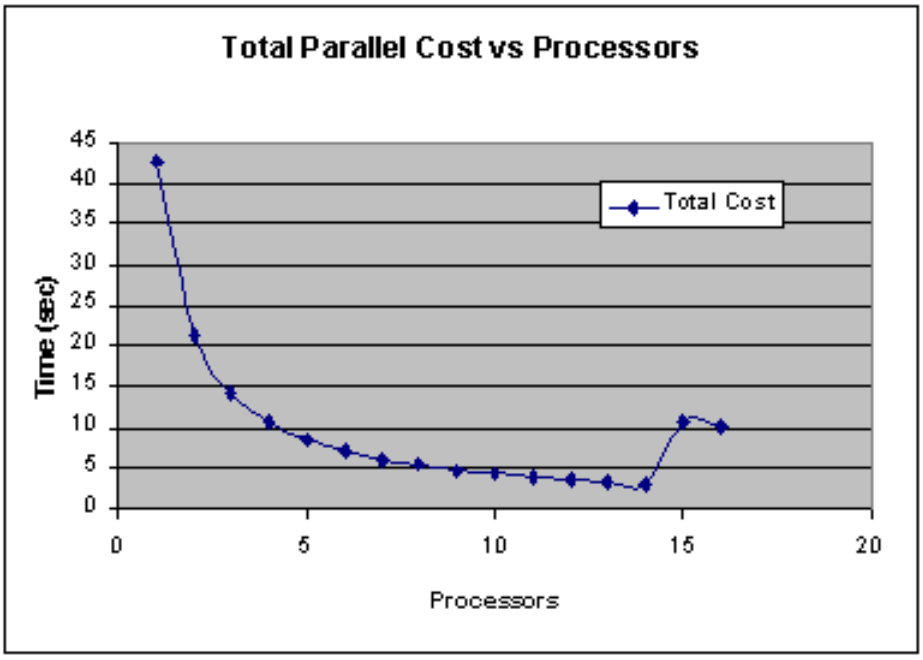

Figure 6.22 Total Parallel Cost

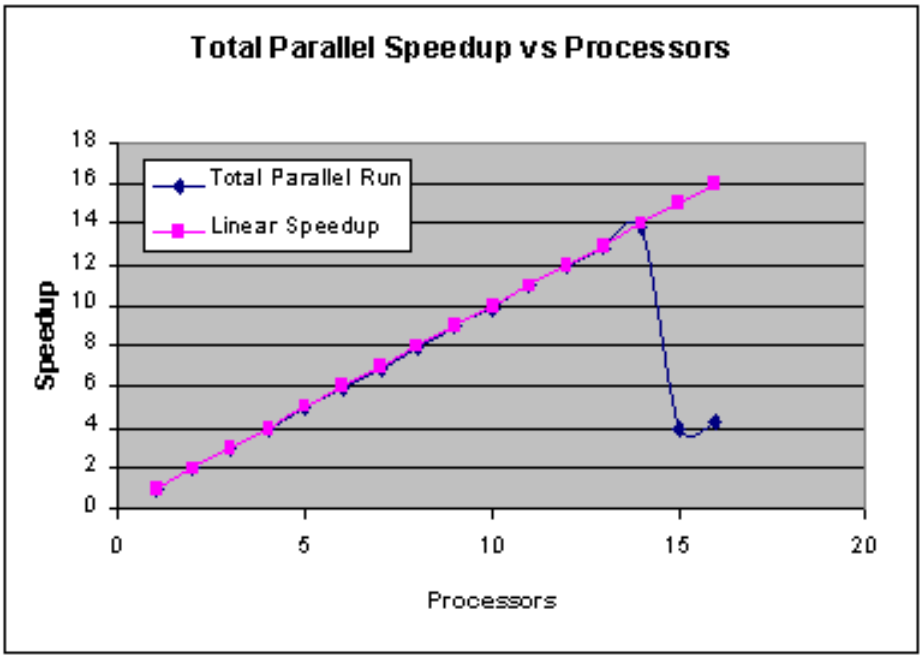

Figure 6.23 Total Parallel Speedup

contains one thorough example of the data that was used in this test and the results that were obtained.

In the particular data cube in question there is a large non-homogeneity at range cells 60-80, with other somewhat smaller inhomogeneities throughout the range cells. Other than these few small areas, the clutter is very homogeneous. A target was also injected into the data set at range cell 300. It is interesting to note that the goal of the non-homogeneity detector is not to detect targets. It is rather an attempt to locate changes in the distribution of clutter and possible additional targets, in order to leave these cells out of the weight calculations [RH99] [MW96]. This test does not prove that the addition of a target does not change the best fit distribution, nor does it prove that the addition of a target does result in the change of the best fit distribution. A more complete statistical analysis may indicate whether the existence of a target may actually change the best fit distributions and to what extent.

Injecting a target into the range cell is a non-trivial task in itself. A stand alone utility was created that prompts the user for characteristics of the target. The program reads in the entire data cube, calculate the return that would be added to the clutter by a target with specified attributes, and returns the new data cube with the target in place. In this case the target was added with the parameters given in table 6.14.

Table 6.14 Specific Target Injection Test

Target Injection Parameters	
Target Range Cell	300
Normalized Doppler	.25
Frequency	45×10^6
Element Distance	.10922
Phi Angle	-.001
Theta Angle	0
Amplitude	.1

These parameters are used to construct a space time steering vector. This is accomplished through the Kronecker product of the temporal steering vector and the spatial steering vector. This produces a particular coefficient for every element in the range cell. This coefficient corresponds to the needed amount of delay for the respective antenna

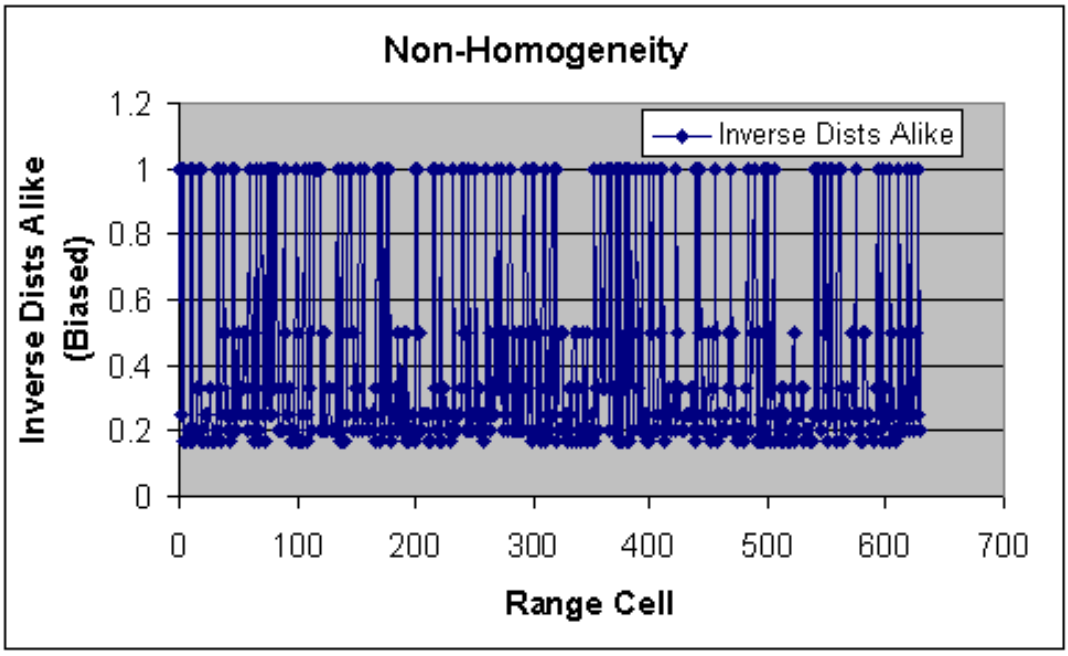

Figure 6.24 Initial Non-Homogeneity Detector Results

element and PRI. These coefficients are multiplied by the amplitude parameter. This amplitude is the strength of the return. Once the return for each element is calculated, it is then added to the clutter that already existed in the range cell of interest.

Once this test data set was complete, the new parallel non-homogeneity detector was used to process the data. The parallel viability of the design has already been validated, now the task is to demonstrate that the results are viable and useful in the general radar application. The initial results were nearly useless as noted in Figure 6.24. Since these results were so ineffective, the windowed approach discussed in Chapter III was implemented and tested. The new results with the window scaling are shown in figure 6.25. It is also interesting to note that there is a clear non-homogeneity at range cell 300 where the target was injected. In this case, the target was interpreted as a heterogeneity, however, this may not be the case in every scenario.

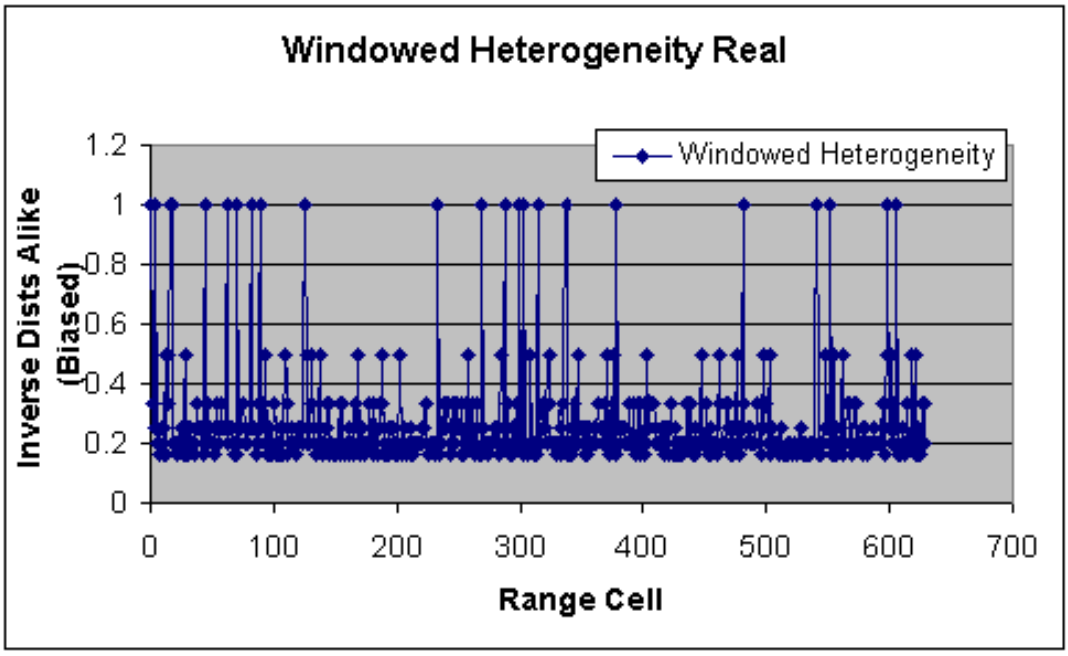

Figure 6.25 Clutter Classification Results with Windowing

6.5 Clutter Classification as a Target Detector Results

As discussed in Chapter III it also may be possible to use the Clutter Classification application as a target detector. Again, the same MCARM data set was used as previously described. As shown in figure 6.26, the target that was injected at range cell 300 is clearly visible, and of a much greater magnitude than other clutter in the data set, (Test 16).

Next, a shift in the simple mean was analyzed. It is interesting to note that the large area of clutter in the first part of the data set is extremely strong and overpowering. However, the smaller return of the target at range cell 300 is also visible. This result is reflected in figure 6.27. All of these experiments were conducted with both the real and imaginary portion of the data set. The results were nearly identical, (they showed the same non-homogeneities and targets), so the imaginary return plots are omitted in interest of brevity. When the clutter classification results and the simple mean shift results are compared in tandem, it leads one to suspect that there is a very high probability that a target may reside at range cell 300, while there is a significant amount of inhomogeneous terrain in the beginning of the data set that does not indicate target presence.

