AFIT/DS/ENG/94-03

MULTIRATE TIME-FREQUENCY DISTRIBUTIONS

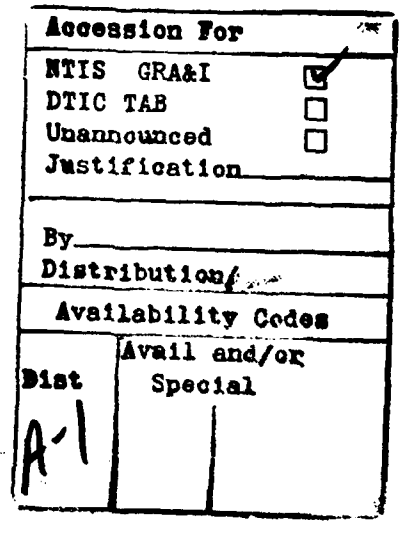
DISSERTATION

John R. O'Hair, B.S., M.B.A., M.S.
Captain, USAF

AFIT/DS/ENG/94-03

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense for the U. S. Government.



AFIT/DS/ENG/94-03

MULTIRATE TIME-FREQUENCY DISTRIBUTIONS

DISSERTATION

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

John R. O'Hair, B.S., M.B.A., M.S.

Captain, USAF

June 1994

AFIT/DS/ENG/94-03

MULTIRATE TIME-FREQUENCY DISTRIBUTIONS

John R. O'Hair, B.S., M.B.A., M.S.
Captain, USAF

Approved:

Bruce W. Suter — 2 May 94

[illegible] — 2 May 94

Mark E. Oxley — 2 May 94

[illegible] — May 2/94

[illegible] — 2 May 1994

Accepted:

[illegible] 13 May 1994

Institute Senior Dean

Acknowledgments

I would like to extend my deepest appreciation to the members of my research advisory committee, Dr. Steve Rogers and Dr. Mark Oxley. I would also like to thank Dr. Matthew Kabrisky for his sage advice throughout my stay at AFIT, and to Dr. Leon Cohen for his guidance and support. My thanks also go to Dr. Martin DeSimio for serving as the Dean's Representative to my advisory committee.

My most heartfelt gratitude goes to Dr. Bruce Suter who had the unenviable task of serving as my advisor. His patience, endurance and willingness to go the extra mile were what got me through the Ph.D. program at AFIT. Without his help, I would never have graduated.

My final thanks go the One for whom all this is done and who makes it all worthwhile. To my Lord and Saviour, Christ Jesus, THANK YOU!

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Terms

Term	Definition
AF-DWD	Alias-Free Discrete Wigner Distribution
AF-GDTFD	Alias-Free Generalized Discrete Time-Frequency Distribution In this dissertation, this is synonomous to GDTFD
CK	Continuous Kernel
DFT	Discrete Fourier Transform
D-GDTFD	Decimated Generalized Discrete Time-Frequency Distribution
DHT	Discrete Hartley Transform
DTFD	Discrete Time-Frequency Distribution
DWD	Discrete Wigner Distribution
FFT	Fast Fourier Transform
GDHT	Generalized Discrete Hartley Transform
GDTFD	Generalized Discrete Time-Frequency Distribution In this dissertation, this is synonomous to AF-GDTFD
GTFD	Generalized Time-Frequency Distribution
MIMD	Multiple Instruction streams Multiple Data streams
MR DHT	Multirate Discrete Hartley Transform
MR FFT	Multirate Fast Fourier Transform
MRTFD	Multirate Time-Frequency Distribution
STFT	Short-Time Fourier Transform
TFA	Time-Frequency Analysis
TFD	Time-Frequency Distribution
WZT	Windowed Zak Transform
ZS	Zak-Spectrogram

Abstract

Multirate systems, which find application in the design and analysis of filter banks, are demonstrated to also be useful as a computational paradigm. It is shown that any problem which can be expressed as a set of vector-vector, matrix-vector or matrix-matrix operations can be recast using multirate. This means all of numerical linear algebra can be recast using multirate as the underlying computational paradigm. By viewing multirate as a computational paradigm, many problems found in signal processing can also be reformulated into fast parallel algorithms. For example, this paradigm is applied in a straight forward fashion to the Fast Fourier Transform (FFT) and the Discrete Hartley Transform (DHT) to create fast parallel, or multirate, versions of these algorithms.

As a non-trivial example, the multirate computational paradigm is applied to the problem of Generalized Discrete Time-Frequency Distributions (GDTFD) to create a new family of fast algorithms for the calculation of Time-Frequency Distributions (TFD). The first result of the application of multirate as a computational paradigm to GDTFD's is a new class of distributions called the Decimated GDTFD (D-GDTFD). These distributions, which are based upon the Zak transform, trade bandwidth for speed. For a decimation factor of m, there is an m fold increase in throughput (or speed of calculation). The corresponding reduction in discrete bandwidth is from 2π for the GDTFD to $2\pi/m$ for the D-GDTFD. An important attribute of the D-GDTFD is that it requires significantly less storage than the GDTFD. The D-GDTFD requires only $1/m^2$ of the storage of the GDTFD.

By combining several D-GDTFD's, it is possible to reconstruct a GDTFD. This reconstruction of D-GDTFD's is the Multirate Time-Frequency Distribution (MRTFD). Each D-GDTFD is independent, and as a result, the MRTFD can easily be implemented in parallel for an increase in throughput on the order of m. If additional parallel paths are

available, the individual D-GDTFD's can also be implemented in parallel leading to improvements in throughput on the order of m^2 or more.

Two distinct MRTFD algorithms are presented. The first MRTFD is based upon the inner product form of the GDTFD and combines the Zak transform, weighted spectrograms and Singular Value Decomposition (SVD). It is called the SVD MRTFD and calculates the distribution for particular instants of time. The second MRTFD is based upon the outer product form of the GDTFD and is called the Circular Convolution MRTFD. It also builds upon the Zak transform and calculates the distribution for blocks of time instead of isolated instants.

Three kernel design techniques are also developed to allow the movement of alias-free kernels between the ambiguity and time-lag domains. This allows the use of any kernel defined in either domain in continuous or discrete form in a D-GDTFD or MRTFD. These techniques can also be used with the GDTFD.

Multirate Time-Frequency Distributions

1. Introduction

An important part of understanding a nonstationary signal is being able to characterize its changing frequency content as a function of time. Important examples of this are found in areas such as speech recognition and analysis, machine fault detection, sonar, radar, spread spectrum and low probability of intercept communications, etc. The field of Time-Frequency Analysis (TFA) is dedicated to the task of determining this aspect of signal behavior. This dissertation introduces several new TFA tools.

1.1. Historical Background

Time-Frequency could be said to have started in 1946 when Koenig, Dunn and Lacy published "The Sound Spectrograph" which presented a method to examine changing spectral content of speech as a function of time [31]. In 1948, Ville [53] showed that joint Time-Frequency Distributions (TFD) in signal processing could be created based upon a bilinear form of a one dimensional signal. It turns out that this result is mathematically equivalent to Wigner's distribution which was originally developed as Position-Momentum representation in quantum mechanics [55]. Many efforts were subsequently made to create TFA tools. Each tool had its advantages and disadvantages, and they were each thought to be unique and unrelated. Then, in 1966, Leon Cohen unified the field of TFA by introducing the Generalized Time-Frequency Distribution (GTFD) [17]. He showed that all the previous TFA tools (or distributions as they are called) were in fact members of the same family of equations related by a variable kernel function.

Unfortunately, the GTFD generates distortions between the "correct" terms. Their removal while maintaining the other desired properties of the TFD is largely dependent upon the correct selection of the kernel function. Since the kernel function profoundly

influences the behavior of the cross-terms in TFD's and defines the other characteristics of the TFD, proper selection of the kernel is paramount. As such, kernel design has been the main focus of Time-Frequency Distribution research. It can be argued that the purpose of this sub-area of research is to improve the accuracy of Time-Frequency Distributions.

Another important sub-area of research is to improve the performance of TFD's. Using traditional techniques, solution of the Generalized Discrete Time-Frequency Distribution (GDTFD) is an $O(N^2 \log N)$ process which can make it impractical for many real time applications [9]. Except for one recent paper by Cunningham and Williams [19], fast algorithms to calculate the GDTFD have been largely overlooked. The goal of this research is to rectify this shortcoming by developing fast algorithms to calculate the GDTFD.

1.2. Problem Statement and Scope

The problem answered in this dissertation is: Is there a systematic means to determine a fast method (or methods) for calculating the GDTFD? If so, what is it, and how can it be used to implement a fast GDTFD? The ideas found in multirate provide a basis for the approach to solve this problem. It furnishes a powerful divide and conquer formalism which can incorporate the Time-Frequency Distribution problem as a particular application. Multirate provides the scope for this problem and is, in fact, the systematic means that is sought.

1.3. General Approach

The approach to fast algorithms taken in this dissertation is to use multirate as an underlying computational paradigm. This is a new computational paradigm. Instead of using multirate as a tool to design and implement filter banks only, multirate is used as a powerful tool to design and implement, in parallel, divide and conquer algorithms. While the focus of this dissertation is to develop new TFA tools, it is shown that this computa-

tional paradigm can be applied to any problem which is composed of a set of vector-vector, matrix-vector or matrix-matrix operations. This effectively encompasses all of numerical linear algebra and much of signal processing.

The first step in developing a fast GDTFD algorithm is to demonstrate that the new paradigm is useful for commonly used, rather straightforward, signal processing algorithms—the Fast Fourier Transform (FFT) and the Discrete Hartly Transform (DHT). The remainder of the dissertation applies this new computational paradigm to a non-trivial application—the calculation of Generalized Discrete Time-Frequency Distributions.

The successful development of the multirate FFT and DHT suggest that multirate can be extended to other, more complicated, algorithms such as the GDTFD, but, before developing a Multirate Time-Frequency Distribution (MRTFD), it is first necessary to extend the state-of-the-art in kernel design techniques. Three new methods to move kernels between the ambiguity and time-lag domains are developed. This allows the easy calculation or modification of kernels for use with the MRTFD regardless of the domain in which the kernel was designed.

Utilizing the new kernel design techniques, together with the Zak transform and the idea of weighted spectrograms, permits the creation of a new class of distributions called the Decimated Generalized Discrete Time-Frequency Distribution (D-GDTFD). The D-GDTFD forms the basic building block for the MRTFD. Each of these building blocks requires less storage and can be implemented much faster than the GDTFD. By combining D-GDTFD's, it is possible to create a MRTFD which is exactly equivalent to the GDTFD but can be calculated in parallel and in significantly less time.

1.4. Organization

In Chapter 2, background information on Time-Frequency Distributions, multirate and other necessary subject areas is given. The new computational paradigm is introduced in Chapter 3 along with the multirate vector-vector, matrix-vector and matrix-matrix oper-

ations as well as the multirate FFT and DHT. In Chapter 4, the new tools for kernel design are presented, and the Zak transform and its connection to the D-GDTFD is developed in Chapter 5. With these tools, the Multirate Time-Frequency Distribution is developed and presented in Chapter 6. Finally, in Chapter 7, the summary and recommendations are given.

2. Background

2.1. Introduction

An overview consisting of definitions and properties of Time-Frequency Distributions, multirate, parallel algorithms, tensor notation and the Zak transform is presented. This material forms a foundation for the following chapters and establishes conventions with regard to definitions and notation that will be followed throughout the dissertation.

2.2. Some Important Domains

In this dissertation, reference is continually made to three different domains or planes. These are the time-lag (or t-τ) domain, ambiguity (or θ–τ) domain and time-frequency (or t-ω) domain. The time-lag plane is the domain of the bilinear form of the signal, $R_f(t,\tau)$, defined as

$$R_f(t,\tau) = f\left(t+\frac{\tau}{2}\right)f^*\left(t-\frac{\tau}{2}\right) \tag{2.1}$$

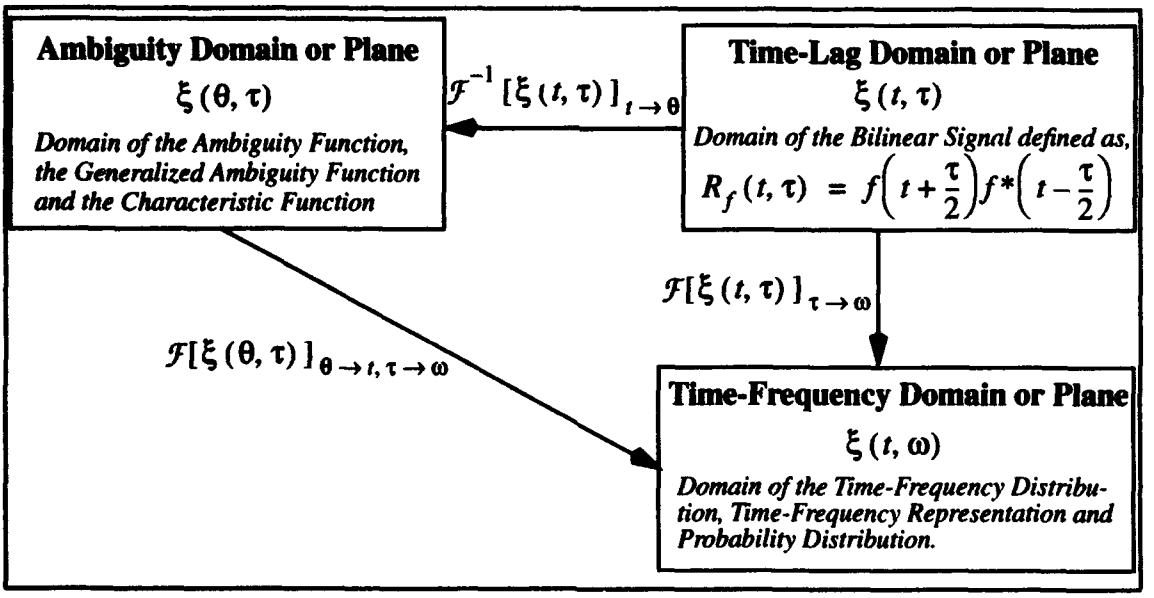


Figure 2.1. The Domains of the AF-GDTFD and Their Relationships One to Another.

where f is a one dimensional signal. It is related to the ambiguity function by the inverse Fourier transform of $R_f(t,\tau)$ with respect to t. The ambiguity plane is also the domain of the generalized ambiguity function (the product of the bilinear signal, $R_f(t,\tau)$, and the ambiguity plane kernel, $\phi(t,\tau)$). In the case where the end result is a probability density function, the generalized ambiguity function is also called the characteristic function. The ambiguity plane is related to the time-frequency plane via a two dimensional Fourier transform. (See Figure 2.1.)

2.3. Generalized Time-Frequency Distributions

2.3.1. Outer Product Formulation. The Generalized Time-Frequency Distribution (GTFD) is given by Cohen [16]

$$C_f(t,\omega;\phi) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f\left(u+\frac{\tau}{2}\right) f^*\left(u-\frac{\tau}{2}\right)\psi(t-u,\tau)\, e^{-j\tau\omega}\, du\, d\tau \tag{2.2}$$

where $\psi(t,\tau)$ is the kernel in the time-lag domain. It is related to the kernel in the ambiguity plane, $\phi(\theta,\tau)$, by

$$\psi(t,\tau) = \int_{-\infty}^{\infty} \phi(\theta,\tau)\, e^{-j\theta t}\, d\theta . \tag{2.3}$$

A common discrete form of equation (2.2), called the Discrete Time-Frequency Distribution (DTFD) is given by

$$C(n,k;\phi) = \sum_{\tau=-\frac{N}{2}}^{\frac{N}{2}-1}\left(\sum_{u=-N}^{N-1} f(u+\tau)\, f^*(u-\tau)\,\psi(n-u,\tau)\right) e^{-\frac{j2\pi k\tau}{N}} \tag{2.4}$$

where N is the number of samples and for ease of computation is chosen to be a power of two.

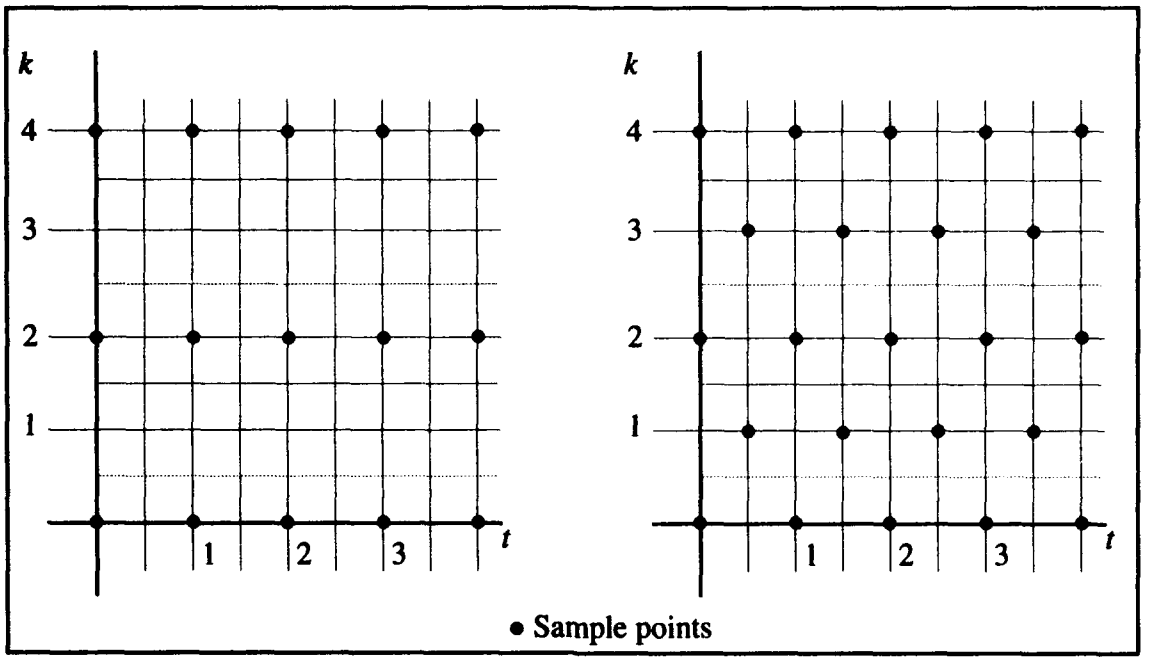


Figure 2.2a. $R_f(t,k)$ – Rectangular Grid.

Figure 2.2b. $R_f(t,k)$ – Hexagonally Decimated Rectangular Grid.

For a discrete signal, f, the values of $f(t)$, $t \in Z$, are known, but the formulation $f(u+\tau)\,f^*(u-\tau)$, $u, \tau \in Z$, includes only half of the possible bilinear data points. To illustrate, define a function $R_f(t,k)$ such that

$$R_f(t,k) = f\left(t+\frac{k}{2}\right)f^*\left(t-\frac{k}{2}\right) \tag{2.5}$$

where $t, k \in Z$. Now examine R_f on the t-k plane for the case of t and k restricted to integers values. The support of the function is on a rectangular sampling grid, but it does not have any non-zero values for k odd. Thus, half of the information in the k-direction is lost. Figure 2.2a demonstrates this property. This inadvertent decimation in the k-direction means that the GDTFD has only half of the bandwidth theoretically possible for a given sampling rate. It is, therefore, considered to be periodic in π rather than 2π.

In [29], Jeong and Williams note data is available whenever the argument of f, i.e. $t \pm k/2$, is an integer. This suggests that data is present when k is even and t is an integer and when k is odd and t is an integer plus one-half. As Jeong and Williams point out,

$R_f(t,k)$ can be thought of as lying on the grid shown in Figure 2.2b. This is a function which has been sampled at intervals of 1/2 in t and k and then hexagonally decimated by a factor of four (see, for example, Vaidyanathan [50], p573 and p648).

By taking $R_f(t,k)$ to be on the hexagonally decimated rectangular sampling grid (or, simply, hexagonal grid) and using this instead of the rectangular grid, all terms of the discrete signal, f, are being used, making the new distribution periodic in 2π. This is the basis of the Jeong and Williams method which they call the Alias-Free Generalized Discrete Time-Frequency Distribution (AF-GDTFD). This will be the outer product form of the GDTFD which will be used throughout this dissertation, and GDTFD will be synonymous to AF-GDTFD.

2.3.2. Inner Product Formulation. The inner product formulation can be derived from the outer product by means of a double change of variables. Starting with (2.2), let $u = t_1 - t - \tau/2$, then

$$C_f(t,\omega;\psi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t+t_1)\, f^*(t+t_1-\tau)\, \psi\left(-t_1+\frac{\tau}{2},\tau\right) e^{-j2\pi\tau\omega}\, dt_1\, d\tau. \qquad (2.6)$$