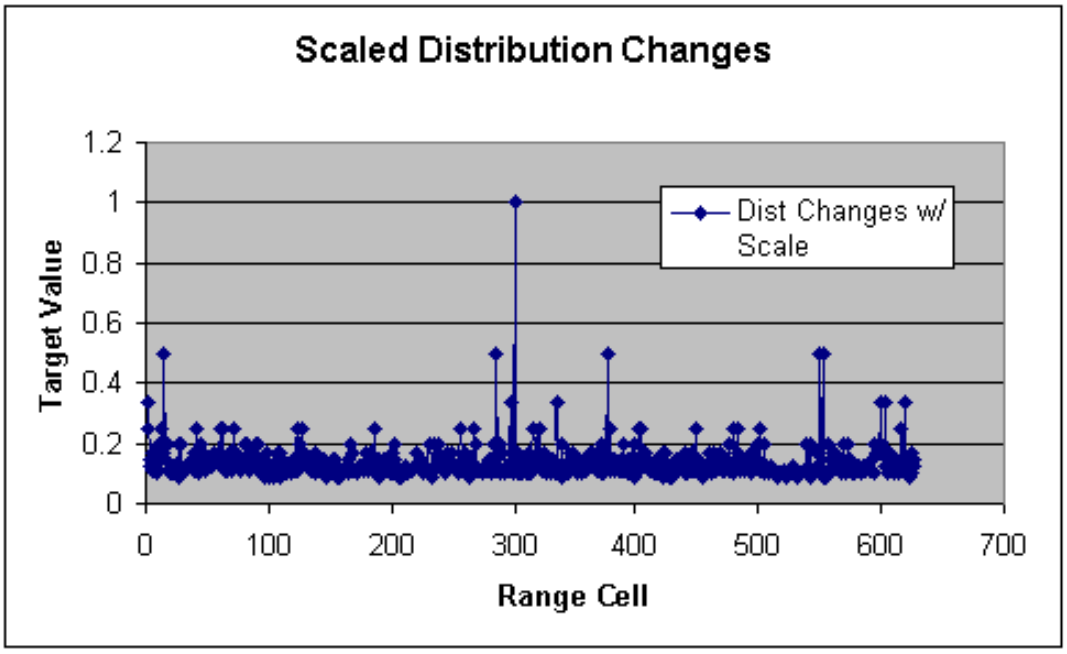

Figure 6.26 Scaled Distribution Change at Each Range

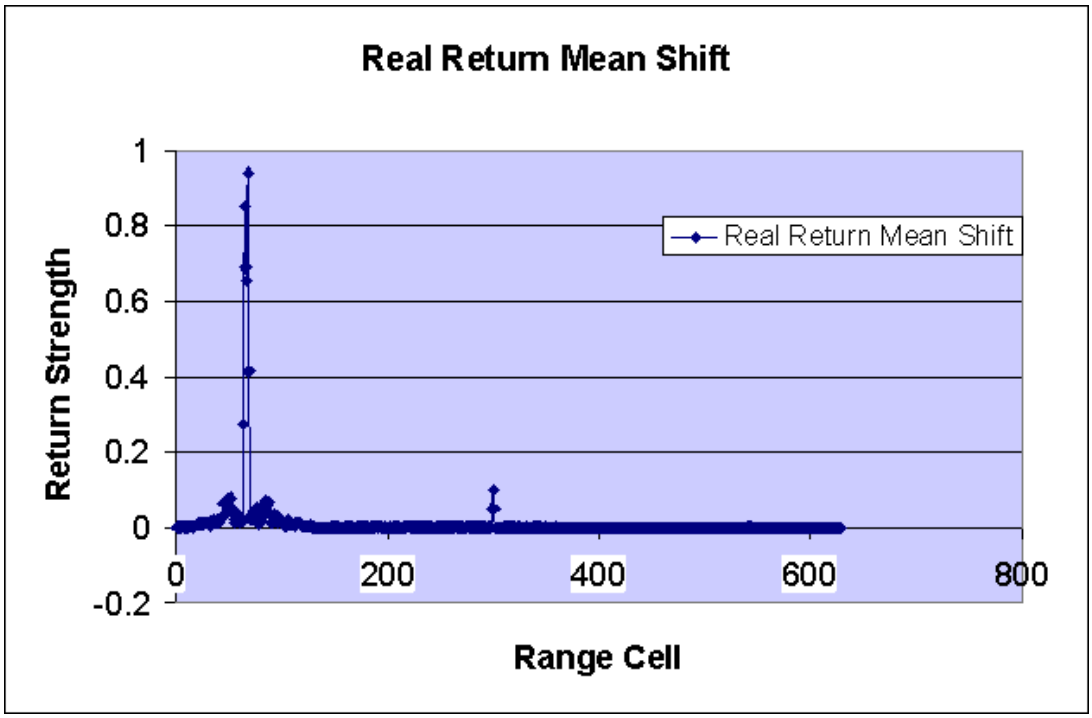

Figure 6.27 Real Mean Shift Returns

This method may not be fool proof however. This method relies on the assumption that the addition of a target to a range cell changes its best fit distributions. In reality that may not be the case. If the addition of a target does not change the distribution of that range cell, that range cell is not regarded as a target. If non-homogeneities just happen to appear in the correct distance from each other in the data cube, this may also be construed as a target when in reality it is not. It may be possible to do several runs with different scale factors to conclude if possible targets are really targets, or if they are simply non-homogeneous returns. If it were possible to quantify just how the addition of a target with specific parameters may change a base distribution, this product may be very useful. It is not nearly as computationally complex as the STAP process, and it may very well produce results that are much better if they were properly interpreted.

One must also note that this method only relates changes in local returns. This is another inherent weakness with the method. Clearly, it does tend to highlight areas where a target may reside. However, it does not expose anything about the relative strength of the return, nor does it relate any information about the expected value of the return or differences from that expected value. This may prove to be an intolerable weakness, because there is no qualitative metric that may be used to define what level of returns constitute targets, and which ones do not. If a mean shift of x was encountered between range cells that had an original strength return of y, and a second mean shift of x was observed between other range cells that had an original return strength of $10y$, there would be no distinction between the two in the simple mean shift. Also, there does not appear to be a clear intuitive way to relate the simple mean shifts and the Clutter Classification results. However, even with these weaknesses considered, the results of these tests are promising and may be productive research areas in the future.

6.6 Integrated Product Results and Analysis

This section examines the results obtained from the incremental steps and tests encountered during the creation of the final application. The first subsection deals with optimization of the data passing model. The second deals with changing the data types of the different integrated applications in order to obtain a finer level of granularity, a

consistent data type, and more congruent run times. The third and final stage deal with several optimization that were aimed at reducing the entire complexity of the Ozturk filter while maintaining accurate and reliable results.

6.6.1 Passing the Data Cube Results. The first set of optimizations analyzed deal with the passing of the data cube between the Ozturk stage and the Doppler Filter stage. There were three main solutions, each returning the expected results with one caveat. The last optimization did not behave as expected, but upon further analysis it becomes clear.

The first solution required that every byte of data be passed three times, (Test 17). Once to collect the data, once to pass the data to the next pipeline stage, and once to disseminate the data among the processors of the Doppler filter stage. The resulting run times are illustrated in table 6.15. One immediately notices that the communication times for the Ozturk send are significantly higher, (approximately a factor of two), than the first send in the pipeline before the clutter classification code was integrated The first stage in the pipeline also encounters the longest send time. This is because following stages are allowed to execute an asynchronous send, and then continue with processing. The only time a block must occur is when a process must wait to receive data before it may commence processing. Therefore, the first stage in the pipeline has the true serial time needed to send data to the following stages. This serial time before the integration of the Clutter Classification mechanism varied from 1.5 to 1.8 seconds. Therefore, this is the baseline that the new stage tries to achieve.

The second optimization that was accomplished was created by adding a distributed data passing model, (Test 18). This model allowed each process to determine where it should send and receive its data to or from. Therefore, in this manner, all of the data must only be sent once. However, this method also requires many sends. Each range cell is evaluated individually and then passed to the needed processor in its own send. The initial results of this method are shown in table 6.16.

The third and final method created was the single send model, (Test 19). This model was closely related to the second, however with one difference. Rather than evaluating and sending range cells as encountered, the range cells were marked and then sent in one large

Table 6.15 Initial Integrated Run Time Breakdown

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.6128	56.5689	3.4665	60.6481
Doppler filter	1	60.0143	0.2470	0.1254	60.3868
Easy weight	1	60.6211	0.0343	0.0029	60.6584
Hard weight	1	60.2256	0.4325	0.0001	60.6582
Easy BF	1	60.5959	0.0627	0.0001	60.6586
Hard BF	1	60.5742	0.0881	0.0001	60.6624
Pulse Compr	1	60.5243	0.1328	0.0053	60.6625
CFAR	1	60.6506	0.0113	0.0000	60.6618
Estimated Throughput		0.0160			
Estimated Latency		482.6316			
Measured Throughput		0.0170			
Measured Latency		485.4610			

Table 6.16 Individual Range Cell Send Run Times

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.6128	56.5689	1.4665	58.6481
Doppler filter	1	58.0143	0.2470	0.1254	58.3868
Easy weight	1	58.6211	0.0343	0.0029	58.6584
Hard weight	1	58.2256	0.4325	0.0001	58.6582
Easy BF	1	58.5959	0.0627	0.0001	58.6586
Hard BF	1	58.5742	0.0881	0.0001	58.6624
Pulse Compr	1	58.5243	0.1328	0.0053	58.6625
CFAR	1	58.6506	0.0113	0.0000	58.6618
Estimated Throughput		0.0170			
Estimated Latency		464.6316			
Measured Throughput		0.0170			
Measured Latency		468.4610			

send. This was an attempt to limit the overhead involved when making many sends across a network. However, the results of this method were actually much worse, (factor of 3), than the single range cell iteration method. This seems extremely odd at first glance, however, after further inspection it becomes clear why this is the case. One must keep in mind that the largest physical packet capable of traversing our network is 1500 bytes. This is because of the path MTU set for our Ethernet backbone. The size of a single range cell is 8192 bytes when $N = 16$, $M = 128$, $L = 512$, and the data type being passed is a 64 bit double. Therefore, it is impossible to pass even one range cell in a single packet. This new

implementation does not reduce the amount of overhead associated with multiple sends significantly, because the data set being passed is so large.

However, this does not explain why the new implementation would be any worse than the previous version. This also can be explained with a close look at the communication structure involved. The sends in both versions were blocking sends and receives. Because of the structure, there was some ordering imposed on the communications. The smaller communications were allowed to interleave one another, and if they blocked another communication it would only be for a short duration, because the communication was so short. If one of the larger communications blocked another large communication, the lost time was much larger than in previous implementations. The solution to this problem was to allow all the sends and receives to be accomplished asynchronously, (Test 20). This would allow the maximum interleaving of the communications regardless of ordering. This showed minor improvements over the single range cell send implementation. The results of the first large size blocking send are given in table 6.17, while the improved asynchronous send results are shown in table 6.18.

Table 6.17 Grouped Single Synchronous Sends

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.6598	57.6013	2.4404	60.7015
Doppler filter	1	60.0158	0.2532	0.1255	60.3946
Easy weight	1	60.0279	0.0343	0.0041	60.0663
Hard weight	1	60.6367	0.4283	0.0001	60.0651
Easy BF	1	60.0895	0.0626	0.0001	60.1522
Hard BF	1	60.0607	0.0892	0.0001	60.1500
Pulse compr	1	60.0128	0.1320	0.0052	60.1500
CFAR	1	60.1382	0.0112	0.0000	60.1494
Estimated Throughput		0.0167			
Estimated Latency		478.1537			
Measured Throughput		0.0166			
Measured Latency		480.6322			

6.6.2 Data Type Conversions and Optimizations. Chapter III covers many issues concerning the different data types that are used in the STAP and Clutter Classification applications. Several experiments were conducted concerning the reconciliation of these data types and the results are presented in the section, (Test 21 and 22). The new run

Table 6.18 Grouped Single Asynchronous Sends

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.6598	57.6013	1.3404	58.0015
Doppler filter	1	58.0158	0.2532	0.1255	58.0946
Easy weight	1	58.0279	0.0343	0.0041	59.0663
Hard weight	1	58.0367	0.4283	0.0001	59.0651
Easy BF	1	58.0895	0.0626	0.0001	59.0522
Hard BF	1	58.0607	0.0892	0.0001	59.0500
Pulse compr	1	58.0128	0.1320	0.0052	59.0500
CFAR	1	58.1382	0.0112	0.0000	59.0494
Estimated Throughput		0.0171			
Estimated Latency		460.1537			
Measured Throughput		0.0172			
Measured Latency		464.6322			

times associated with the integrated product using the same data type for all operations is shown below in table 6.19. Note that all of the stages are taking about $\frac{2}{10}$ of a second longer than previously attempted, however, there is still a large disparity between the Ozturk filter and the other stages in the pipeline.

Table 6.19 Double Precision Data Type Performance

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.6175	57.0401	1.4796	59.81371
Doppler filter	1	59.9554	0.4549	0.4680	59.87831
Easy weight	1	59.8822	0.0357	0.0029	59.92081
Hard weight	1	59.4705	0.4535	0.0001	59.92411
Easy BE	1	59.7538	0.1276	0.0001	59.88151
Hard BF	1	59.7200	0.1604	0.0001	59.88051
Pulse compr	1	59.6663	0.2129	0.0048	59.88401
CFAR	1	59.5718	0.2929	0.0000	59.86471
Estimated Throughput		0.0167			
Estimated Latency		478.1537			
Measured Throughput		0.0170			
Measured Latency		478.6322			

6.6.3 Complexity Reduction Results of Clutter Classification. The previous optimizations certainly help close the difference between the new first stage and the former first stage of the STAP application. However, this still leaves the problem of the huge disparity between the Ozturk filter stage computation time and the other pipeline stages

in the STAP application. However, even though there is a large disparity, the Ozturk filter stage scales very nicely with additional processors. An attempt was made to see if the addition of many processors may bring the Ozturk filter within tolerable run times comparable with the other stages. Also, one must keep in mind, if this is to be an effective filter on the front end of the pipeline, it must run significantly faster than the next longest stage of the pipeline. To facilitate this process, every machine available except for one is applied to the Ozturk filter stage. The one remaining machine is required to run every remaining process in the parallel pipeline. This is the most extreme example that may be created on the local AFIT Heterogeneous Cluster. The results of this run are available in 6.20.