Next, interchanging orders of integration and substituting $t_2 = t_1 - \tau$ results in the inner product form, [2][3][18][26]

$$C_f(t,\omega;\psi) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t+t_1)\, e^{-j2\pi\omega(t+t_1)}\, \psi\left(\frac{-t_1-t_2}{2}, t_1-t_2\right)$$

$$\times \left[f(t+t_2)\, e^{-j2\pi\omega(t+t_2)}\right]^* dt_1\, dt_2$$

$$= \langle \tilde{\psi} S_{-t} M_{-\omega} f, S_{-t} M_{-\omega} f \rangle \qquad (2.7)$$

where

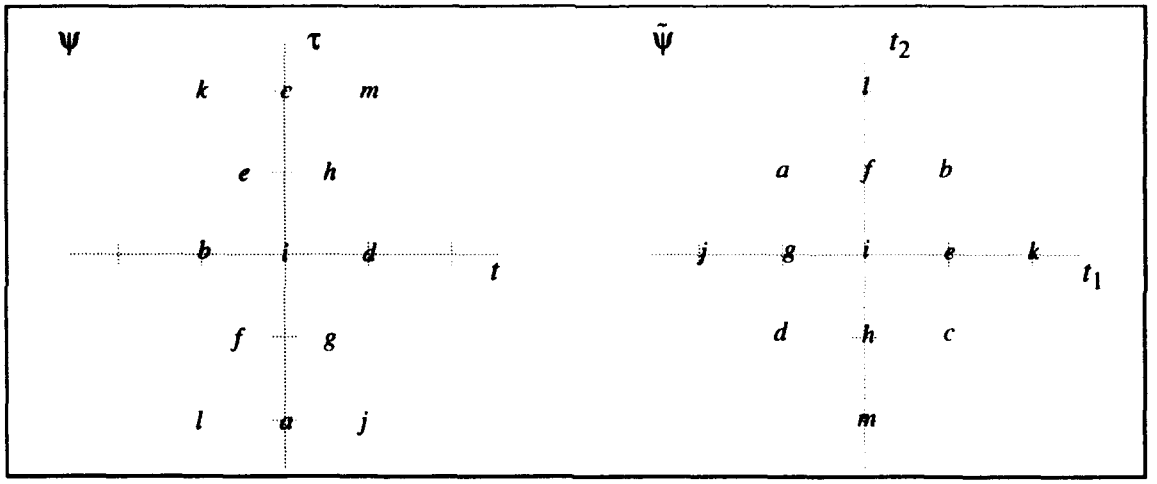


Figure 2.3. Mapping of Points in ψ to $\tilde{\psi}$.

$$\begin{aligned} &\text{Time Shift (by } t\text{) Operator} && (S_t f)(\tau) = f(\tau - t) \\ &\text{Frequency Shift (by } \omega\text{) Operator} && (M_\omega f)(\tau) = f(\tau)\, e^{j\omega\tau} \end{aligned} \tag{2.8}$$

and

$$\tilde{\psi}(t_1, t_2) = \psi\left(\frac{-t_1 - t_2}{2}, t_1 - t_2\right). \tag{2.9}$$

One advantage of the inner product form over the outer product form is seen in the discrete case. The outer product form requires the kernel to be sampled on hexagonally decimated sampling grid to avoid aliasing [29]. This can cause some difficulties when designing kernels as will be discussed in Chapter 4; however, the inner product form is naturally alias-free (i.e. it is 2π periodic rather than π periodic). The sample points in $\tilde{\psi}(t_1, t_2)$, where t_1 and t_2 are integers, correspond to the points in ψ when it has been sampled on the hexagonally decimated grid. Pictorially, this can be seen in Figure 2.3.

A GDTFD can be generated by calculating the sum of weighted spectrograms [18]. The weighting factors are simply the eigenvalues of the kernel $\tilde{\psi}$, and the window functions in the spectrograms are the eigenvectors corresponding to the eigenvalues. In other words, (2.7) can be rewritten as

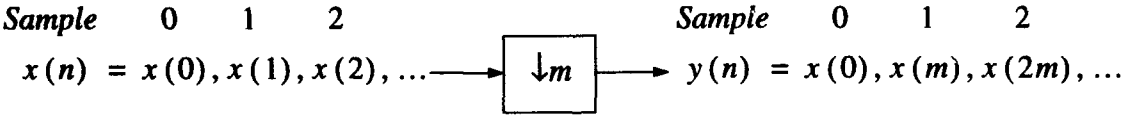
$$C_f(t,\omega;\psi) = \sum_{k=1}^{N} \lambda_k \left| \int_{-\infty}^{\infty} f(t+t_1)\, x_k^*(t_1)\, e^{-j2\pi\omega(t+t_1)}\, dt_1 \right|^2 \qquad (2.10)$$

where λ_k is the k^{th} non-zero eigenvalue and x_k is the k^{th} eigenvector of $\tilde{\psi}$. With this formulation, any GDTFD which has a Hermitian kernel (and hence a complete set of eigenvectors) can be written as the sum of weighted spectrograms.

Only those properties of Time-Frequency Distributions needed for this work are included here. Further details are given in Cohen [16] and Boashash [8].

2.4. Multirate Background

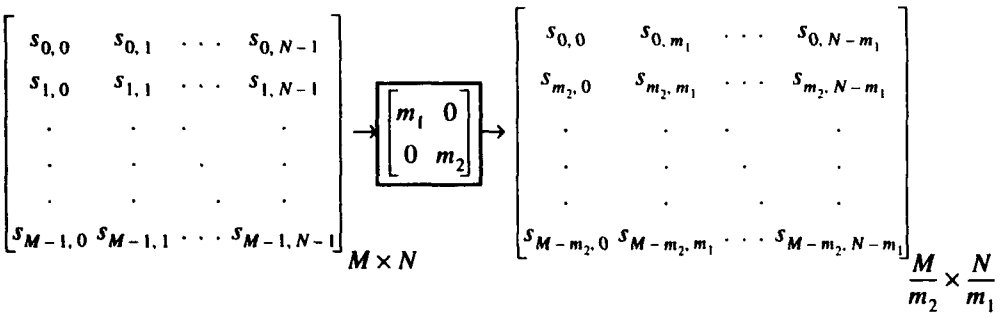
The first multirate building block that is needed is the decimator. The m fold decimator takes an input sequence and shortens it by keeping only every m^{th} sample. A decimator can also be used on multidimensional signals. In this dissertation, both one and two dimensional signals are encountered. The one dimensional decimator is represented by a block containing the down arrow symbol ($\downarrow$) and the decimation factor, i.e.



The two dimensional decimator is a two by two matrix. The matrix need not be diagonal in general; however, in this work the matrix is always a diagonal integer matrix which produces a rectangular decimation scheme (i.e. the decimation in one dimension is independent of that in the other). The decimation matrix,

$$\mu = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}, \qquad (2.11)$$

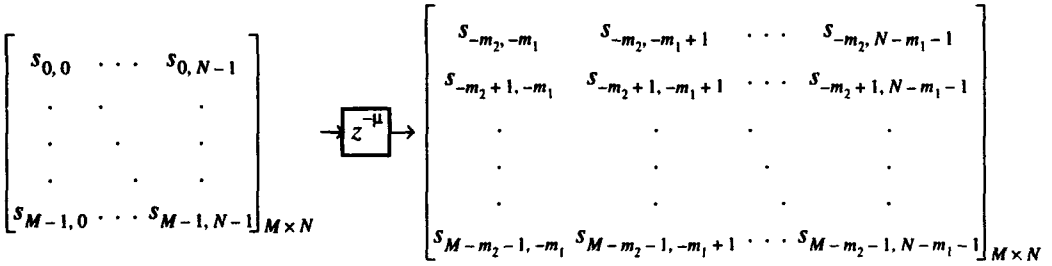
would have the effect of decimating a two dimensional signal by m_1 in the horizontal (or x) direction and m_2 in the vertical (or y) direction, i.e.



The final multirate element to be defined is the delay. The delay can also operate on multidimensional signals. In this dissertation, both the one and two dimensional delays will be used. The one dimensional delay by m would delay a signal by m samples. The symbol for a delay is z^{-m}. In block diagrams, it is placed above an arrow indicating a delay between the blocks being connected by the arrow, i.e.

$$x(n) = \ldots, x(m), x(m+1), x(m+2), \ldots \xrightarrow{z^{-m}} y(n) = \ldots, x(0), x(1), x(2), \ldots$$

The two dimensional delay is represented by $z^{-\mu}$, where μ is defined by (2.11). It has the effect of delaying the signal in the horizontal direction by m_1 and in the vertical direction by m_2, i.e.



Additional information on digital filter design using multirate signal processing, may be found in Chen and Vaidyanathan [14] and Vaidyanathan [50]. Multirate as a computational paradigm is discussed in Chapter 3.

2.5. Parallel Algorithm Background

For the purposes of this dissertation, computer architecture is restricted to Multiple Instruction streams Multiple Data streams (MIMD) shared memory computers, such as the Cray YMP. (For more information on this type of architecture, see, for example, Hwang and Briggs [27].) Thus, the phrase "parallel algorithm" will be used synonymously with the phrase "parallel algorithms on MIMD shared memory computers."

An important concept in parallel algorithms is synchronization of the parallel paths. Synchronization is a place in the algorithm when the results from different processing paths must be recombined before the algorithm can proceed. A place in the algorithm where synchronization is necessary is called a barrier and the computations done between barriers is called a stage. (See, for example, [51].)

Suppose there is an algorithm which has three stages. It would have four barriers: one at the start, between each stage and one at the end. The number of computations necessary in any given path to go from barrier to barrier is not necessarily a constant amount between processor paths; thus, some processors will finish before the others in a given stage and be idle while the others complete their task. The greater the difference between the time it takes to compute the paths in a given stage (i.e. the computational cost of the pathways), the less efficient the algorithm.

For example, consider the algorithm depicted in Figure 2.4. The length of the horizontal arrows indicate the number of computations (or amount of time) it takes to process the job given to a particular processor. Stage one has a good balance of the load between the processors, and as such, it is highly efficient. Stage two, on the other hand, is poorly balanced and, as a result, is inefficient. Stage two also shows a sub-stage. This is caused by a synchronization which is necessary between some fraction of the total processors. If the fraction is small, this would create a sub-stage rather than a stage since most of the proces-

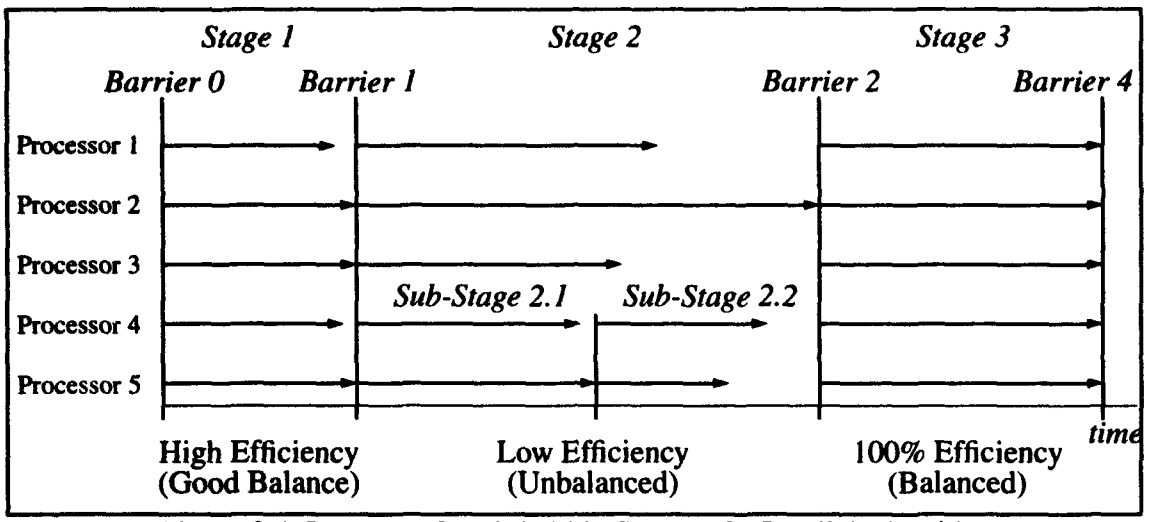


Figure 2.4. Processor Loads Within Stages of a Parallel Algorithm.

sors are unaffected. The final stage is perfectly balanced and is, therefore, 100 percent efficient.