As this result shows, the Ozturk filter stage computation run time does decrease dramatically, however, it is still much larger than any of the other stages, even when they are all running on one processor. The communication times between processors also rises significantly, because of the bottle neck created when passing data to and from one single machine on the network loaded with six different complex processes. It is also apparent that even if the run times of the Ozturk algorithm were within a reasonable range, this type of processor allocation would be impossible to continue as processors were added to the other parallel stages. Even in this limited case, if the run time of the other stages were cut in half by the addition of two or three processors in key locations, it would require at least a doubling of the processors that were added to the Ozturk filter in the best case. Therefore, it is possible to conclude that the Ozturk filter as created and integrated into the STAP application is, for the most part, absolutely useless.

In chapter IV, several reduction in complexity optimizations were discussed. The first is the removal of the scale, (Test 23), and the second keeps the scale, but only samples the range cells, rather than using the entire set. The first modification decreased the run time of the Ozturk filter dramatically. However, it is still too complex to be an efficient stage in the parallel pipeline. The results of this run are shown in table 6.21. It is still an order of magnitude slower than the other stages in the application. For a small number of processors, this may be alleviated by adding more processors to the Ozturk stage, however, after several processors are added, this becomes inefficient and number of

Table 6.20 Parallelization of Ozturk Filter Stage

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	13	0.4128	4.4061	200.6784	205.4973
Doppler filter	1	59.7879	2.5563	157.5530	219.8973
Easy weight	1	211.9345	0.0873	7.4799	219.5018
Hard weight	1	217.6922	2.5457	0.0001	220.2379
Easy BF	1	221.3613	0.1808	0.5305	222.0727
Hard BF	1	218.4096	1.0509	0.5063	219.9668
Pulse compr	1	218.0092	0.5002	1.0960	219.6054
CFAR	1	219.8286	0.1729	0.0000	220.0015
Estimated Throughput		0.005			
Estimated Latency		1768.1537			
Measured Throughput		0.0051			
Measured Latency		1776.6322			

processors required to execute the Ozturk stage rises too high to be efficient, (Test 24). Moreover, the quality problem comes into play here. Again, the target was injected into range cell 300 with the same parameters as previously described. Notice in Figure 6.28 that there is clearly something present at range cell 300. However, there is also too many surrounding inhomogeneities and other returns to get a clear image of where a target is, and where it is not.

Table 6.21 No Scale Ozturk Filter Evaluation

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.7316	5.9070	1.8706	8.5092
Doppler filter	1	8.1817	0.2473	0.1248	8.5538
Easy weight	1	8.4962	0.0335	0.0031	8.5329
Hard weight	1	8.1028	0.4302	0.0001	8.5331
Easy BF	1	8.4933	0.0620	0.0001	8.5554
Hard BF	1	8.4690	0.0880	0.0001	8.5570
Pulse compr	1	8.4321	0.1183	0.0050	8.5553
CFAR	1	8.5232	0.0113	0.0000	8.5345
Estimated Throughput		0.1169			
Estimated Latency		62.4562			
Measured Throughput		0.1211			
Measured Latency		63.6322			

The second modification made was to completely move to the absolute minimum amount of work needed in the application. There is a window applied to the Ozturk filter.

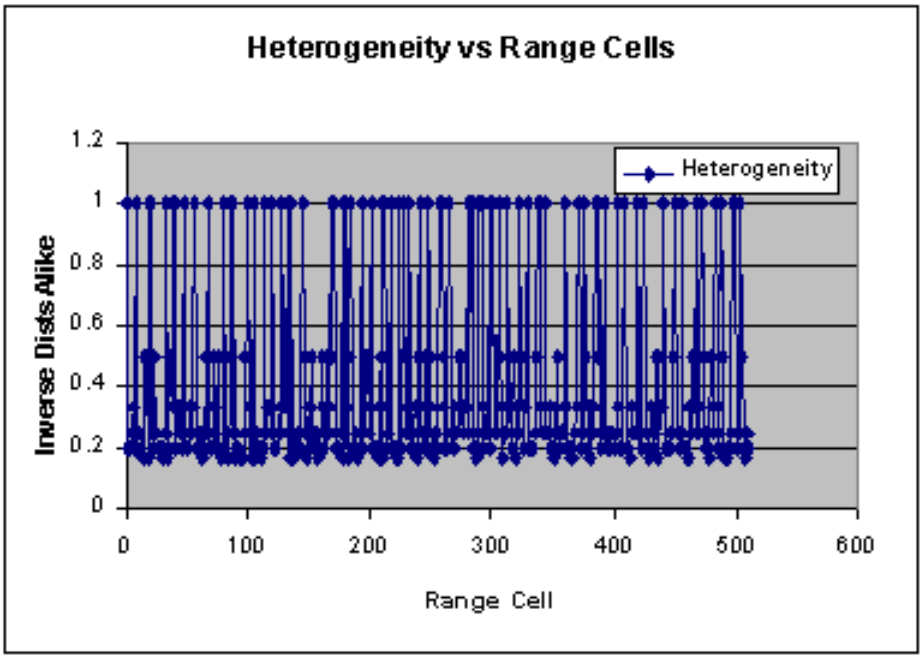

Figure 6.28 No Scale Qualitative Ozturk Results

Also, only a few data points from the each range cell is used. The typical range cell, (the ones that arrived with the STAP application deliverable), are 2048 samples. Considering this number, about five percent, or one hundred samples are used in the distribution identification process, (Test 25). The results of this minimal run are contained below in table 6.22. As shown in that table, the run time of the Ozturk filter has decreased significantly, to the point that it is now actually comparable with the other stages of the parallel pipeline. However, the qualitative question still remains. The qualitative target/inhomogeneity detection results of the random sampling technique are contained in Figure 6.29.

In light of these results, it is very difficult to get quality results at run times significantly less than the other pipeline stages without a significant amount of additional processors. Again, a mass parallelization was undertaken where all available processors, except for one is allotted to the Ozturk filter stage, (Test 26). In this manner, it is possible to understand how well this application and especially this first stage behave with additional processors. It also serves as a baseline for the possibility of speedup that may

Table 6.22 Random Sampling Scale Included

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	1	0.3993	0.7310	2.6815	3.8118
Doppler filter	1	3.4844	0.2454	0.1249	3.8547
Easy weight	1	3.8177	0.0337	0.0031	3.8545
Hard weight	1	3.4269	0.4283	0.0001	3.8554
Easy BF	1	3.7929	0.0626	0.0001	3.8556
Hard BF	1	3.7681	0.0879	0.0001	3.8557
Pulse compr	1	3.7322	0.1186	0.0049	3.8557
CFAR	1	3.8238	0.0113	0.0000	3.8352
Estimated Throughput		0.260			
Estimated Latency		31.2			
Measured Throughput		0.2597			
Measured Latency		30.822			

Table 6.23 Parallel Ozturk Random Sampling Scale Included

Pipeline Stage	Processors	Receive	Computation	Send	Total
Ozturk filter	13	0.0319	0.0575	202.6663	202.7558
Doppler filter	1	53.4177	2.5218	166.1373	222.0768
Easy weight	1	210.8143	0.0709	10.3738	221.2591
Hard weight	1	219.3631	2.4863	0.0001	221.8495
Easy BF	1	221.6477	0.1438	0.5853	222.3768
Hard BF	1	221.0112	0.8981	0.5701	222.4795
Pulse compr	1	220.8427	0.3848	1.2157	222.4432
CFAR	1	222.3354	0.0456	0.0000	222.3810
Estimated Throughput		0.0045			
Estimated Latency		1774.1537			
Measured Throughput		0.0044			
Measured Latency		1780.6322			

be achieved in the entire augmented pipeline. However, it does not address the issue that once the Ozturk filter has parsed the data and found areas of interest, it is questionable as to how much benefit is added by further processing the data cube in the STAP pipeline. The results of this Ozturk intensive parallelization are contained in table 6.23. Clearly this shows that the computation cost of the Ozturk algorithm may be significantly smaller than the other stages. However, one should keep in mind that all other processors were allocated to one machines and compare the computation cost of the Ozturk method to the cost of the other stages when they are allocated to their own processor.

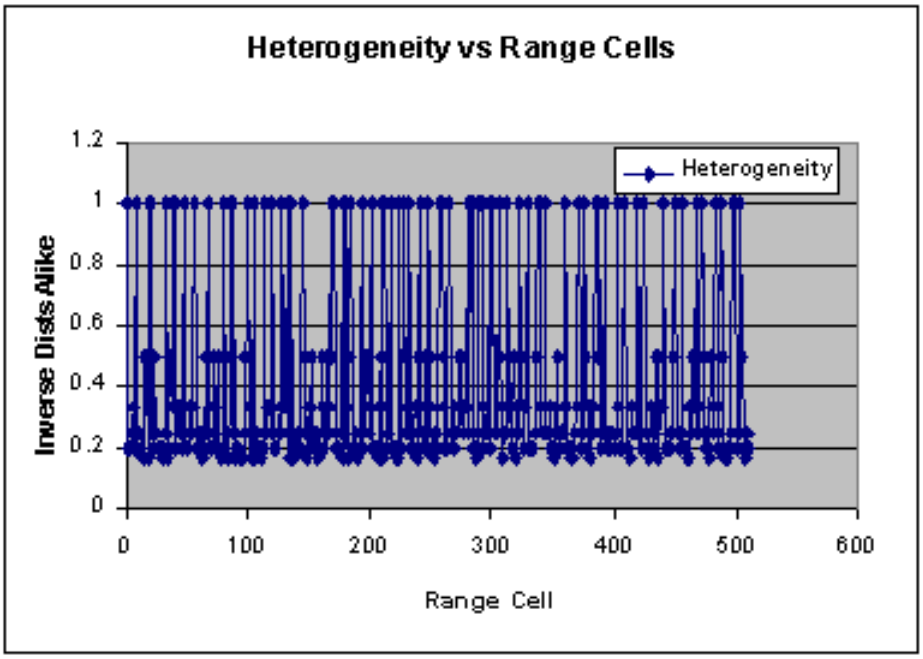

Figure 6.29 Random Sampling Qualitative Results

6.7 Parallel Pipelined STAP vs Ozturk Pipelined STAP

The final experiment ran was a comparison between the parallel pipelines STAP as delivered by Rome Labs and the newly created version using the Ozturk algorithm as a front end filter, (Test 27). As noted in previous experiments it is certainly possible to reduce the computation complexity of the Ozturk algorithm to near the same time cost as other stages of the pipeline. Furthermore, this stage scales with additional processors much better than the other stages. Therefore, it is clearly possible to reduce the cost of the Ozturk filter to a much lower time cost that that of the other stages. It now becomes a question of how many data cubes must be dropped to increase performance, as well as the many different effects of allocating more processors to different stages of the applications in different areas to speed up performance.

As noted previously, the latency of the new application is longer than the unoptimized version. However, at this expense, a gain in throughput is experienced. Lowering this cost per CPI is illustrated in Figure 6.30. As expected, as the number of CPI that are removed from the pipeline increases, the cost per CPI decreases dramatically. This is because the

pipeline may operate at the rate of the Ozturk filter for cubes that may be dropped, whereas it must operate at the cost of the longest stage for cubes that remain in the pipeline. Therefore, it is indeed possible to increase the throughput of the STAP pipeline by filtering the input data. However, this optimization results in an increase in latency that is the cost of the longest pipeline stage. Whether this is an acceptable tradeoff or not is a qualitative question dependent upon real time constraints.

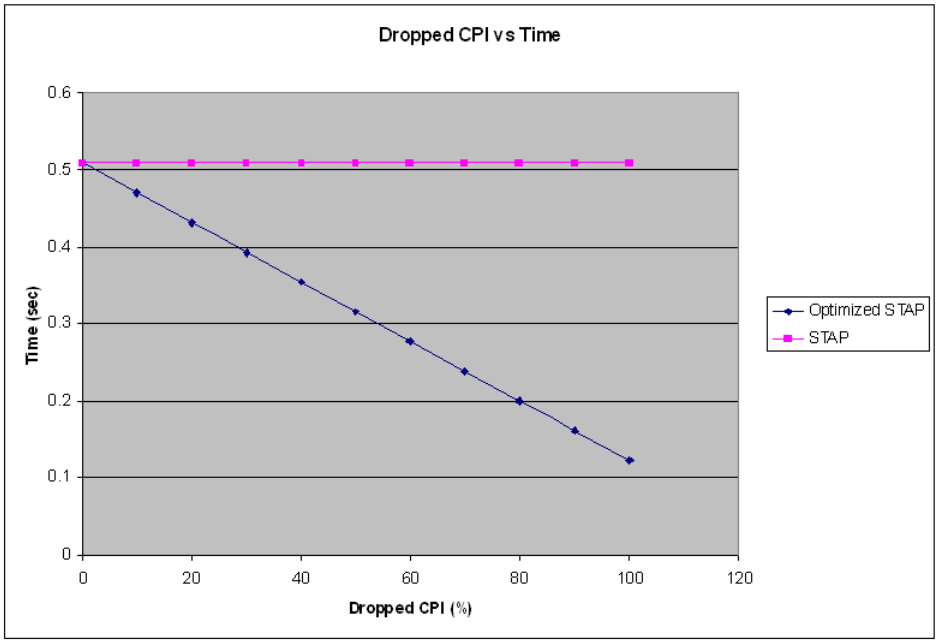

Figure 6.30 Effects of Discard Ratio on Time Cost

6.8 Analysis Summary

In conclusion, the Chapter VI addresses analysis and results. It contains the results of the experimentation described in Chapter V. In this chapter, results are displayed from the particular experiments conducted, along with generalized trends that have been found during the experiments. These results serve as a foundation for the observations and conclusions that are presented in Chapter VII. This chapter has presented a thorough discussion of the separate applications and their performance attributes in isolation. It also discusses the use of the Clutter Classification application as both a non-homogeneity detector as well as a target detector. The Clutter Classification is analyzed both qualitatively

and quantitatively; its parallel run time behaviors are discusses as is its effectiveness. Then the performance of the integrated product is discussed in many different aspects including both efficiency and effectiveness metrics.

VII. Conclusion and Future Work

In conclusion, it is clear that the integration of Space Time Adaptive Processing and Clutter Classification may actually increase the throughput significantly (Test 27). However, there are serious issues with using this integration in a real time application. Throughput and latency are directly dependent upon the operating environment, which is not a desirable attribute for a real time system. One would like to know how long each data cube takes to process, regardless of the characteristics of surrounding cells.

The nature of the parallel pipeline raises some issues as well. As pipeline theory suggests, and our mathematical models show, unless the cost of the first pipeline stage (the Ozturk filter) is significantly smaller than the cost of the surrounding stages, dropping data cubes after processing them with the Ozturk filter is pointless because the bubble injected into the pipeline has the same cost as processing a true data cube. The process relies on the ability of the Ozturk filter to drop more than one data cube in the time cost required to process through the longest stage of the pipeline. Therefore, if the Ozturk filter is not significantly shorter than the other stages, one should process every cube through the STAP pipeline and disregard the Ozturk filter all together.

Because of the run-time behavior of the Ozturk filter application, it is not appropriate as a front end filter for the STAP application. To create a filter that is powerful enough to detect possible targets and eliminate other data cubes from the pipeline, the cost of the pipeline is too high to be a viable option. It is important to note that a good front end filter should have a higher false positive rate than the more work-intensive following stages of the pipeline, but this also was not the case. It appears that the clutter classification does well at identifying targets, however, empirical evidence suggests that Clutter Classification may actually have a lower false positive than the STAP application. In essence, Clutter Classification determines whether there may be a target, and then the STAP application reproduces the same conclusion. There is no clear reason why one would wish to do this in general.

Therefore, the fundamental premise of this research, the useful integration of Clutter Classification and the STAP application, was a mixed success. The integration itself is

functional and does produce accurate results (Tests 21-26). However, it is not efficient, and the work is redundant in many locations. Also, because of the pipelined nature of the STAP application, the use of the Ozturk filter as a front end is essentially useless (Tests 21-26). Even though it works well to separate interesting and non-interesting data sets, there is no great speedup realized because of the speed of the initial stage. This result is related to pipeline fundamentals, pipeline throughput calculations, and the fact that dropping data from the pipeline and replacing it with bubbles does not result in any real speedup. Therefore, the main objective, though reached, is not a particularly good solution, and should be abandoned.

On a positive note, the Ozturk Clutter Classification application itself is extremely promising. This application appears to have the ability to detect targets, detect inhomogeneity, and produce reliable results in a known expected time. Also, it has been shown to be extremely well formed for a parallel environment. It scales linearly with added processors, at least to as many processors as are available on the AFIT cluster. Judging by the structure of the application, there is no reason to believe that it would not continue to scale linearly until the processor count nearly reaches the number of range cells in the data cube. This new application has been shown to be orders of magnitude more efficient than the original FORTRAN port (Test 6). Not only is the serial algorithm much better than the original code, this new port is also highly parallel in nature. It has been shown to scale nearly linearly with additional processors (Tests 7-14). Qualitatively the results are also good. It appears, (but is not statistically proven), that Clutter Classification does have the ability to identify radar returns that are likely to contain areas of interest.

In summary, this research effort combines Space Time Adaptive Processing with Clutter Classification in an effort to increase the efficiency and effectiveness of real time radar technology. It is important to both Department of Defense and civilian interests in pursuit of advancements in radar technology that may increase its effectiveness and efficiency. STAP has been extensively researched, and is known to produce quality results. However, computational complexity forbids quality real time implementation, hence the focus of this research. The Ozturk algorithm is also well researched. However, using the Ozturk method as a non-homogeneity detector and target detector are entirely new and

exciting concepts. The integration of the STAP and Clutter Classification applications is also a completely new idea. Although this integration was for all practical purposes a failure, other aspects were great successes. The Clutter Classification is clearly of great benefit.

This area offers much for future work, but not in the initial direction that was taken for this thesis. It would be useful to focus on the capabilities of the Clutter Classification application. A thorough mathematical and statistical analysis of the hypotheses and conjectures made in this thesis would be value. It is not understood what statistical and mathematical properties apply to the addition of targets to radar returns. Some extremely important questions arise: Does the introduction of a target into a radar return statistically change how that return is distributed? Furthermore, how do these statistical changes apply when the area of analysis spans local range cells? These questions indicate significant areas for future research and development.

Appendix A. Distributions Searched In Clutter Classifiation

1. Normal
2. Uniform
3. Exponential
4. Laplace
5. Logistic
6. Cauchy
7. Extreme
8. Gumbel
9. Gamma
10. Pareto
11. Weibull
12. LogNormal
13. K-Distribution
14. Beta(0.2)
15. Beta(0.4)
16. Beta(0.8)
17. Beta(1.6)
18. Beta(3.2)
19. Johnson(-0.7)
20. Johnson(-0.4)
21. Johnson(-0.2)
22. Johnson(-0.1)
23. Johnson(0.0)

24. Johnson(0.2)

25. Johnson(0.4)

26. Johnson(0.8)

27. Johnson(4.0)

Table B.1- B.3 contain a small section of the first datacube that was given by Rome Labs for this research project. All of these returns are from the real domain, and are provided to understand the order of magnitude of the short integers that were contained in the datacube. Values this large were not expected to reside in the data cube.

Table B.1 Short Integer Data Cube Section

Element	Value	Element	Value	Element	Value
1	256.000000	26	633.000000	51	2105.000000
2	0.000000	27	-1054.000000	52	2394.000000
3	0.000000	28	-123.000000	53	2973.000000
4	0.000000	29	48.000000	54	2450.000000
5	28771.000000	30	-391.000000	55	1433.000000
6	-3328.000000	31	1101.000000	56	364.000000
7	-1076.000000	32	2134.000000	57	-154.000000
8	1074.000000	33	856.000000	58	-410.000000
9	977.000000	34	-1676.000000	59	6.000000
10	-598.000000	35	-2779.000000	60	113.000000
11	-2004.000000	36	-2139.000000	61	-800.000000
12	-372.000000	37	-1308.000000	62	-1725.000000
13	3172.000000	38	-507.000000	63	-1117.000000
14	3865.000000	39	-285.000000	64	527.000000
15	1993.000000	40	-272.000000	65	2958.000000
16	738.000000	41	389.000000	66	2798.000000
17	-611.000000	42	755.000000	67	390.000000
18	-1139.000000	43	363.000000	68	-1851.000000
19	-779.000000	44	-650.000000	69	-2324.000000
20	-2353.000000	45	-232.000000	70	-2596.000000
21	-2488.000000	46	1545.000000	71	-960.000000
22	21.000000	47	1484.000000	72	1229.000000
23	1255.000000	48	499.000000	73	2225.000000
24	1055.000000	49	959.000000	74	529.000000
25	1645.000000	50	2183.000000	75	-840.000000

Table B.2 Short Integer Data Cube Section

Element	Value	Element	Value	Element	Value
76	-663.000000	101	-1010.000000	126	878.000000
77	1700.000000	102	-1858.000000	127	-586.000000
78	2378.000000	103	-1780.000000	128	140.000000
79	2048.000000	104	-858.000000	129	2413.000000
80	1348.000000	105	-699.000000	130	2190.000000
81	-40.000000	106	208.000000	131	315.000000
82	-674.000000	107	-448.000000	132	-675.000000
83	-213.000000	108	-1295.000000	133	50.000000
84	-1858.000000	109	-571.000000	134	-1415.000000
85	-3551.000000	110	628.000000	135	72.000000
86	-2157.000000	111	-481.000000	136	1124.000000
87	-544.000000	112	-1040.000000	137	1654.000000
88	1451.000000	113	530.000000	138	827.000000
89	2521.000000	114	2588.000000	139	913.000000
90	1241.000000	115	3986.000000	140	283.000000
91	-420.000000	116	3902.000000	141	-127.000000
92	260.000000	117	2402.000000	142	-265.000000
93	1844.000000	118	1374.000000	143	351.000000
94	643.000000	119	-592.000000	144	1865.000000
95	394.000000	120	-1152.000000	145	2721.000000
96	-722.000000	121	202.000000	146	2576.000000
97	-522.000000	122	1178.000000	147	1813.000000
98	-276.000000	123	2079.000000	148	-255.000000
99	500.000000	124	2258.000000	149	-1664.000000
100	572.000000	125	1991.000000	150	-1642.000000

Table B.3 Short Integer Data Cube Section

Element	Value	Element	Value
151	-711.000000	176	-3224.000000
152	-783.000000	177	-2028.000000
153	2360.000000	178	947.000000
154	1935.000000	179	3425.000000
155	616.000000	180	3323.000000
156	738.000000	181	508.000000
157	2552.000000	182	-810.000000
158	2569.000000	183	-1008.000000
159	-28.000000	184	-632.000000
160	-2767.000000	185	539.000000
161	-2383.000000	186	1629.000000
162	240.000000	187	2154.000000
163	2314.000000	188	2626.000000
164	3306.000000	189	3761.000000
165	1175.000000	190	2438.000000
166	-1567.000000	191	-2.000000
167	-557.000000	192	184.000000
168	424.000000	193	1246.000000
169	-1584.000000	194	288.000000
170	-1419.000000	195	-1240.000000
171	-374.000000	196	-888.000000
172	-233.000000	197	989.000000
173	-11.000000	198	1750.000000
174	-850.000000	199	2118.000000
175	-2439.000000	200	1581.000000

Appendix C. Integrated Product Startup File

This appendix contains the startup file that is used by the integrated STAP and Clutter Classification application. These parameters are contained in Table C.1 along with a breif description describing what it represents.

Table C.1 Startup Parameter File

Parameter	Value	Description
-k	512	number of range cells
-j	16	number of channels
-n	128	number of pulses
-r	3	number of reference CPIs
-m	26	total number of CPIs (besides the reference CPIs)
-p	3	number of zero padding
-w	Hanning	windowing function: Hanning or Hamming
-h	56	number of hard Doppler bins
-e	26	number of range samples for easy weight
-u	0.3333	fraction of range cells for extracting easy weight samples
-s	39	number of range samples for hard weight
-g	6	number of segments for each hard Doppler bin
-l	5	number of broad transmit beams
-d	2	broad transmit beams direction
-b	6	number of receive beams for each broad transmit beam
-V	SVs	filename of the steering vector (in Matlab 4.0)
-c	0.5	beam constraint weight
-f	0.05	frequency constraint weight
-o	0.6	forgetting factor
-C	replica	filename for replica array used in pulse compression
-a	2	number of guard cells for the sliding window
-i	10	number of range cells for the window size
-q	12.	7 false alarm factor
-v	0.0001	probability of false alarm for order statistic CFAR
-y	0.0 7	guessing left boundary root of solving threshold equation
-z	100.0	guessing right boundary root of solving threshold equation
-x	0.00001	accuracy of bisection root finding for solving threshold
-t	17	order number for order statistic CFAR
-R	17400	recording start range (in meters)
-S	1.0E6	A/D sampling frequency (in Hz)
-N	16	number of bits representing one CPI element
-P	61.1E-6	transmit pulse width (in seconds)
-F	450.0E6	transmit frequency (in Hz)
-B	0.5E6	transmit bandwidth (in Hz)
-D	0.333	azimuth element spacing (in meters)
-A	90	mechanical boresight azimuth (degree)
-E	3	mechanical boresight elevation (degree)

Appendix D. Processor Allocation and Communication Ordering Optimization

D.1 Introduction

Space Time Adaptive Processing is a statistical method to evaluate returns from an array of radar antennae. This statistical analysis relies on different returns spatially and temporally to inject nulls into the return in the direction of clutter and jamming. This process is known as adaptive, because the weighting parameters are calculated from the actual return. In this manner, it is possible to suppress clutter and jamming noise with no prior knowledge of the return or landscape. The only drawback with this method is the computational complexity [JW94]. It requires a vast amount of computing power to process these returns real time. Rome Labs, NY, has a parallel pipelined version of STAP that incrementally processes the data in parallel stages [CA96]. However, processors must be allocated to these stages manually, and there is no method to determine if a current configuration is optimal. The data passed through this pipeline must be passed from stage to stage via Message Passing Interface. However the ordering of these messages may make a difference in performance depending on the communications backbone in use. These are the two optimizations that are addressed in this paper.