For additional information on parallel algorithms and associated computer architecture, see, for example, Van Loan [51], Hwang and Briggs [27] or Akl [1].

2.6. Tensor Notation Background

The tensor product (also known as the Kronecker product) of two matrices, A and B, is represented by the symbol $A \otimes B$. There are two possible definition of the tensor product: the left tensor product and the right tensor product. Since both are equally valid, the right tensor product has been chosen. Suppose A is $n \times m$ and B is $i \times j$, then the right tensor product is defined as [25] [43]

$$A_{n,m} \otimes B_{i,j} = \begin{bmatrix} a_{0,0}B & a_{0,1}B & \cdots & a_{0,m-1}B \\ a_{1,0}B & a_{1,1}B & \cdots & a_{1,m-1}B \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{n-1,0}B & a_{n-1,1}B & \cdots & a_{n-1,m-1}B \end{bmatrix}_{ni \times mj} \tag{2.12}$$

The second form of tensor notation used is the direct sum represented by $A \oplus B$. It creates a block diagonal matrix and defined as [43]

$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \tag{2.13}$$

If A is $n \times m$ and B is $i \times j$, then $A \oplus B$ is $(n+i) \times (m+j)$. For the set of matrices $\{A_i\}$, $i = 0,\ldots, N-1$, the direct sum is

$$\bigoplus_{i=0}^{N-1} A_i = \begin{bmatrix} A_0 & & & 0 \\ & A_1 & & \\ & & \ddots & \\ 0 & & & A_{N-1} \end{bmatrix}. \tag{2.14}$$

Tensor notation also makes use of a particular type of permutation matrix called the stride by m permutation. It is represented by $P_{N,m}$ where N is the length of the sequence being permuted and m is the stride. The effect of the stride permutation is to rearrange a vector, $x = [x(0)\, x(1) \ldots x(N-1)]^T$, such that its permutation, $P_{N,m}x = x'$, is given by

$$\begin{aligned} x'(n) &= \left[x(0)\ x(m)\ \ldots\ x((L-1)m) \mid \ldots \mid x(m-1)\ x(2m-1)\ \ldots\ x(N-1)\right]^T \\ &= \left[x_0(n)\ x_1(n)\ \ldots\ x_i(n)\ \ldots\ x_{L-1}(n)\right]^T \end{aligned} \tag{2.15}$$

where $L = N/m \in \mathbf{Z}$ and

$$x_i(n) = \left[x(i)\ x(m+i)\ \ldots\ x(mn+i)\ \ldots\ x(m(L-1)+i)\right], \tag{2.16}$$

$n = 0, 1, \ldots, L-1$. This implies

$$x'(k) = x\left(m(mk \text{ modulo } N) + \left\lfloor \frac{mk}{N} \right\rfloor\right) \tag{2.17}$$

where $k = 0, 1, \ldots, N-1$, and $\lfloor\ \rfloor$ indicates the integer portion of the quantity.

It is important to note that the stride permutation is a *mathematical* operation which is often implemented as an *addressing* operation [25]. This means that for many hardware configurations it can be implemented at almost no cost; however, the mathematical nature of the operation must never be forgotten.

Additional theorems and definitions for tensor products may be found in many sources, including Granata, et al, [25], Van Loan [51] and Regalia and Mitra [43].

2.7. Zak Transform

The Zak transform, like a TFD, transforms a one-dimensional signal into a two-dimensional representation. It comes in several forms [6][28], but only the finite discrete Zak transform is of interest in this work. Suppose there is a continuous one-dimensional signal, f_c, which is sampled at intervals of ρ to produce a discrete signal, f, then the finite discrete signal is given by $f(u\rho)$, where $u \in \{0, 1, \ldots, N-1\}$, $N, m, L \in Z^+$ and $N = Lm$. The two-dimensional discrete Zak transform of the finite discrete one-dimensional signal, f, is given by

$$Z_f(n,k) = \sum_{l=0}^{L-1} f([ml+n]\rho)\, e^{-j2\pi lk/L} \tag{2.18}$$

where $n = 0, 1, \ldots, m-1$ and $k = 0, 1, \ldots, L-1$.

Another way to look at (2.18) is to consider laying out the samples of f in a two-dimensional array indexed by the integers n and l. Call this two-dimensional function $z_f(n,l)$. Figure 2.5 shows this mapping. Then, take the discrete Fourier transform of each column of $z_f(n,l)$. The result is the discrete Zak transform, $Z_f(n,k)$.

Additional information regarding Zak transforms may be found in Zak [57], Auslander, Gertner and Tolimieri [6] and Jansen [28].

$$f(u\rho) \rightarrow \begin{bmatrix} f(0) & f(1) & \cdots & f[(m-1)\rho] \\ f(m\rho) & f[(m+1)\rho] & \cdots & f[(2m-1)\rho] \\ \vdots & \vdots & \vdots & \vdots \\ f[(L-1)m\rho] & f\{[(L-1)m+1]\rho\} & \cdots & f[(N-1)\rho] \end{bmatrix}_{L \times m} = z_f(n, l)$$

Figure 2.5. Mapping of f into Two-Dimensional Array

2.8. Conclusion

The preceding contains the fundamental background and definitions used throughout this dissertation. It is important to keep in mind the relationship of the three domains (time-lag, ambiguity and time-frequency) and to understand that the function of the GTFD is to project a bilinear signal defined in the time-lag domain onto the time-frequency domain. It is sometimes fastest computationally to do this by first projecting the bilinear signal from the time-lag domain into the ambiguity plane and subsequently projecting it into the time-frequency plane. This concept is the basis of TFD's.

Multirate is a means of sub-dividing a problem into smaller, more manageable, pieces. The basic building blocks for the sub-division process are the decimator and the delay. These are the two main multirate tools used throughout this dissertation, and they will be applied to the problem of creating a Multirate Time-Frequency Distribution.

With this foundation laid, a new computational paradigm will be introduced in the next chapter. Specifically, multirate will be examined as a computational paradigm for the solution of numerical linear algebraic problems and for two special signal processing cases: the Fast Fourier Transform and the Discrete Hartley Transform.

3. Multirate: A New Computational Paradigm

3.1. Introduction

A new multirate computational paradigm is presented along with a natural way to express numerical linear algebra and signal processing algorithms. This is the first step toward the goal of creating a Multirate Time-Frequency Distribution. The paradigm introduced and the techniques which result from that paradigm form the basic building blocks of the MRTFD.

3.2. Paradigm

The field of multirate signal processing, while rich in its potential, has remained limited in the types of problems it has been used to solve. From the start, it has been used as a means to design and implement filter banks for signal processing systems. Multirate has shown itself to be a powerful design tool, (see, for example, [50]) but it has the potential to be used to solve a much larger class of problems.

Because multirate breaks the problem into smaller independent sub-problems which require fewer computations to solve, a multirate system provides three benefits. First, for a given throughput, the individual components can operate at significantly reduced speed when compared to a non-multirate systems performing the same task. This means the individual components can be much less expensive to produce. Second, by reducing the required clock rate of the individual components the heat dissipation and power consumption budget per device (and, occasionally, for the entire multirate system) will be reduced. Finally, if throughput is the governing factor, multirate can be used to design systems which perform faster than the fastest possible sequential system. This improvement in speed can be several orders of magnitude over a non-multirate sequential system. The MRTFD algorithms presented in Chapter 6 are examples of this improvement in speed.

There are two fundamental underlying assumptions made in the vast majority of the multirate literature: the process to which multirate is applied, and the type of data on which the process will act. Multirate has been almost exclusively viewed as a tool for designing digital filter banks, usually for the purpose of encoding or compressing a signal. The data has been assumed to be an infinite length sequence. These assumptions have limited the possible applications for which multirate is appropriate and profitable. There has been a limited amount of filter bank design which has been done with finite length (or blocked) input data (See, for example, Chapter 10 in [50] or [45]). The utilization of blocked input data provides a starting point for the new multirate applications discussed in this chapter.

Underlying multirate is a larger class of problem solving techniques known as divide and conquer algorithms. Divide and conquer is a broad field of algorithmic techniques used to break problems into smaller independent sub-problems, solve those sub-problems independently and recombine their results to gain the solution to the entire problem. (See, for example, [12].)

It is clear that multirate partitions a given problem into a set of sub-problems and is, therefore, a divide and conquer algorithm. Can it be applied to a broader class of problems? The answer is a resounding, yes. In this chapter, multirate will be shown to be a powerful tool useful as a means of rapidly solving problems encountered in numerical linear algebra and digital signal processing. In section 3.3, the utility of multirate when applied to the basic building blocks of numerical linear algebra will be discussed. In section 3.4, some examples of numerical linear algebraic building blocks will be applied to signal processing applications. By making use of multirate in this unconventional sense, significant improvement in performance is possible for traditional signal processing tasks. This is demonstrated later in this chapter and in Chapter 5 and Chapter 6.

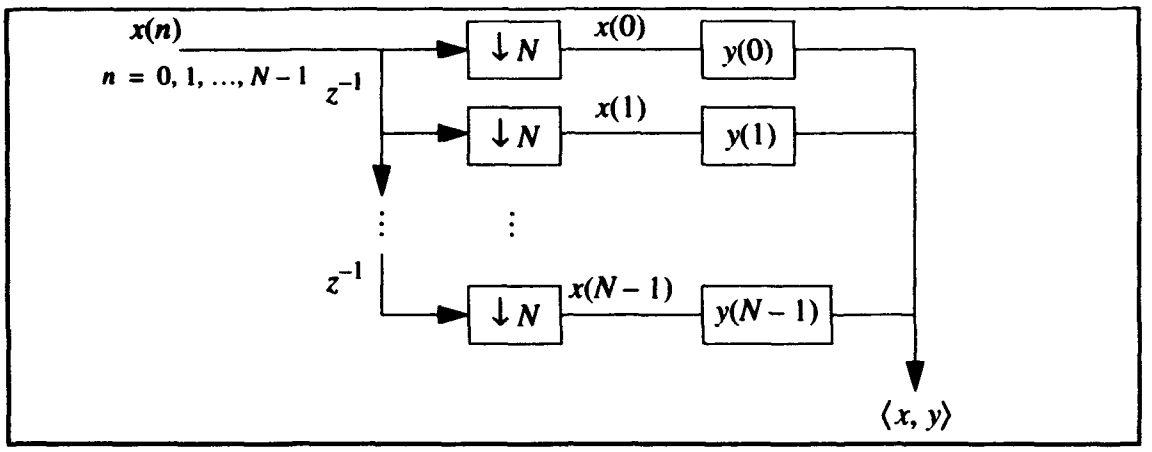


Figure 3.1. Single Stage Multirate Summation.