The structure of this report follows the scientific process in general. First, in section D.1, an introduction and overview of the problem is discussed, processor allocation and message passing order. Section D.2 addresses evolutionary techniques in general , as well as how they specifically apply to the problems at hand. Section D.3 addresses the actual design of the experiment, the parameters that are used, the specific test cases, and the rational for choosing them. The results of these experiments are presented in section D.4. Analysis and Conclusions are presented in sections D.5 and D.6 respectively.

D.1.1 Processor Allocation Problem. This specific problem dealt with, actually has little to do with the STAP application itself. STAP is computationally intense process, and the results of that process are time critical. The end goal is to create a system capable of processing radar returns real time. Towards that end, Rome Labs has developed a parallel pipelined version of STAP. The different STAP computations are broken into pipeline stages, and then the stages themselves are ran in parallel. This solution is one step

closer to the real time processing capability, however, it also introduces other complications as well. Each processing stage requires a different amount of computation than its sibling stages. Therefore, a "good" allocation of processors to stages must be determined to make efficient use of the limited resources at hand.[CA96].

This problem essentially becomes a multiple knapsack problem. In the new version of the parallel pipeline, there are eight stages that all accomplish a certain task. Therefore, each stage must have at least one processor. Other than that, there are really no restrictions on the allocation of processors. In a typical knapsack problem, a zero/one formulation may be used to mathematically represent the problem description. This problem requires the modification of this description, (shown below in equation D.1). Rather than a 1 or a 0 meaning that an item is either in or out of the knapsack, the range of values increases to 8 in this case. Zero would indicate that a processor is not in use, while a value from 1-7 would indicate that the processor has been allocated to that particular group. The metric of measurement to indicate the usefulness of a solution would be total run time in the longest pipeline stage.

$$\begin{aligned} & max\{cost(stage1), \ldots, cost(stage8)\} \\ & \quad where: \quad cost(stageX) \propto \frac{1}{\sum_{i=0}^{32} \xi \times 1} \end{aligned} \tag{D.1}$$

There is little difference between pedagogical trivial examples of this problem and the actual real world problem at hand. A multiple knapsack problem is not new uncharted territory. The only difference is the actual calculation of the fitness function. Rather than maximizing the number of items stored in the knapsacks, the objective of the optimization is to minimize the largest knapsack, and thus decrease the operational length of the pipeline. This increases both throughput and latency of the entire STAP application. Due to the modification of this multiple knapsack problem, as well as the allocation constraints, the search space for this problem is:

$$7^n < Complexity < 8^n \tag{D.2}$$

D.1.1.1 Genotype/Phenotype. The fitness or usefulness of a particular solution is said to be its phenotype. In simple terms it is the outward observed appearance without regard to the structure that created the observed fitness. In this case, the phenotype is the time that the longest stage in the parallel pipeline costs. The genotype on the other hand is the particular underlying structure that allows the observed characteristics to be observed. These are the actual genes in the structure, whether they are observed in the individual or not. In this particular case, the genotype consists of a set of processors that have or have not been assigned to particular stages of the pipeline. Even though two solutions may exhibit nearly the same phenotype, their underlying genotypes may differ greatly.

D.1.2 Communication Ordering Problem. The communication ordering problem is another combinatorial optimization problem that is directly applicable to the parallel pipeline STAP. This optimization is an attempt to order communication between different stages in the pipeline in such a manner that overhead, traffic, and collisions are reduced in order to increase throughput and decrease latency. It is possible that every processor in one stage must communicate with every processor in the next stage. These communications are not temporally dependent upon each other; the only requirement is that all communications are completed before computation in that stage may commence. Therefore, any possible ordering of these communications is completely viable, and results in a search space of *communications*!.

The fitness of a particular solution is much more difficult to estimate in this situation. To facilitate this evaluation, an application that executes a given ordering of communications was created. The chosen Evolutionary Algorithm then generates the solutions according to the particulars of that specific algorithm. In all of the possible evolutionary techniques a fitness function is required. This application serves as the fitness function in the chosen solution for implementation. [JW00b] This problem is also very similar to its pedagogical counterparts. In fact if it weren't for the need to time the execution of a separate binary in the fitness evaluation, it would have been possible to easily attack this problem with current existing applications. This problem is essentially a traveling sales-

man problem. However, rather than the salesman moving from place to place to create a tour, the tour consists of an ordering of message sends. It is also the case that in this problem, the number of processors never increases above 24, (the maximum amount of processors that can be connected at one time on the AFIT switch), and this problem is not that complex. It may be possible to find a deterministic algorithm to solve for the optimal answer. However, since this exercise is academic in nature, evolutionary methods are used to attack this problem.

D.1.2.1 Genotype/Phenotype. The phenotype for this optimization is the time costs for the entire group of messages sent. It is not the order that they were sent, but rather the time costs and fitness values that result. Again, there is no regard to the actual parameters that make up this result. The genotype on the other hand is the ordering of the solution that resulted in the particular fitness that was returned. In this case, it is a permutation selected from one of the possible orderings of all the messages that must be sent.

D.2 Evolutionary Computation Domain

There are several possible Evolutionary Algorithms that may be chosen for implementation. Each has its own strengths and weakness. The end goal is to gain some knowledge of the search space through exploration, and then exploit that knowledge to find a close-to-optimal solution in minimal time. A short discussion of several of these algorithms follow, as well as an in depth discussion on the algorithm of choice.

D.2.1 Evolutionary Programming. Evolutionary programming was one of the earliest developed evolutionary algorithms. It models Darwinian selection and evolution. In its pure form, it is elitist in nature; it is always striving for a better solution, discarding intermediate solutions that may be unfit. The mutation operator is the only operator that is used to perturb the population. This mutation operator changes the individual member according to a given statistical distribution. One major difference between this method and other methods is the fact that it does not rely on good building blocks. With many optimization problems, the joining of good building blocks does not necessarily result

in good solutions. Therefore, in landscapes where local optimizations in concert do not yield favorable results, a evolutionary programming technique may be a good solution. A mathematical description of a general EP follows:[TB00]

$t := 0;$
initialize $P(0) := \{a'_1(0), a'_2(0), \ldots, a'_\mu(0)\}$
evaluate $P(0) : \{\Phi(a'_1(0)), \Phi(a'_2(0)), \ldots, \Phi(a'_\mu(0))\}$
iterate
{
mutate: $P'(t) := m_{\Theta_m}(P(t))$
evaluate:$P'(t) : \{\Phi(a'_1(t)), \Phi(a'_2(t)), \ldots, \Phi(a'_\lambda(t))\}$
select: $P(t+1) := s_{\Theta_s}(P'(t) \cup Q)$
$t := t + 1;$
}

When considering the specific problem domain for the Processor Alocation Problem, the symbols from the above algorithm correspond in the following manner:

- a' is a population member. In this case it is a possible allocation of processors to pipeline stages. In reality it is a string of thirty-two integers that have values ranging from 0-8.
- μ is the size of the parent population. In this case it is determined by the user according to parameters desired at runtime.
- λ is the size of the offspring population. Again this is determined by the user at runtime.
- $P(0) := \{a'_1(0), a'_2(0), \ldots, a'_\mu(0)\}$ is the population at time t. Specifically, it is all the different combinations of processors that are being tested at iteration time t.
- Φ is a fitness mapping function. Specifically it is the run time of the longest stage in a parallel pipeline for a specific a'.
- m_{Θ_m} is the mutation operator with parameters. This is the operator that perturbs the population member and adjusts it in a direction according to a distribution.

- s_{Θ_s} is the selection parameters. Specifically this is an elitist selection that selects the μ best members of the population and sets them as the new parent population.
- Q is the parent solutions. These must be accounted for, because the elitist selection algorithm is a $(\mu + \lambda)$ type operation.

Evolutionary programming may be applied to the communication ordering problem domain as well. One merely must create a data structure for the problem, and then apply the same operators in the algorithm to the new problem domain. Using this technique, it is possible to apply the same algorithm to many different problem domains with minimal effort.

- a' is a population member. In this case it is a specific permutation of communications that must be sent from one parallel stage to the next.
- μ is the size of the parent population. In this case it is determined by the user according to parameters desired at runtime.
- λ is the size of the offspring population. Again this is determined by the user at runtime.
- $P(0) := \{a'_1(0), a'_2(0), \ldots, a'_\mu(0)\}$ is the population at time t. Specifically, it is all the different combinations of permutations of communications that are being tested at iteration time t.
- Φ is a fitness mapping function. Specifically it is the communication time for a specific ordering a'.
- m_{Θ_m} is the mutation operator with parameters. This is the operator that perturbs the population member and adjusts it in a direction according to a distribution.
- s_{Θ_s} is the selection parameters. Specifically this is an elitist selection that selects the μ best members of the population and sets them as the new parent population.
- Q is the parent solutions. These must be accounted for, because the elitist selection algorithm is a $(\mu + \lambda)$ type operation.

D.2.2 Evolutionary Strategies. Evolutionary strategies were developed independently from genetic algorithms. However, over time they have grown increasingly similar. Just as with the typical genetic algorithm, an evolutionary strategy relies on selection, mutation, evaluation, and recombination. However, how this is actually accomplished is somewhat different that the typical genetic algorithm. Selection is generally elitist. Premature convergence is avoided by a larger population size. This encourages less fit individuals to survive longer. Mutation comes in two varieties: Object parameter mutation and Strategy parameter mutation. Both these types of mutation are more closely related to the mathematical mutation based on distributions that is found in Evolutionary Programming. However, they are more complicated and different in how the parameters are created and how the parameters themselves may be mutated. Recombination is very similar to the crossover that is found in typical GA's.[FH01]

D.2.3 Genetic Algorithm. The genetic algorithm is another evolutionary technique that uses the idea of building blocks via the schema theorem to create near-optimal solutions to problems that are typically too complex to solve deterministically to an optimal answer. In the most general sense, a genetic algorithm relies on crossover, mutation, evaluation, and selection to explore and exploit large search spaces. The different types of these operators are limitless, and more are introduced every day. Most are founded in biology and immunology, however others have their roots in different evolutionary functions. The general algorithm for a GA follows:[TB00]

$t := 0;$
$P(t) := initialize(\mu)$
$P(0) := evalutate(P(t), \mu)$
while$(\iota(P(t), \Theta_\iota) \neq true)$
{
$P'(t) := recombine(P(t), \Theta_r);$
$P''(t) := mutate(P'(t), \Theta_m)$
F(t) := evaluate:$P''(t), \lambda)$
$P(t+1) := select(P''(t), F(t), \mu, \Theta_s);$

$t := t + 1;$

}

Given this mathematical definition of a generic outline of a genetic algorithimn, it is much easier to apply this to our specific problem at hand. In the case of the processor allocation problem, the above parameters correspond to the specific data structures and operations outlined below:

- t is the iteration or generation count.
- $P(t)$ is the population at time t. In this scenario, this would correspond to a set of possible solutions or processor allocations. These allocation would be a string of integers that range from 0-8
- μ is the parent population.
- λ is the next generation population, or all the children.
- $F(t)$ is the fitness function. It evaluates the different members of the population. In this scenario it is the time cost of the longest stage of the pipeline.
- Θ_ι are the stopping conditions. It may be a certain number of iterations, a measure of convergence, or in this particular case, a threshold where real time processing speed has been achieved.
- Θ_r are the recombination parameters. These may vary, even within the supplied solution. For example, two point, single point, and uniform selection are all available.
- Θ_m are the mutation parameters. This may be a rate of mutation, as well as how the integer string is mutated. It may be a shuffle, inversion, random changes, or many other schemes that are well known.
- $Theta_s$ are the selection parameters. In this case, there are two different sets of parameters. One is a roulette wheel. The selection pressure is not extremely high, and it tends to keep some slower solutions which may still contain good building blocks in the population. The other solution is elitist; it only keeps solutions that provide the shortest maximal pipeline stage.

In the same manner, it is possible to apply the same mathematical expression to the communication ordering problem. Many of the parameters may be significanlty alike. Therefore, if the code is constructed correctly, it may be possible to use the same algorithm to optimize the two different problems. All that must change is the fitness function. It must constructed such that each type of population member may be correctly analyzed.

- t is the iteration or generation count.
- $P(t)$ is the population at time t. In this scenario, this would correspond to a set of possible communication orderings. These allocation would be a string of integers that range from $0 - MessageCount$. Any permutation of communications is an acceptable solution.
- μ is the parent population.
- λ is the next generation population, or all the children.
- $F(t)$ is the fitness function. It evaluates the different members of the population. In this scenario it is the amount of time taken to accomplish all of the sends between two pipeline stages.
- Θ_ι are the stopping conditions. It may be a certain number of iterations, a measure of convergence.
- Θ_r are the recombination parameters. These may vary, even within the supplied solution. For example, two point, single point, and uniform selection are all available.
- Θ_m are the mutation parameters. This may be a rate of mutation, as well as how the integer string is mutated. It may be a shuffle, inversion, random changes, or many other schemes that are well known.
- $Theta_s$ are the selection parameters. In this case, there are two different sets of parameters. One is a roulette wheel. The selection pressure is not extremely high, and it tends to keep some slower solutions which may still contain good building blocks in the population. The other solution is elitist; it only keeps solutions that provide the minimal amount of communication time.