3.3. Multirate as a Paradigm for Numerical Linear Algebra

Traditional multirate uses finite filters to process a non-finite length sequence. What if both filter and sequence were considered as blocks? This is called block filtering. (See, for example, Chapter 10 in [50] or [45].) Used on block data, it has an inherent benefit of increasing the parallel nature of the filter bank. Moreover, block multirate will be shown to be a powerful tool to implement divide and conquer algorithms on numerical linear algebraic problems. In this section, it will be shown that multirate ideas can be applied to perform any vector-vector, matrix-vector or matrix-matrix operation. It is important to realize that state-of-the-art numerical linear algebraic software, such as LAPACK [4], is built upon a set of core routines called Basic Linear Algebra Subroutines (BLAS) [20][21] which are frequently utilized combinations of vector-vector, matrix-vector and matrix-matrix operations. Thus, it will be shown that multirate provides an important new paradigm for solving problems in numerical linear algebra.

3.3.1. The Multirate Inner Product: A Vector-Vector Operation. The standard discrete inner product is given by

$$\langle x, y \rangle = \sum_{n=0}^{N-1} x(n)\, y(n) \,. \tag{3.1}$$

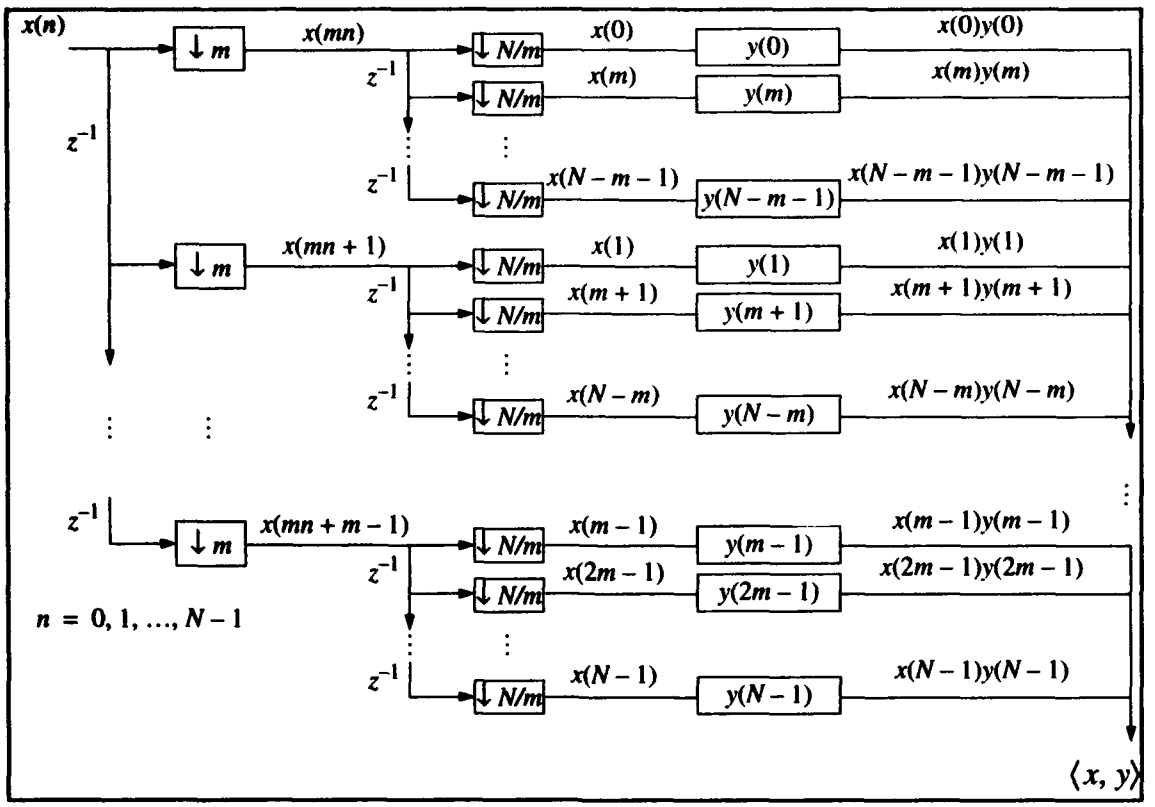


Figure 3.2. Two Stage Multirate Summation with Analysis Filters.

Suppose the sequences x is passed through a delay chain and decimated by N. The analysis filters have an impulse response given by a single coefficient corresponding to a particular element in the sequence y. Figure 3.1 is then a N channel multirate system which sums N elements of the product of the sequences x and y. That is, it calculates the inner product.

Another, equally valid way to decimate x is to decimate in two stages. For example, the first stage would decimate by m and the second stage would decimate by N/m. The analysis filters have an impulse response given by a single coefficient corresponding to a particular element in the sequence y. In other words, Figure 3.2 is a two stage multirate inner product of two sequences. This is expressed mathematically as

$$\langle x, y \rangle = \sum_{i=0}^{m-1} \sum_{n=0}^{\frac{N}{m}-1} x(nm+i)\, y(nm+i) \tag{3.2}$$

Note that (3.2) indicates that both of the sequences have been decimated prior to pointwise multiplication of the sequences. This is considered a m channel multirate system since in practice it would be expected that the computation done after the m fold decimators would be performed in a single channel; hence, there would be m channels.

3.3.2. The Multirate Matrix-Vector Multiply: A Matrix-Vector Operation. Multirate may also be applied to matrix-vector operations. In the case of the vector-vector operation, both sequences were decimated in the same fashion. In the matrix-vector operation both elements will also be decimated. Since the matrix is a two dimensional object, some added complexity has been introduced into the decimation operation. A multirate matrix-vector operation could be developed by decimating the matrix in only one direction, namely decimating in the horizontal direction only; however, this is not the standard way an array is decimated.

Decimation in two dimensions requires a decimation factor for both dimensions. The simplest case is for these to be equal. This is called square decimation and is represented by a decimation matrix

$$\mu = \begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix}. \tag{3.3}$$

The multirate decomposition by this decimation matrix would require one channel for every possible delay in either dimension. Since their are m possible delays in each direction, a total of m^2 channels are required creating m^2 sub-matrices.

Suppose this decimation scheme was applied to A, an $N \times N$ matrix where $N/m = L \in \mathbf{Z}$. If the sub-matrices were put in block matrix form, a new matrix, call it A', would be created. It would have the form

$$A' = \begin{bmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,m-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,m-1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ A_{m-1,0} & A_{m-1,1} & \cdots & A_{m-1,m-1} \end{bmatrix}_{m \times m} \tag{3.4}$$

where the subscripts represent the delay applied to A before it was decimated. The first subscript is the delay in the vertical direction, and the second is the delay in the horizontal direction. The the block matrix $A_{i,j}$ is constructed from the elements of A in the following fashion:

$$A_{i,j} = \begin{bmatrix} a_{i,j} & a_{i,j+m} & \cdots & a_{i,j+(L-1)m} \\ a_{i+m,j} & a_{i+m,j+m} & \cdots & a_{i+m,j+(L-1)m} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{i+(L-1)m,j} & a_{i+(L-1)M,j+m} & \cdots & a_{i+(L-1)m,j+(L-1)m} \end{bmatrix}_{L \times L} \tag{3.5}$$

where the $a_{i,j}$ terms are the elements of A.

It can be seen from the definition for the stride by m operator, (2.17), and from (3.4) and (3.5) that A' is really $A' = P_{N,m} A P_{N,m}$ and the elements of A' are $a_{pm+j,\,nm+i}$. The value i is the delay in the horizontal direction, j is the delay in the vertical direction and p, $n = 0, 1, \ldots, L-1$.

The signal, x, must also be delayed and decimated but, naturally, in only one direction. It must be delayed and decimated exactly as the rows of A are (i.e. in the horizontal direction). Therefore, there are m sub-sequences produced by m channels. If the sub-sequences of x from each channel of the delay and decimation chain are concatenated, the result is the sequence, x', seen in (2.15). In other words, it is $P_{N,m}x$.

The matrix-vector product can be written as the product of A' and x', i.e.

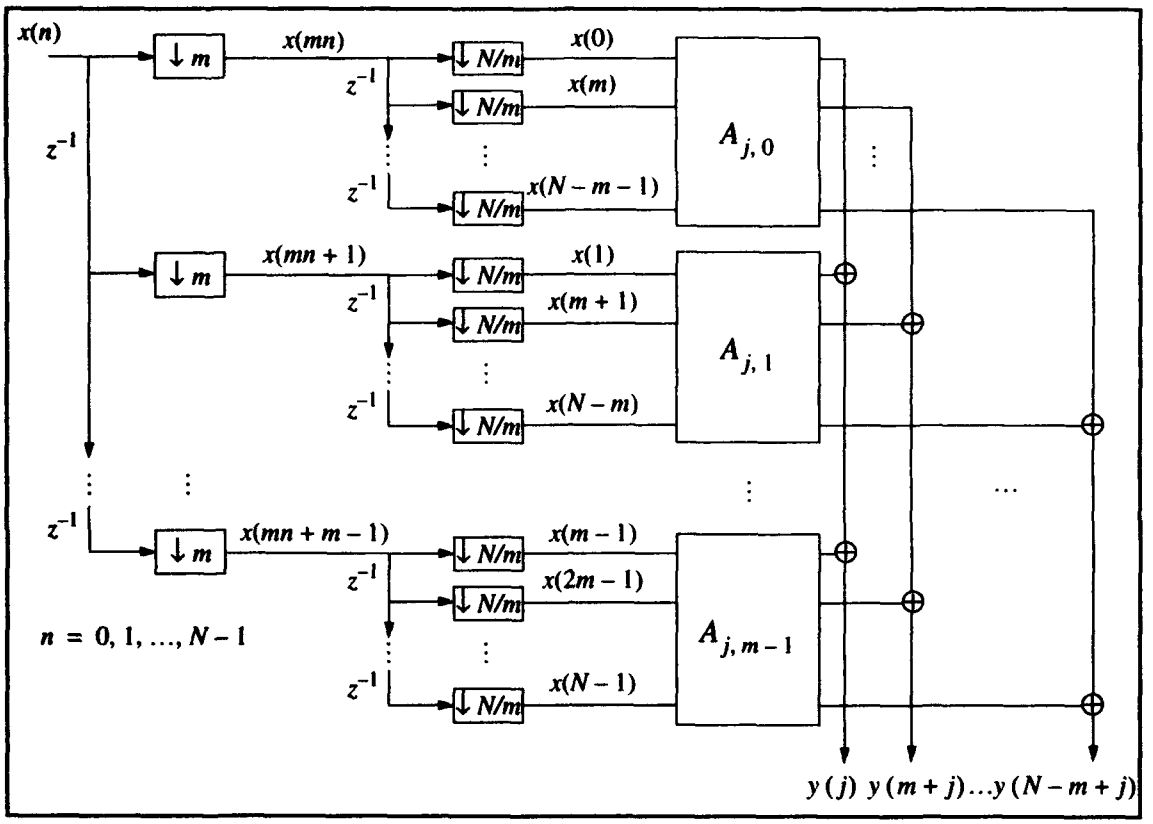


Figure 3.3. Multirate Matrix-Vector Multiplication.

$$y' = A'x' = (P_{N,m} A P_{N,m}) P_{N,m} x. \tag{3.6}$$

Note the result is y' and not y. This is a consequence of the shuffling of the rows done in (3.4). By (2.17), (3.6) could also be written as the double summation

$$y(pm + j) = \sum_{i=0}^{m-1} \sum_{n=0}^{\frac{N}{m}-1} a_{pm+j,\,nm+i}\, x(nm + i) \tag{3.7}$$

where $p = 0, 1, \ldots, N/m - 1$ and $j = 0, 1, \ldots, m - 1$. A block diagram of this for a particular value of j is shown in Figure 3.3. There would be m^2 channels (one for each block of A') for a complete implementation of (3.7).