D.2.4 Genetic Programming. Genetic programming is the most different of the four mentioned evolutionary algorithms. In the previous methods, the actual implementation was static. Only the data set was manipulated and changed in a evolutionary manner. Using this technique, the actual program that produces the results is the object of evolution creativity and pressure. A population of random programs are created. These programs are judged by their fitness, (ability to create the correct output). The actual programs are mutated and changed. Operators and terminals may be added or removed by the mutations. They are also recombined. Different subtrees in the program may be exchanged with subtrees of other programs or within themselves. These recombined mutated programs are the offspring. Just as with other evolutionary techniques, selective pressure is allowed to push the fitness of the population toward the correct solution.[JK01]

At first glance, neither of the particular problems addressed here are well suited for the genetic programming paradigm. The problem of is not trying to create a program capable of computing the tasks, but rather one of intelligently covering an extremely large search space. There is little doubt that genetic programming could create a program capable of solving the problems at hand, however, just as with the deterministic optimal search, it would probably take too long to run.

D.2.5 Implementing A Solution: Genetic Algorithm. The chosen solution for these problems is a genetic algorithm. The method seems particularly well suited for the problems at hand. This is specifically because of the data structures that may be employed as well. It is very easy to represent the chromosome as a string of integers that may take on different values. It is also easy to recombine and mutate these values. Evolutionary programming and evolutionary strategies may also work quite well with a somewhat different data structure representation. The only hurdle one must overcome is the mutation complexities. Both of these algorithms require that mutation occur on individuals within a given distribution, even though that distribution and its parameters are not necessarily static. It may be somewhat difficult to implement these qualities without creating a more complicate representation. For this reason, a simple genetic

algorithm was chosen for this optimization and should produce quality results with minimal programming complexity.[TB00]

D.3 Design of Experiments

Given the genetic algorithm discussed in section D.2.5, one must now conduct experiment to test the effectiveness and effeciency of this implementation. The first problem that is addressed is the processor allocation problem experiment design, after which, the message ordering problem may be addressed. Because of background knowledge gained in these specific problem domains, it is possible to gear the experiments toward these applications to acheive better results.

D.3.1 Processor Alocation Experiment Design. First and foremost, it is already known that the fitness function for any solution to this problem takes about a minute to run. Hopefully as the solutions get better, this time actually decreases. However, the experiement is designed with this worst case run time in mind. To keep the number of fitness calculations to a minimum, the population size is kept small. This decreases the amount of fitness calculations that must be accomplished for each generation, and increase the amount of generations that may be completed in minimal time. The next operator that is tuned for this specific application is the selection algorithm. The selection algorithm is extremely elitist in nature. The reason for this is to force premature convergence of the solution set. This causes the quality of the solutions to be worse than if the selection algorithm did not apply so much selective pressure to the population, however, if this pressure is not supplied, it is likely that no quality solution is found in reasonable time. he elitism in this application is so strong that the best $\frac{1}{2}$ of the population is always kept, and the worst half is discarded. The mutation rate is also be kept low. This lowers the chance that new genetic material is injected into the population. The basic goal of this experiemnt is to generate a population and gain the best solution that may be found by recombining that genetic material in a 12 hour window. The parameters for the conducted experiements are contained in table D.1.

Table D.1 Processor Alocation Problem Parameters

Experiment #	Pop Size	Mutation Rate	Generations
1	20	.05%	50
2	25	.5%	50
3	30	1%	50

Considering this problem, there is no clear delineation between what may be considered a benchmark and what may be the actual real world problem. The STAP application itself would never be boarded on an airborn platform on a cluster of linux PC's. The most likely system for this application would be a mercury super computer. Therefore, this application in itself is simply a benchmark tool and proof of concept platform. This in turn implies that the optimization of this benchmark would be a benchmark itself. However, this is as close to the real world application as one may get without boarding it on specific hardware and testing it in a "real" environment. Other than minory hardware considerations, there is little difference between the optimizations that must be made on the AFIT cluster and the optimizations that must be made on an airborn super computer. There is really no way, nor is there desire to "dumb down" this problem to a pedagogical benchmark. The whole point of the the genetic algorithm is to optimize the runtime of a real binary. If it is simplified, it no longer is a true measure of the goal.

The main performance metric that is evaluated is throughput. There are several other metrics that are interesting, but in reality they are all closely related to throughput and somewhat superfulous in that light. Other possible metrics that are of interest would relate to the actual performance of the genetic algorithm, such as generations, convergence, percent of optimal, and other metrics that may expose bottlenecks and weaknesses in the actual genetic algorithm rather than in the STAP application. Unfortunately, due to time constraints, it is not feasible to run many instances of the same genetic algorithm. This is becuase it takes neary 24 hours to complete one run with fifty generations. Therefore, it is impossible to say something statistical about behavior of the program with any certainty. It may be that the results of the experimentation are completely abnormal; until further testing is completed there is no way to know that this is a ususual or unusual result.

Table D.2 Communication Ordering Experiment Parameters

Pop Size	Mutation Rate	Generations
30	.05%	100
		1000
		10000
	.5%	100
		1000
		10000
	1%	100
		1000
		10000
60	.05%	100
		1000
		10000
	.5%	100
		1000
		10000
	1%	100
		1000
		10000
90	.05%	100
		1000
		10000
	.5%	100
		1000
		10000
	1%	100
		1000
		10000

D.3.2 Communication Ordering Experiment Design. The design of this experiement is much different than the design of the processor alocation problem, even thought it uses the same genetic algorithm for the optimization problem. The only thing that has been changed is the fitness function, and what the items in the chromosome represent. The repair function also had to be slightly modified to account for repeated messages in the tour. This main problem with the previous experimentation was the fitness function bottleneck. That is not the case in this scenario. Therefore, the parameters for this optimization may be varied and perturbed much more than in the previous case. There is no reason to have an extremely high selection pressure as was the case with the processor allocation. There is no need to force premature convergence. The selection pressure was decreased through the use of roulette wheel selection. There is also much more leeway granted in the population sizes that may be used during experimentation. A much wider range of population sizes were used, as indicated in table D.2. The mutation rate was also varied significantly. The parameter test values for the entire experiment are contained in table D.2.

Because of the ability to run more experiments in this configuration, a statistical analysis is much more useful for this optimization. Each of the experiments described in table D.2 was completed five times. This allows for a statistical comparison of what parameters settings are more useful than others, as well as which parameter setting allow for the best performance overall.

D.4 Experimentation Results

This section discusses the results obtained by the actual experimentation covered in section D.3. First the results of the processor alocation problem are discussed. Theses results are quite interesting and may prove to be very useful in future work. Next, the communication ordering results are covered. However, due to the nature of the communication backbone on the AFIT cluster, as well as measuring methodologies, the actual output of this experiment is basically useless in the light of evolutionary optimization. Even though it certainly did not result in what may be considered desireable output, it certainly is interesting to understand the behavior of the AFIT cluster's networking hardware and software.

Both of these optimization were ran on the AFIT linux cluster alone. There are several reasons for this, the main reason being the need for many processors to effectively test the optimization problem. If the network of workstations were used, the maximum amount of processors available is eight. Given that the minimum processes needed to run the STAP application is seven, this does not make for an interesting optimization problem. If it were ran on the windows side of the AFIT cluster, the situation is even worse. It is not possible to run the STAP application on the windows side of the cluster without doubling up processes on several processors. Again, this would be useless application of the algorithm since there is not enough processors available for computation.

D.4.1 Processor Alocation Results. The processor allocation optimization problem yielded very interesting results. When optimizing a pipeline, it seems to make sense that one would simply add more processors to the longest stage in the pipeline. However, that is simply not that case with the STAP application on the AFIT heterogeneous cluster.

If one merely throws more processors at the longest stage, it is highly unlikely that the optimal solution is the result. For example, lets assume that the hardest stage in the pipeline is the hard weight calculation, (in reality it is the most difficult). There are currently three processors assigned to that stage, and each of those processors are 1.7GHz PIV machines. Even with three processors in this stage, it is still the longest stage in the system; it must run faster in order to speed up the pipeline. The addition of another processor in this stage does not garuntee that the pipeline stage does indeed speed up. In fact, if the processor added is the "wrong" processor it may slow the entire stage worse than without the additional processor. When work is divided among the parallel processors of a stage, each processor recieves the same amount of work regardless of processor capability. The entire stage takes at least as long to process as the longest running process in the parallel stage. Therefore, if the fourth processor takes longer to process the smaller amount of allotted to it than the faster processors took to process the somewhat larger data set themselves, the additional process only hinders the pipeline.

This hetrogeity property may be exploited by a genetic algorithm, because it makes to attempt to make locally optimal decisions to create a usable solution. Because of this, the genetic algorithm has the ability to create solutions that may place the "right" processors in the correct proportions along the parallel pipeline stages. First, the latency reduction results for experiment 1, 2, and 3 are shown in figures D.1, D.2, and D.3 respectively. One must also keep in mind that just because one experiment appears to have better results than the next, that may not really be the case. This is because there has been no statistical analysis on this application. It takes nearly 24 hours to complete 50 generations of this optimization, and the current results have to suffice. From these results, it is hard to say that one method is "better" than the next. However, it is clear that in all three cases, the total run time was reduced by a very significant amount. After the processor optimization, the STAP application ran nearly twice as fast as it previously did in all cases. This is a better tuning than has been found by manually optimizing the STAP pipeline and processor allocation by hand.

However, what is even more interesting is the final configuration of machines that was encountered during the optimization program. With all of the available processors, it

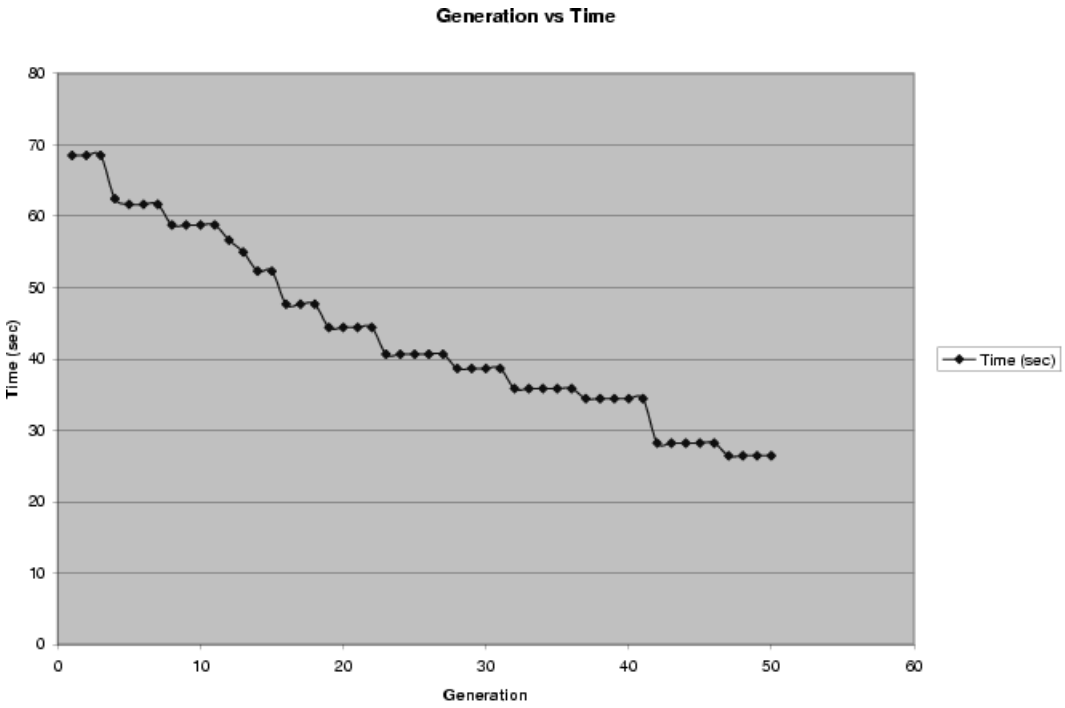

Figure D.1 Experiment 1 Processor Allocation Results

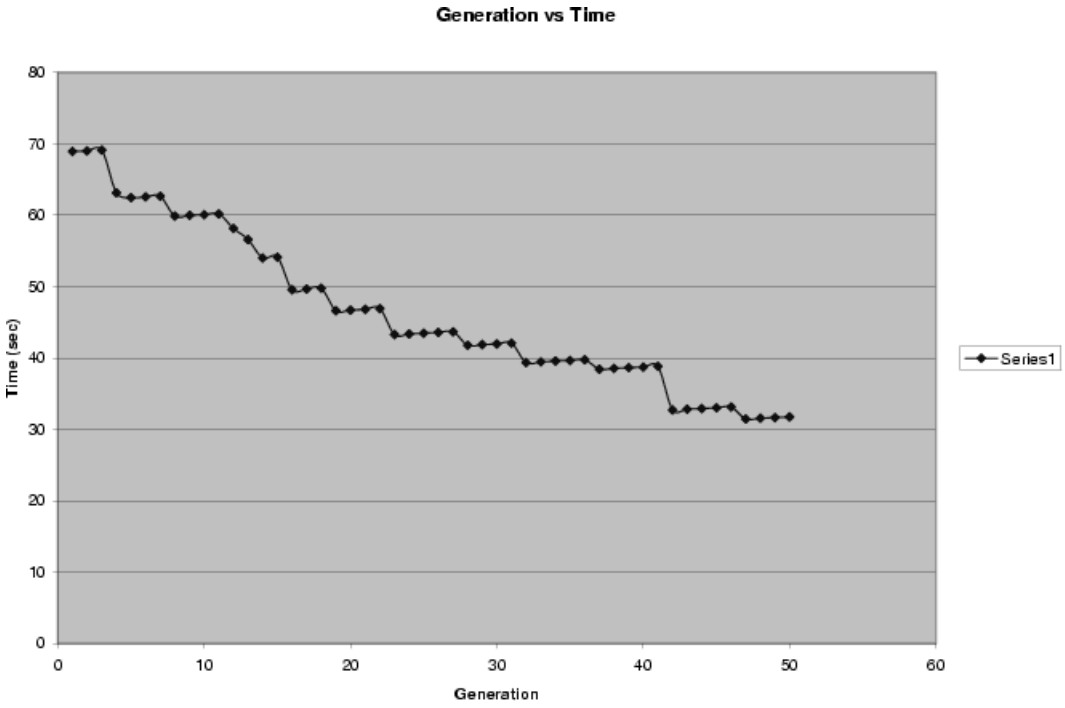

Figure D.2 Experiment 2 Processor Allocation Results

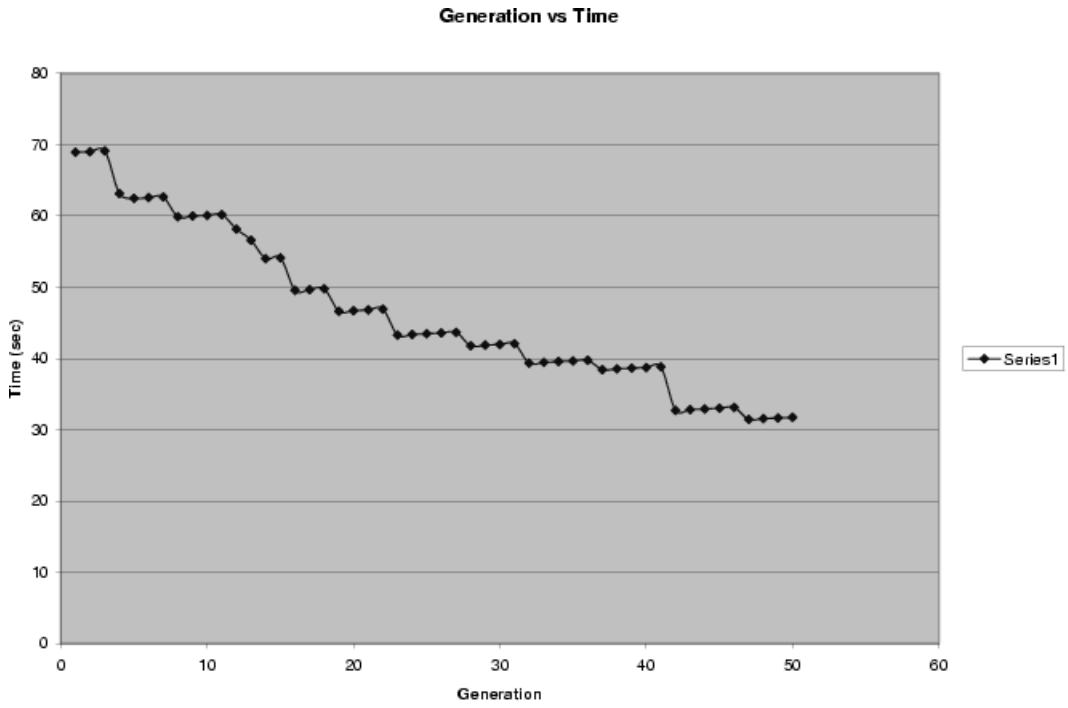

Figure D.3 Experiment 3 Processor Allocation Results

is interesting to note how the slower machines were actually left out of the solution entirely. The configuration of the machines for the best run found is given in table D.3. Note that there are no machines that are slower than 1GHz. This is interesting considering that by random generation, around $\frac{1}{3}$ of all the machined found should be slower than 1GHz.

D.4.2 Communication Ordering Results. Unfortunately, from an evolutionary algorithms standpoint, the results of the communications ordering optimization are useless for the most part. This is because it turns out that on our particular cluster, the ordering of communications is irrelevant for the most part. In reality this may not be the case, however with current methods of timing those communications, it is impossible to reach the granularity needed to accurately describe how long each particular tour takes. Below is a basic ordering of all the major operations that are required in the genetic algorithm. Each is discussed along with its impact on the optimization. After examining the many layers of abstraction involved with ordering the communications, it becomes clear why it is extremely difficult to get a reliable measure of how long it takes to actually complete a send. When looking at the results for this optimization graphically, it is clear that ordering

Table D.3 Best Solution Processor Alocation

Processor Stage	Processors
Doppler Filter	ABC-B14 ABC-B11
Easy Weight	ABC-B12
Hard Weight	ABC-B13 ABC-B18
Easy Beamform	ABC-B19
Hard Beamform	ABC-B20 ABC- B22
Pulse Compression	ABC-B23
CFAR	ABC-B24 ABC-B35

is not a dominant factor in calculating performance. This is because timing even the same ordering mulitple times may result in significantly different return values.

1. Start time stamp – An initial time stamp must be taken. This is accomplished in the java driver program. Therefore, the virtual machine must interface to the real machine, the real machine must make a system call, and the call must return after the interrupt was handled with the number of ms that have elapsed.

2. MPI program spawn –The next step is to spawn the MPI binaries on all the different machines. This is accomplished with an rsh command. This in itself is clearly not an exacting process; it may take different time for all the systems to respond, spawn the correct process, and synchronize with the other processes.

3. MPI Communications – The actual communications must occur. Depending upon the type of communications, these processes are not extremely predictable. It requires that the operating system be notified and then respond to the program and send the messages. There are two types of sends, and each has its own complications:

   - MPI_Send – This is a blocking call that has problems when trying to accurately measure the latency. This is because the call blocks, allows the operating system to take control, pass the messages, and then return control to the program. This does not always behave in the same manner.

- MPI_Isend – Even though it no longer blocks, there are even more complications with this type of send. To do an asynchronous send, a separate process must be spawned that completes the send, notify completion of the send, and allow the master to contiue. Each of these actions take time, and the amount of time differs under like conditions

4. Kill the Process – After the program completes, it must terminate and then pass control back to the calling java program. Just as in all the other actions, the control passing does not complete at a consisten time cost.
5. Final Time Stamp – Just as with the starting time stamp, the final stamp suffers from the same shortcomings.
6. Clock Granularity – The final problem encountered is clock granularity. It does not take much time at all to pass a message from one machine to the next. This small amount of time becomes lost in the other times that are included in the time stamp process and cannot be avoided. It seems that taking these time stamps is merely measuring the noise in the system, without accurately gathering detail.

Because of these reasons, it is impossible to get an accurate account of what time it takes to actually pass the messages at a fine enough granularity to determine if one message ordering is better than another. This is graphically protrayed in figure D.4. Clearly there appears to be no real correlation between order, latency, or efficiency.

D.5 Analysis

It is difficult to imply that the results found in section D.4 mean anything about quality offuture results and consistency of those results. This is because too few runs were conducted. The only thing that can truly be said is that it appears that the application has the ability to optimize the processor allocation problem. One may not do it better than another, and it clearly is not enough results to conclude anything significant about the parameter settings. However the apparent ability of the program to leave certain processors out of the run and to isolate better processors to certain stages in certain locations is very promising. Unfortunately, the results from the communication ordering

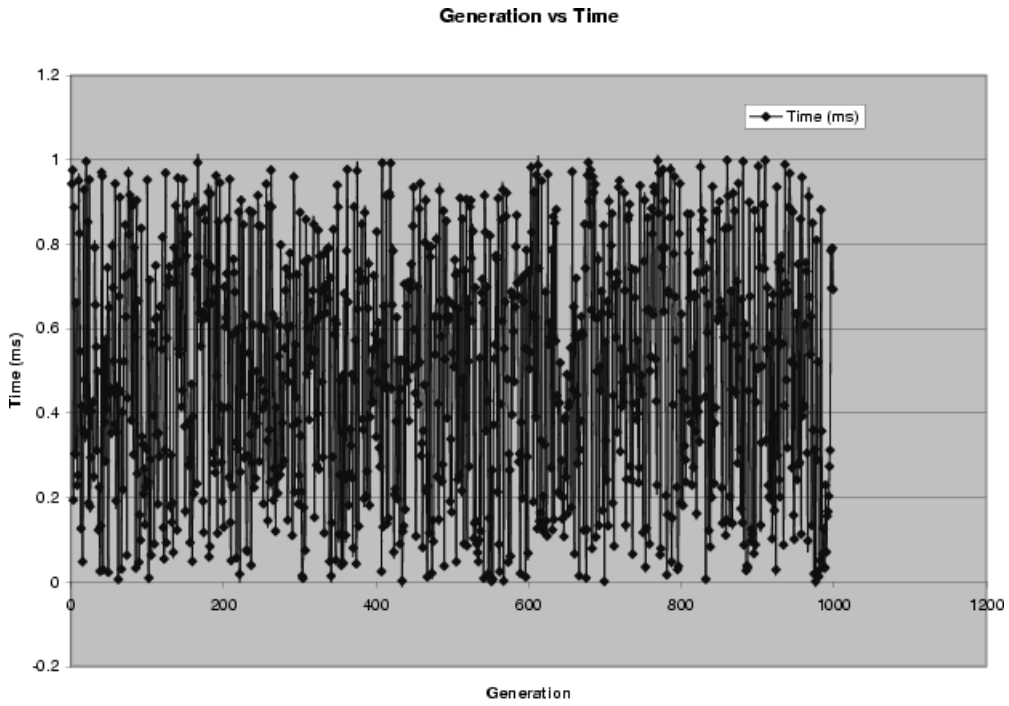

Figure D.4 Communication Ordering Optimization Results

problem were useless for the most part. However, it does bring up some interesting points about how MPI and the communications backbone on the AFIT cluster function.

D.6 Conclustions and Future Work

In conclusion, the only real known is that there is promising reseach yet to be conducted in the use of a genetic algorithm to optimize the alocation of processors for the STAP and other similar applications. It would be extremely useful to conduct an entire suite of tests to fully understand the potential of this method, as well as the parameters that yeild the best results for this specific problem domain. The communication ordering problem may offer better results with a different implementation as well. If it were possible to accurately measure the communication times, as well as model different communications backbones, this may also offer significant improvement to the STAP application.

This appendix contains the hardware configuration of the several parrallel clusters based at AFIT. Three main cluster are utilized, AFIT Heterogeneous Cluster, AFIT Homogeneous Cluster, and AFIT Cluster of Workstaions.

Table E.1 Homogeneous AFIT Cluster Hardware Configuration

Processor Count	Processor	Physical RAM (MB)	Network Speed (Mb/s)	Operating System	Number of Processors
16	Athalon 1.2GHz	786	100	Red Hat 7.1	1

Table E.2 AFIT Cluster of Workstations Hardware Configuration

Processor Count	Processor	Physical RAM (MB)	Network Speed (Mb/s)	Operating System	Number of Processors
8	Sparc Ultra 10	1000	100	Solaris 5.8	1

Table E.3 Heterogeneous AFIT Cluster Hardware Configuration

Hostname	Processor	Physical RAM (MB)	Network Speed (Mb/s)	Operating System	Number of Processors
Linstar	Pentium III 600	512	1000	Red Hat 6.2	2
ABC-A3	Pentium III 600	384	1000	Red Hat 6.1	1
ABC-A4	Pentium III 600	384	1000	Red Hat 6.1	1
ABC-A5	Pentium III 600	384	1000	Red Hat 6.2	1
ABC-B5	Pentium III 600	384	100	Red Hat 6.1	1
ABC-B6	Pentium III 600	384	100	Red Hat 6.1	1
ABC-B7	Pentium III 600	384	100	Red Hat 6.1	1
ABC-B8	Pentium III 600	384	100	Red Hat 6.1	1
ABC-B9	Pentium III 600	384	100	Red Hat 6.1	1
ABC-B10	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B11	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B12	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B13	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B14	Pentium III 400	256	100	Red Hat 6.2	1
ABC-B15	Pentium III 450	256	100	Red Hat 6.2	1
ABC-B16	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B17	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B18	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B19	Pentium 4 1700	512	100	Red Hat 7.1	1
ABC-B20	Pentium III 1000	512	100	Red Hat 6.2	1
ABC-B21	Pentium III 1000	512	100	Red Hat 6.2	1
ABC-B22	Pentium III 1000	512	100	Red Hat 6.2	1
ABC-B23	Pentium III 1000	512	100	Red Hat 6.2	1
ABC-B24	Pentium III 1000	512	100	Red Hat 6.2	1
ABC-B25	Pentium III 1000	512	100	Red Hat 6.2	1
ABC-B26	Pentium III 933	256	100	Red Hat 6.2	1
ABC-B27	Pentium III 933	256	100	Red Hat 6.2	1

[RH99] Adve, Raviraj S., et al. "Transform Domain Localized Processing Using Measured Steering Vectors and Non-Homogeneity Detection." *Journal of Multivariate Analysis, 46*. 285–290. 1999.