The decimation matrix, μ, can be generalized such that the values along the diagonal need not be equal. The resulting decimation matrix would have the form

$$\mu = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}. \tag{3.8}$$

Changing μ will change (3.7) and allow the rows and columns of A to be decimated at different rates. Let the $N \times M$ matrix, A, be decimated according to μ, then the new multirate matrix-vector product is

$$y(pm_2 + k) = \sum_{i=0}^{m_1 - 1} \sum_{n=0}^{\frac{M}{m_1} - 1} a_{pm_2 + k,\, nm_1 + i} x(nm_1 + i) \tag{3.9}$$

where $p = 0, 1, \ldots, N/m_2 - 1$, $k = 0, 1, \ldots, m_2 - 1$. The only restriction is that $M/m_1, N/m_2 \in \mathbf{Z}$. This may be useful in certain applications such as under-defined or over-defined systems of equations where $N \neq M$. This is the multirate matrix-vector multiply.

3.3.3. Multirate Matrix-Matrix Multiplication. Matrix-matrix multiplication can be viewed as an extension of the matrix-vector multiplication discussed in the previous section. In the product, AB, the operation repeatedly calculates a matrix-vector multiply for each column of B. This implies that the decimation of B in the vertical direction must be the same as that of A in the horizontal direction. The decimation in the vertical direction of A and the horizontal direction of B do not need to be the same, but these will impact the structure of the result.

Define three decimation factors, m_1, m_2, and m_3. Let m_1 be the decimation factor in the horizontal direction for A and the vertical direction for B. Let m_2 be the decimation factor in the vertical direction for A, and m_3 be the decimation factor in the horizontal direction for B. Then, for a $N \times M$ matrix, A, and a $M \times K$ matrix, B, define A' and B' as

$$A' = \begin{bmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,m_1-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,m_1-1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ A_{m_2-1,0} & A_{m_2-1,1} & \cdots & A_{m_2-1,m_1-1} \end{bmatrix}_{m_2 \times m_1} \tag{3.10}$$

$$B' = \begin{bmatrix} B_{0,0} & B_{0,1} & \cdots & B_{0,m_3-1} \\ B_{1,0} & B_{1,1} & \cdots & B_{1,m_3-1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ B_{m_1-1,0} & B_{m_1-1,1} & \cdots & B_{m_1-1,m_3-1} \end{bmatrix}_{m_1 \times m_3} \tag{3.11}$$

where

$$A_{i,j} = \begin{bmatrix} a_{i,j} & \cdots & a_{i,j+\left(\frac{M}{m_1}-1\right)m_1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ a_{i+\left(\frac{N}{m_2}-1\right)m_2,j} & \cdots & a_{i+\left(\frac{N}{m_2}-1\right)m_2,j+\left(\frac{N}{m_1}-1\right)m_1} \end{bmatrix}_{\frac{N}{m_2} \times \frac{M}{m_1}} \tag{3.12}$$

and

$$B_{i,j} = \begin{bmatrix} b_{i,j} & \cdots & b_{i,j+\left(\frac{K}{m_3}-1\right)m_3} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ b_{i+\left(\frac{M}{m_1}-1\right)m_1,j} & \cdots & b_{i+\left(\frac{M}{m_1}-1\right)m_1,j+\left(\frac{K}{m_3}-1\right)m_3} \end{bmatrix}_{\frac{M}{m_1} \times \frac{K}{m_3}} \tag{3.13}$$

From the definition for the stride given in (2.15) and from (3.10) through (3.13), it can be seen that $A' = P_{N, m_2} A P_{M, m_1}$ and $B' = P_{M, m_1} B P_{K, m_3}$. If the product of $AB = Y$, then the product of A' and B' is

$$Y' = P_{N, m_2} Y P_{K, m_3} = (P_{N, m_2} A P_{M, m_1}) (P_{M, m_1} B P_{K, m_3}) , \tag{3.14}$$

and applying (2.17), it is possible to write the equation for a multirate matrix-matrix multiply as

$$y_{pm_2+q, lm_3+k} = \sum_{i=0}^{m_1-1} \sum_{n=0}^{\frac{M}{m_1}-1} a_{pm_2+q, nm_1+i} b_{nm_1+i, lm_3+k} \tag{3.15}$$

where $k = 0, 1, \ldots, m_3 - 1$, $q = 0, 1, \ldots, m_2 - 1$, $l = 0, 1, \ldots, K/m_3 - 1$, and $p = 0, 1, \ldots, N/m_2 - 1$.

With the addition of the multirate matrix-matrix multiply to the previously defined multirate vector-vector and matrix-vector operations, it can now be said that multirate presents a new paradigm for numerical linear algebraic problems. As a demonstration of the utility of this paradigm, in the next section, the new multirate linear algebraic tools discussed above will be used to construct signal processing examples.

3.4. Multirate as a Paradigm for Signal Processing

In this section, two examples are present to show how multirate can be applied in a new fashion to old problems. The object of these examples is to recast the Fast Fourier Transform and the Discrete Hartley Transform based upon the paradigm discussed earlier. In section 3.4.1, the multirate FFT (MR FFT) is presented, and the multirate DHT (MR DHT) is introduced in section 3.4.2

3.4.1. The Multirate Fast Fourier Transform.

In this section, multirate is used as an alternate means to develop the Four Step FFT reported by Van Loan [51] for MIMD shared memory architectures. The key feature of the multirate FFT is the subdividing of the transform into smaller blocks which can still be solved using FFT's. This section is di-

vided into three parts. First, the theoretical background for the MR FFT is discussed in 3.4.1.1, and the MR FFT algorithm is covered in section 3.4.1.2. Lastly, the cost to compute the MR FFT is compared to the cost of the FFT in section 3.4.1.3.

3.4.1.1. MR FFT Theory. The MR FFT is derived from the discrete Zak transform. In Chapter 5, it will be shown that with slight modification the Zak transform is a generalization of the Short-Time Fourier Transform (STFT). In this section, it is shown that the Zak transform is the basis for a multirate implementation of the STFT which can make use of the computational speed of the FFT.

The definition of the DFT is given by [39]

$$X(r) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nr}{N}} \tag{3.16}$$

If all values of r are considered (i.e. $r = 0, 1, \ldots, N-1$), (3.16) is a $N \times N$ matrix-vector multiply. Using (3.9), (3.16) can be rewritten as

$$X(pm_2 + k) = \sum_{i=0}^{m_1 - 1} \sum_{n=0}^{\frac{N}{m_1} - 1} x(nm_1 + i)\, e^{\frac{-j2\pi (nm_1 + i)\,(pm_2 + k)}{N}} \tag{3.17}$$

where $p = 0, 1, \ldots, N/m_2 - 1$ and $k = 0, 1, \ldots, m_2 - 1$. Let $m = m_1$ and $m_2 = L = N/m$, then (3.17) can be expressed as

$$X(pL + k) = \sum_{i=0}^{m-1} \sum_{n=0}^{L-1} x(nm + i)\, e^{\frac{-j2\pi (nm + i)\,(pL + k)}{N}} \tag{3.18}$$

where now $p = 0, 1, \ldots, m - 1$ and $k = 0, 1, \ldots, L - 1$. With some algebraic manipulation, (3.18) becomes

$$X(pL + k) = \sum_{i=0}^{m-1} \left[e^{\frac{-j2\pi ik}{N}} \sum_{n=0}^{L-1} x(nm + i)\, e^{\frac{-j2\pi nk}{L}} \right] e^{\frac{-j2\pi ip}{m}}. \tag{3.19}$$

The innermost summation is the Zak transform which is then multiplied by a phase shift $e^{-j2\pi ik/N}$. The outer summation is a DFT of the rows of the two dimensional product of the Zak transform and the phase shift.

Equation (3.19) can be expressed in tensor notation as a combination of stride permutations, tensor products and a diagonal matrix multiply. The first step in translating (3.19) to tensor notation is to apply the stride permutation to the input vector, x, i.e. $P_{N,m}x$. Next, the discrete Fourier transform of each L samples of the permuted input must be taken. This is represented by the right tensor product $I_m \otimes W_L$ where W_L is the Fourier transform matrix of dimension L.

The phase shift term in (3.19) is implemented as a diagonal matrix, D. It is the direct sum of the set of diagonal matrices $\{\Omega(l)\}_i$, $i = 0, 1, \ldots, m-1$, defined as

$$\Omega_i(l) = \begin{bmatrix} 1 & & & \\ & e^{-j2\pi i/N} & & \\ & & \ddots & \\ & & & e^{-j2\pi i(l-1)/N} \end{bmatrix}_{l \times l} . \tag{3.20}$$

The resulting diagonal matrix, D, is given by

$$D = \bigoplus_{i=0}^{m-1} \Omega_{i,L} \tag{3.21}$$

Next, a permutation is performed to prepare the vector for the second set of DFT's. The permutation is actually a combination of two strides, $P_{N,m}$ and $P_{N,L}$, performed simultaneously. The first removes the original stride of m, and the second establishes a new stride of L. The last step is to apply the tensor product for the L m point DFT's, i.e. $I_L \otimes W_m$, followed by another stride to restore the output sequence to the original order. The entire tensor product form of (3.19) is

$$X = P_{N,L}(I_L \otimes W_m)\,(P_{N,L}P_{N,m})\,D\,(I_m \otimes W_L)\,P_L x . \tag{3.22}$$

It should be noted that the DFT operations W_m and W_L would be implemented using sequential FFT's. This implies that (3.22) could be further decomposed in terms of tensor products; however, this would confuse the parallel nature of this formulation and is not done here.

3.4.1.2. MR FFT Algorithm. The MR FFT algorithm is a parallel algorithm. The most efficient algorithm in terms of processor loading requires a dependence between the length of the input signal, N, and the decimation factor, m. The MR FFT algorithm will first calculate the Zak transform (which implies the one dimensional vector, x, is mapped into a two dimensional array as is depicted in Figure 2.5) and apply the phase shift in the first stage and then calculate the Fourier transform of the rows in the second stage. The Fourier transforms are calculated using off-the-shelf FFT algorithms. Thus, the MR FFT performs a FFT on the columns of the two dimensional array, z_f, in the first stage, and a FFT of the rows of the two dimensional array in the second. The phase shift could be performed either at the end of the first stage or the beginning of the second. In this implementation, it is performed in the first stage. If each FFT is assigned to an independent processor in each stage, the only way to have the same number of row processors as column processors (keeping the processor load balanced during each stage) is to set $m = \sqrt{N}$. A block diagram of the MR FFT can be seen in Figure 3.4.