[TB00] Bäck, T., et al. *Evolutionary Computation 1* (1st Edition), *1*. Institue of Physics, 2000.

[CB01] Bell, Colin R. "Microbiology Mutations." http://plato.acadiau.ca/courses/biol/Microbiology/mutation.htm, November 2001.

[PB95] Bhat, P., et al., "Issues in using Heterogeneous HPC Systems for Embedded Real Time Signal Processing Applications," 1995.

[JB01] Blamir, John. "Biology Background." http://www.brooklyn.cuny.edu/bc/ahp/BioInfo/GP/Definition.html, November 2001.

[BFS87] Bratley, P., et al. *A Guide to Simulation*. Springer-Verlag, 1987.

[RB89] Bronson, Richard. *Schaums' Outlines: Matrix Operations*. McGraw-Hill, 1989.

[JB98] Burley, James C. *GNU Project Fortran Compiler*. Free Software Foundation, Inc, 1998.

[KC98] Cain, Kenneth C. and Brian Sroka. *Portable Software Library Optimization*. Technical Report MTR 98B0000037, 1998.

[AC99b] Choudhary, Alok, et al. *User Manual For a Parallel Pipelined PRI-Staggered Post-Doppler STAP Application*. Technical Report, United States Air Force Rome Labratory, 1996.

[CA96] Choudhary, Alok, et al. "Design, Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers," *unknown* (1996).

[AC99a] Choudhary, Alok., et al., "Design, Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers."

[TC00] Cormen, Thomas H., et al. *Introduction to Algorithms*. McGraw-Hill Book Company, 2000.

[DD97] Dasgupta, Dipankar and Nii Attoh-Okine. "Immune-Based Systems: A Survey," *IEEE International Conference, October 12-15* (1997).

[DD99] Dasgupta, Dipankar, et al. "An Immunologic Approach to Spectra Recognition," *Gecco Conference, July 13-17* (1999).

[DD94] Deitel, H. M. and P. J. Deitel. *How to Program C*. Prentice Hall Publishing, 1988.

[NRL00] Division, NRL Radar, "Space Time Adaptive Processing."

[GF99] Fried, George H. and George J. Hademenos. *Schaum's Outline: Biology* (2nd Edition), *1*. McGraw-Hill, 1999.

[DG95] Gilly, Daniel. *Unix in a Nutshell: System V v 2.0* (2nd Edition). O'Reilly and Associates, 1995.

[KG96] Grassman, Karl and Jean-Paul Tremblay. *Logic and Discrete Mathematics*. Prentice Hall, Inc, 1996.

[JG95] Greenfield, John, et al. *Introduction to MPI*. University of New Mexico, 1999.

[NG98] Gupta, Nikhil D., et al. "Reconfigurable Computing for Space-Time Adaptive Processing." *IEEE Symposium on FPGAs for Custom Computing Machines*, edited by Kenneth L. Pocek and Jeffrey Arnold. 335–336. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[TH01] Hale, Todd, "MCARM data set," 2001.

[JH98] Hennessy, John L. and David A. Patterson. *Computer Organization and Design*. Morgan Kaufman Publishers, Inc, 1998.

[FH01] Hoffmeister, F. and T. B"ack. "Genetic Algorithms and Evolution Strategies." http://www.cpsc.ucalgary.ca/ dawasoni/533/webnotes/main.html, November 2001.

[KH99] Hwang, K., et al. "Resource Scaling Effects on MPP Performance: The STAP Benchmark Implications," *IEEE Transactions on Parallel and Distributed Systems*, *10*(5):509–517 (1999).

[CC01] Jensen, Nathan A. "Clutter Classification C Code.".

[BK88] Kernighan, Brian W. and Dennis M Ritchie. *The C Programming Language*. Prentice Hall Publishing, 1988.

[JK01] Koza, John R. "Genetic Programming Tutorial." http://www.genetic-programming.com/Tutorial, November 2001.

[VK94] Kumar, Vipin, et al. *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms* (2nd Edition). Addison-Wesly, Inc, 1994.

[FC01] Laboratory, AFRL Rome. "Clutter Classification FORTRAN Code.".

[JL96] Lebak, James M., et al. *Toward a Portable Parallel Library for Space-Time Adaptive Methods*. Technical Report 96-242, 1996.

[ML97] Lee, M. and V. Prasanna, "High Throughput-Rate Parallel Algorithms for Space Time Adaptive Processing," 1997.

[WL99a] Liao, Wei-Keng, "Design and Evaluation of I/O Strategies for Parallel Pipelined STAP Applications."

[WL99] Liao, Wei-Keng, et al., "Multi-Threaded Design and Implementation of Parallel Pipelined STAP on Parallel Computers with SMP Nodes," 1999.

[WL99b] Liao, Wei-Keng, et al. "I/O Implementation and Evaluation of Parallel Pipelined STAP on High Performance Computers." *HiPC*. 354–358. 1999.

[ML98] Linderman, Mark H. and Richard W. Linderman. "Real-Time STAP Demonstration on an Embedded High Performance Computer," *IEEE Transactions on Aerospace and Electronic Systems* (1998).

[WL98] Liu, W. and V. Prasanna, "Design of Application Software for Embedded Signal Processing."

[MW96] Melvin, William L. *Non-Homogeneity Detection for Adaptive Signal Processing*. Technical Report, United States Air Force Rome Labratory, 1996.

[MW00] Melvin, William L. "Space-Time Adaptive Radar Performance in Heterogeneous Clutter," *IEEE Transactions on Aerospace and Electronic Systems* (2000).

[ZM94] Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs* (2nd Edition). Springer-Verlag, 1994.

[UTK01] of Tenessee, University, "DopplerEffect."

[AO91] Ozturk, Aydin. "A general algorithm for univariate and multivariate goodness-of-fit tests based on a graphical representation," *Communications in statistics-theory and methods*, *20*(10):3111–3137 (1991).

[AO93] Ozturk, Aydin. "An Application of a Distribution Identification Algorithm to Signal Detection Problems." *1993 Conference Record of the Twenty Seventh Asilomar Conference on Signals, Systems, and Computers*. 248–252. 1993.

[AO92a] Ozturk, Aydin and E. J. Dudewicz. "A new statistical statistical goodness-of-fit test based on graphical representation," *Biometrical Journal*, *43*(4):403–427 (1992).

[AO92b] Ozturk, Aydin and J. L. Romeu. "A new method for assessing multivariate normality with graphical applications," *Communications in Statistics, Simulations, and Computation*, *21*(1):15–34 (1992).

[PP96] Pacheco, Peter. *Parallel Programming with MPI*. Morgan-Kaufmann Publishing, 1996.

[CP00] Peckham, C. D., et al. "Reduced-Rank STAP Performance Analysis," *IEEE Transactions on Aerospace and Electronic Systems* (2000).

[RP01] Pressman, Roger S. *Software Engineering: A Pratitioner's Approach*. McGraw-Hill, 2001.

[RM01] Rangaswamy, Muralidhar, et al. "Performance Analysis of the Nonhomogeneity Detector for STAP Applications," *IEEE* (2001).

[RM95] Rangaswamy, Muralidhar, et al. "Computer Generation of Correlated Non-Gaussian Radar Clutter," *IEEE Transactions on Aerospace and Electronic Systems* (1995).

[RO93] Romeu, J. L. and A. Ozturk. "A Comparative Sudy of Goodness-of-Fit Tests for Multivariate Normality." *Journal of Multivariate Analysis, 46*. 309–334. 1993.

[TKS01] Sarkar, Tapan Kumar, et al. "Deterministic Least-Squares Approach to Space-Time Adaptive Processing (STAP)," *IEEE Transactions on Antennas and Propogation*, *49*(1) (2001).

[YS98] Seliktar, Yaron, "Space-Time Adaptive Monopulse Processing," 1998.

[YS96] Seliktar, Yaron, et al. "Evaluation of Parially Adaptive STAP Algorithms on the Mountain Top Data Set," *Proceedings of 1996 IEEE Conference on Acoustics, Speech, and Signal Processing*, *2*:1169–1172 (1996).

[US98] Shenoy, U. Nagaraj, et al. *A Parallel Goodness of Fit Test Algorithm for Realtime Applications*. Technical Report, Center for Parallel and Distributed Computing, 1998.

[FS99] Silva, Fernando. Master of Science in Computer Science, Air Force Institute of Technology, 1999.

[SF99] Silva, Fernando and Gary B. Lamont. "Parallel Space Time Adaptive Processing on a Cluster of Personal Computers." *11th Symposium on Computer Architecture and High Performance Computing*. 1999.

[SL94] Slaski, Lisa K. and Murali Rangaswamy. *A New Efficient Algorithm for PDF Approximation*. Technical Report, United States Air Force Rome Labratory, 1994.

[PT01] Techau, Paul M., et al. "Effects of Internal Clutter Motion on STAP in a Heterogeneous Environment," *Proceedings of 2001 IEEE Radar Conference*, 204–209 (2001).

[NAV01] University, Rice, "Phased Array Radars," 2001.

[JW94] Ward, J. *Space-Time Adaptive Processing for Airborne Radar*. Technical Report, Lincoln Laboratory, Massachusetts Institute of Technology, 1994.

[WJ94] Ward, J. *Space-Time Adaptive Processing for Airborne Radar*. Technical Report, Lincoln Laboratory, Massachusetts Institute of Technology, 1994.

[XM00a] Wei, Wang, et al. "A Simple and Effective Reduced-Rank STAP Approach." *Proceedings of ICSP2000*. 2000.

[JW00] West, Jack M. *A Genetic Algorithm Approach to Scheduling Communications for a Class of Parallel Space-Time Adaptive Processing Algorithms*. Technical Report, University of Oklahoma, 2000.

[JW00b] West, Jack M. and John K. Antonio. "A Genetic Algorithm Approach to Scheduling Communications for a Class of Parallel Space-Time Adaptive Processing Algorithms." *IPDPS Workshops*. 855–861. 2000.

[DR00] Woodward, P. M. "Detection and Ranging: Radar in the Twentieth Century," *IEEE Aerospace and Electronic System Magazine*, *15*(10):27–46 (October 2000).

[HSW01] Works, How Stuff, "How Radar Works."

[XM00] Xiaoyan, Ma, et al. "An Approach of Radar Clutter Recognition Based on High Order Statistics Combination." *Proceedings of ICSP2000*. 2000.

[MX00] Xiaoyan, Ma, et al. "An Approach of Radar Clutter Recognition Based on Higher-Order Statistics Combination," *5th International Conference on signal Processing Proceedings* (2000).

[KY00] Yang, K., et al., "Space-time adaptive processing based on unequally spaced antenna arrays," 2000.

[KY99] Yang, Kehu, et al., "Performance Estimation of Space-Time Adaptive Processing via Subband Processing in Mobile Communications."

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
xx-03-2002	Master's Thesis	Mar 2001 - Mar 2002

4. TITLE AND SUBTITLE

SPACE TIME ADAPTIVE PROCESSING AND CLUTTER CLASSIFICATION INTEGRATION AND EVALUATION

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Jensen, Nathan A., 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFT/EN)
2950 P. Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/02M-05

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/IFTC
Attn: Zenon Pryk
26 Electronics Parkway
Rome, NY 13441-4514
Comm: (315) 330-2596 DSN: 587-2596
Email: zenon.pryk@rl.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/IFTC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

Dr. Gary B. Lamont, ENG, (937) 255-3636 x4718, Gary.Lamont@afit.edu

14. ABSTRACT

Radar is a fundamental technology in today's military and civilian environment, and continuing development of this technology is of utmost importance to maintaining a technological advantages this realm. Current radar technologies suffer from jamming and clutter limitations. STAP is a statistical method to remove this noise, however it is extremely computationally intensive, and presents several real time processing hurdles.

Clutter Classification is another method to classify the radar returns that are found according to the best fit statistical distribution that the return follows. This research investigation attempts to use this clutter classification technology to aid in the detection of targets by filtering the radar returns and then passing only the target rich data the computationally complex STAP application. This research effort also attempts to optimize the STAP application through this integration to provide real time STAP radar processing power to current platforms with minimal hardware requirements.

15. SUBJECT TERMS

Space Time Adaptive Processing, Clutter Classification, High Performance Computing, Goodness of Fit, Radar Efficiency and Effectiveness

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gary B. Lamont
Unclassified	Unclassified	Unclassified	UL	167	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636 x4718