Algorithm 3.1 is another way to describe the MIMD shared memory Four Step parallel FFT reported by Van Loan [51]. Stage one contains steps one and two, the FFT and phase shift, as well has half of the third step, transposition of the data. The load at the beginning of stage two completes the transpose, and the fourth step is the calculation of L m-point FFT's.

It is important to note that the multirate paradigm could also be used to efficiently implement the many significant Fourier transform algorithms of Winograd, Tolimieri and Auslander. See, for example, [7]. One last comment on the algorithm before moving on to

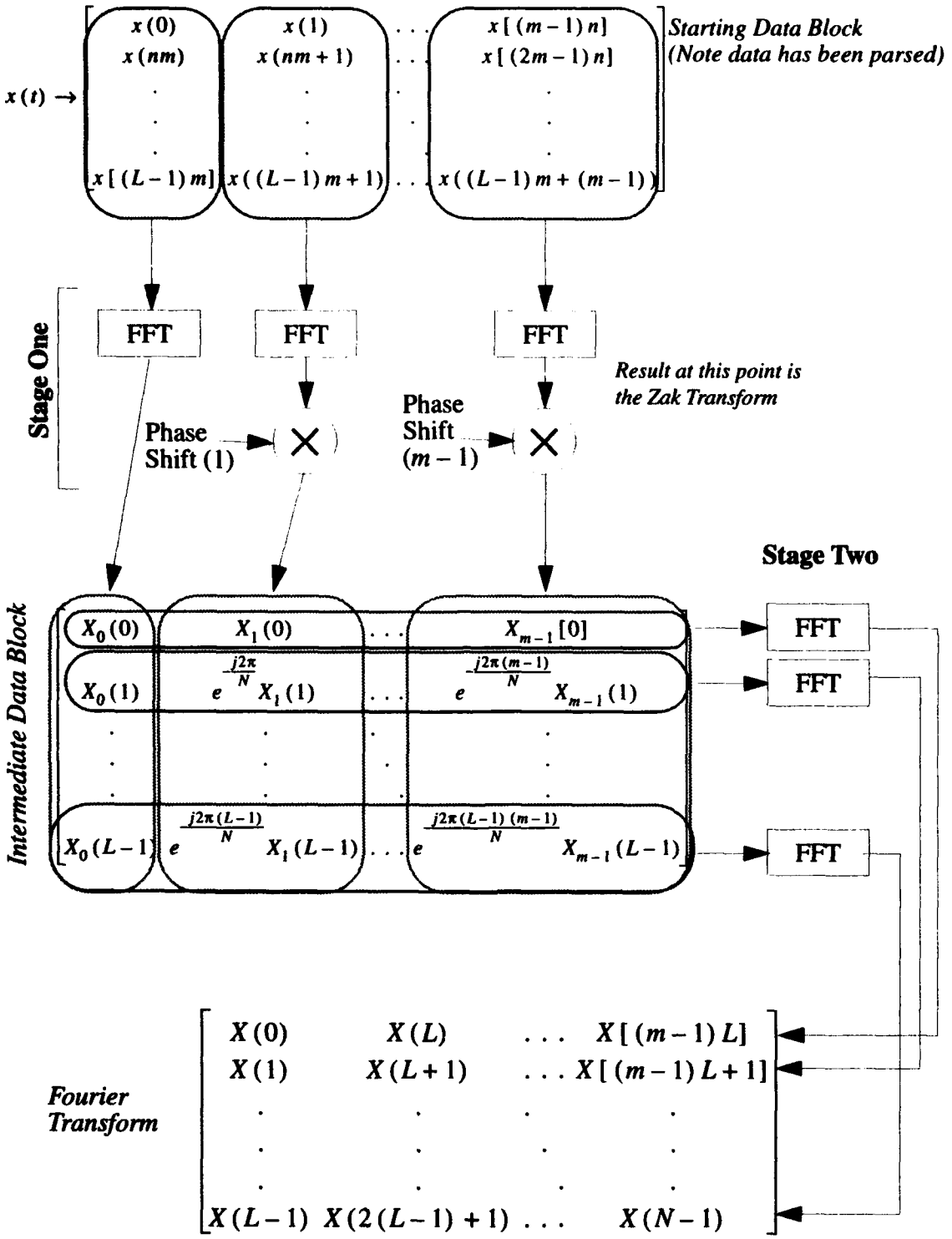


Figure 3.4. Block diagram of the Multirate Fast Fourier Transform.

Algorithm 3.1: Multirate Fast Fourier Transform

	Barrier 0: Start
Stage 1:	Load N data points by parsing L data points into m processors
	Each processor calculates an L point FFT
	Each processor multiplies the L point FFT result by a phase shift
	Output results to Output Array
	Barrier 1:
Stage 2:	Load m data points into L processors from Output Array
	Each processor calculates an m point FFT
	Output results to Output Array
	Barrier 2: End

the DHT. The operations performed on each set of L data points in stage one and set of m data points in stage two are identical in each processor. This implies that the algorithm is suitable for implementation in a SIMD architecture [27].

3.4.1.3. Computational Cost of MR FFT. The MR FFT like the Four Step FFT uses multiple instances of the same sequential FFT algorithm implemented in parallel. The MR and Four Step FFT's require more computations than a sequential FFT would for a given length signal; however, because they are implemented with shorter length sequential FFT's in parallel, the execution time is significantly reduced.

If m is incorrectly chosen, it is very easy to obtain an algorithm which is not very efficient. In considering the cost of this algorithm, it will be assumed that both N and m are correctly selected such that $m = \sqrt{N}$ results in practical split radix-2 FFT's in both stages of the algorithm and that the loading of the processors is optimized.

In Table 3.1, the cost of the algorithm in terms of real multiplications and additions is presented. The numbers are based upon the split radix-2 FFT introduced by Sorensen, Heideman and Burrus in [46]. It requires $N \log_2 N - 3N + 4$ multiplies and $3N \log_2 N - 3N + 4$ additions to calculate an N point DFT via the split radix-2 FFT. This FFT is the basic

building block of the MR FFT. If the type of FFT is changed, then the cost of computing the MR FFT will change also.

Assume the time associated with communications between processors and memory transfers can be neglected and there are as many parallel processors available as needed. Then, the time it takes to calculate the MR FFT and the FFT can be compared based upon the number of computation along the longest path. This is defined to be the processor path which requires the greatest number of computation within a given stage. For a sequential algorithm, this is the total number of computations for the algorithm. For a parallel algorithms, it is the sum of the longest path of each stage in the algorithm.

The longest path for the sequential FFT is the number of real multiplications and additions it takes to calculate an N point DFT. For the MR FFT, the cost of the longest path in stage one depends upon the time it takes to compute a single N/m point FFT followed by N/m complex multiplies to apply the phase shift for that column of the resulting Zak transform. The cost of stage two is governed by the cost of the FFT of the rows.

The time to compute or the throughput of the MR FFT system is determined by the cost of the longest path in stage one plus the longest path in stage two. The result of this is the Longest Path column in Table 3.1 Next, the Parallel Complexity is calculated which is the total number of processors necessary times the Longest Path. The total number of calculation necessary to perform the multirate FFT are given under the Sequential Complexity column. This represents the calculation that would need to be done if the algorithm where implemented on a single processor.

A graphical comparison of the relative time it takes to implement a N point MR FFT and FFT is given in Figure 3.5. The figure shows the fraction of the time it takes to calculate the MR FFT for different decimation values relative to the FFT. The FFT is a sequential algorithm which is, therefore, processed along a single path and is defined to take a time of one to implement. The MR FFT's have as many processors as necessary to achieve

Table 3.1: Computational Cost of Multirate Fast Fourier Transform

Operation	Stage One	Stage Two	Longest Path $m = \sqrt{N}$	Parallel Complexity	Sequential Complexity
×	$\frac{N}{m}\log_2\frac{N}{m} + 4$	$m\log_2 m - 3m + 4$	$\sqrt{N}\log_2 N - 3\sqrt{N} + 5$	$N\log_2 N - 3N + 5\sqrt{N}$	$N\log_2 N - 3N + 2\sqrt{N} + 3$
+	$3\frac{N}{m}\log_2\frac{N}{m} + 4$	$3m\log_2 m - 3m + 4$	$3\sqrt{N}\log_2 N - 3\sqrt{N} + 5$	$3N\log_2 N - 3N + 5\sqrt{N}$	$3N\log_2 N - 3N + 2\sqrt{N} + 3$

the fastest possible throughput. In the case of decimation factors of four and eight, This leads to large numbers of processors being idle in the first stage since only four or eight processors are needed at that point, but $N/4$ and $N/8$ processors are needed in the second stage. For a balanced load between the processors, it is necessary to let the decimation factor be the square root of the length of the input. This is the line labeled, "Efficient MR FFT." For the case of $N = 1024$, the efficient MR FFT would require 32 processors and would be approximately 31.6 times faster than the sequential FFT.

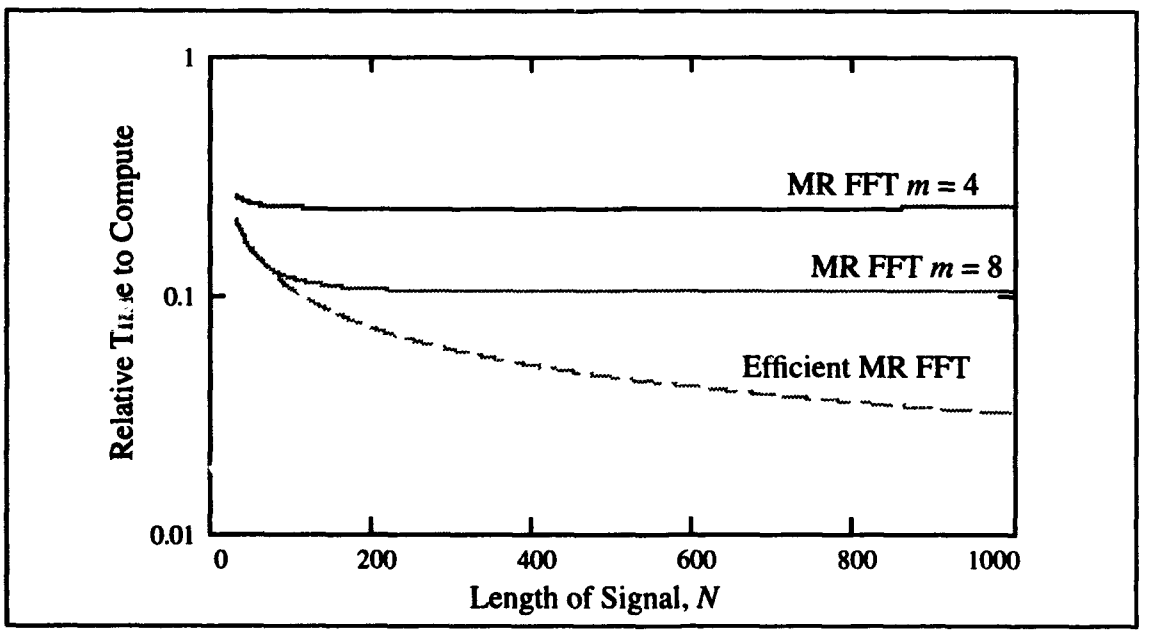


Figure 3.5. Relative Time to Compute MR FFT Compared to Sequential FFT.

3.4.2. The Multirate Discrete Hartley Transform. As a further example of the usefulness of the paradigm presented in this chapter, the Discrete Hartley Transform is examined in this section. This section is divided into two sub-sections: first, the theoretical background for the multirate implementation of the DHT is presented in 3.4.2.1. Next, the MR DHT algorithm is given in 3.4.2.2

3.4.2.1. MR DHT Theory. Neng-Chung Hu, Hong-I Chang and O. K. Ersoy developed a modified DHT in [32] called the generalized DHT (GDHT). The authors state that it generalizes the DHT the same way the DFT is generalized; however, a more useful connection for this example is the similarity between the relationship of the DHT to the odd GDHT and relationship of DCT-I to DCT-II and DST-I to DST-II. The odd GDHT is given by

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} H_2\left(n+\frac{1}{2},k\right)x(n) \\ &= \sum_{n=0}^{N-1}\left[\cos\left(\frac{2\pi\left(n+\frac{1}{2}\right)k}{N}\right)+\sin\left(\frac{2\pi\left(n+\frac{1}{2}\right)k}{N}\right)\right]x(n) \end{aligned} \tag{3.23}$$

where $k = 0,\ldots, N-1$. Applying (3.9), the two stage multirate formalism for matrix-vector products, (3.23) can be rewritten as

$$X(pL+f) = \sum_{i-0}^{m-1}\sum_{n=0}^{L-1}\operatorname{cas}\left(\frac{2\pi\left(nm+i+\frac{1}{2}\right)(pL+f)}{N}\right)x(nm+i) \tag{3.24}$$

where $p = 0, 1,\ldots, m-1$ and $f = 0, 1,\ldots, L-1$, $L = N/m$ and $\operatorname{cas}(\alpha) = \cos(\alpha) + \sin(\alpha)$.

Making use of the identities,

$$\begin{aligned} \cos(\alpha+\beta) &= \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta) \\ \sin(\alpha+\beta) &= \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta) \end{aligned}, \tag{3.25}$$

equation (3.24) can be expressed as

$$X(pL+f) = \sum_{i=0}^{m-1}\sum_{n=0}^{L-1} \cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)(pL+f)}{N}\right)\mathrm{cas}\left(\frac{2\pi nf}{L}\right)x(nm+i)$$
$$+ \sum_{i=0}^{m-1}\sum_{n=0}^{L-1} \sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)(pL+f)}{N}\right)\mathrm{cas}\left(\frac{2\pi nf}{L}\right)x(N-m(n+1)+i) \tag{3.26}$$

since

$$\mathrm{cas}\left(\frac{2\pi n(pL+f)}{L}\right) = \mathrm{cas}\left(\frac{2\pi nf}{L}\right). \tag{3.27}$$

Define

$$X_i(f) = \sum_{n=0}^{L-1} \mathrm{cas}\left(\frac{2\pi nf}{L}\right)x(nm+i) \tag{3.28}$$

and

$$\tilde{X}_i(f) = \sum_{n=0}^{L-1} \mathrm{cas}\left(\frac{2\pi nf}{L}\right)x(N-m(n+1)+i)\,. \tag{3.29}$$

Then, (3.26) can be expressed as

$$X(pL+f) = \sum_{i=0}^{m-1} \cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)p}{m}\right)\left[\cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)X_i(f) + \sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)\tilde{X}_i(f)\right]$$
$$+ \sum_{i=0}^{m-1} \sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)p}{m}\right)\left[\cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)\tilde{X}_i(f) - \sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)X_i(f)\right]. \tag{3.30}$$

Because cosine is an even symmetric function,

$$\cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)p}{m}\right) = \cos\left(\frac{2\pi\left(m-i-\frac{1}{2}\right)p}{m}\right) \tag{3.31}$$

and sine being an odd symmetric function means

$$\sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)p}{m}\right) = -\sin\left(\frac{2\pi\left(m-i-\frac{1}{2}\right)p}{m}\right) \tag{3.32}$$

for $i = 0, 1, \ldots, m/2 - 1$. This implies (3.30) can be reduced to

$$X(pL+f) = \tag{3.33}$$

$$\left(\frac{1}{k_p}\right)\left(k_p \sum_{i=0}^{M-1} \cos\left(\frac{\pi\left(i+\frac{1}{2}\right)p}{M}\right)\hat{X}_i(f) + k_p \sum_{i=0}^{M-1} \sin\left(\frac{\pi\left(i+\frac{1}{2}\right)p}{M}\right)\hat{X}_i'(f)\right)$$

where $M = m/2$,

$$\begin{aligned}\hat{X}_i(f) &= \left[\cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)X_i(f) + \sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)\tilde{X}_i(f)\right] \\ &+ \left[\cos\left(\frac{2\pi\left(m-i-\frac{1}{2}\right)f}{N}\right)X_{m-i-1}(f) + \sin\left(\frac{2\pi\left(m-i-\frac{1}{2}\right)f}{N}\right)\tilde{X}_{m-i-1}(f)\right]\end{aligned}, \tag{3.34}$$

$$\begin{aligned}\hat{X}_i'(f) &= \left[\cos\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)X_i(f) - \sin\left(\frac{2\pi\left(i+\frac{1}{2}\right)f}{N}\right)\tilde{X}_i(f)\right] \\ &- \left[\cos\left(\frac{2\pi\left(m-i-\frac{1}{2}\right)f}{N}\right)X_{m-i-1}(f) - \sin\left(\frac{2\pi\left(m-i-\frac{1}{2}\right)f}{N}\right)\tilde{X}_{m-i-1}(f)\right]\end{aligned} \tag{3.35}$$

and

$$k_p = \begin{cases} \sqrt{M} & p = 0 \\ \sqrt{\frac{2}{M}} & p = 1, 2, \ldots, M-1. \end{cases} \tag{3.36}$$

This is the multirate DHT.

In other words, (3.33) is the sum of the type II Discrete Cosine Transforms of $\hat{X}_i(f)$ and the type II Discrete Sine Transforms of $\hat{X}_i'(f)$. (For more information on the DCT and DST, see [42].)

Equation (3.33) directly calculates the values for $p = 0, 1, \ldots, M-1$. The values of $X(pL+f)$ for $p = M, M+1, \ldots, m-1$ are found indirectly. Prior to adding the results of the Cosine and Sine transforms together, the value of the transforms for $p = M, M+1, \ldots, m-1$ must be calculated. For the DCT-II, they are given by $-X_c((m-p+1)L+f)$ for $p = M+1, M+2, \ldots, m-1$ (X_c being the result of the Cosine transform). The value of the DCT-II for M is zero. For the DST-II, the value of the transform for $p = M+1, M+2, \ldots, m-1$ is given by $X_s((m-p+1)L+f)$, and for $p = M$, it is the sum

$$X_s(f) = \sum_{i=0}^{M-1} (-1)^i \hat{X}_i'(f) \tag{3.37}$$

The DHT can also be expressed in terms of tensor notation as was done for the MR FFT. As with the MR FFT, the first step is to perform a stride by m permutation, i.e. $P_{N,m}x$. The permuted input is broken into two paths. The first path is operated on by $I_m \otimes H_L$ where H_L is the DHT operator of dimension L. The second path is permuted by the inversion operator, $I_m \otimes J_L$, where J_L is defined as

$$J_L = \begin{bmatrix} & & 1 \\ & \cdot & \\ & \cdot & \\ \cdot & & \\ 1 & & \end{bmatrix}_{L \times L} \tag{3.38}$$

and then operated on by $I_m \otimes H_L$.

The result from each path is again split into two branches. In the first branch the result is multiplied by a diagonal operator, D_c, and the second branch is multiplied by the diagonal operator, D_s. The diagonal operators are the direct sums

$$D_\alpha = \bigoplus_{i=0}^{m-1} \Omega_{i,l}^{\alpha} \tag{3.39}$$

for α equal to c or s and $\Omega_{i,l}^{\alpha}$ is defined as

$$\Omega_{i,l}^{c} = \begin{bmatrix} 1 & & & \\ & \cos\left(\frac{2\pi i}{N}\right) & & \\ & & \ddots & \\ & & & \cos\left(\frac{2\pi i\,(l-1)}{N}\right) \end{bmatrix}_{l \times l} \tag{3.40}$$

and

$$\Omega_{i,l}^{s} = \begin{bmatrix} 1 & & & \\ & \sin\left(\frac{2\pi i}{N}\right) & & \\ & & \ddots & \\ & & & \sin\left(\frac{2\pi i\,(l-1)}{N}\right) \end{bmatrix}_{l \times l}, \tag{3.41}$$

respectively.

The four branches are recombined such that the first branch of the first path is added to the second branch of the second path (call this recombined path one) and the second branch of the first path is subtracted from the first branch of the second path (call this re-combined path two). The four branches have thus been recombined into two paths.

The paths are permuted by a combination stride permutation operator, $P_{N,L}P_{N,m}$. Once the permutations have been performed, the first path is operated on by $I_L \otimes C_m$, and the second path is operated on by $I_L \otimes S_m$ where C_m is the kernel with elements $\cos(2\pi ip/m)$ and S_m is the kernel with elements $\sin(2\pi ip/m)$, $i,p = 0, 1,\ldots, m-1$. C_m and S_m are

modified Cosine and Sine Transforms. Each block of the operators $I_L \otimes C_m$ and $I_L \otimes S_m$ are calculated by first forming $\hat{X}_i(f)$ and $\hat{X}_i'(f)$ and then taking the DCT and DST, respectively.

The tensor notation version of the MR DHT can now be stated as

$$X(k) = P_{N,L}\{(I_L \otimes C_m) P_{N,L} P_{N,m} [D_c (I_m \otimes H_L) P_{N,m} x(n) \tag{3.42}$$

$$+ D_s (I_m \otimes H_L)(I_m \otimes J_L) P_{N,m} x(n)] + (I_L \otimes S_m) P_{N,L} P_{N,m}$$

$$[D_c (I_m \otimes H_L)(I_m \otimes J_L) P_{N,m} x(n) - D_s (I_m \otimes H_L) P_{N,m} x(n)]\}$$

3.4.2.2. MR DHT Algorithm. The first stage of the MR DHT implements the $2m$ L-point DHT's calculated using a standard Fast DHT algorithm such as that given in [47]. There are m DHT's associated with the decimated signal and m associated with the decimated and reversed signal. The second stage performs the multiplication by $\Omega_{i,l}^{\alpha}$ for $\alpha = c$ and s. For each i, there are four sets of multiplications, $\Omega_{i,l}^{c}$ with the DHT of the decimated signal and the decimated and reversed signal and $\Omega_{i,l}^{s}$ with the DHT of the decimated signal and the decimated and reversed signal.

Each combination is calculated in independent processors. For maximum speed, this requires $4m$ processors, but maximum efficiency is obtained with $2m$ processors. The algorithm given here is the maximum efficiency version. The results of the $\Omega_{i,l}^{\alpha}$ multiplications are summed together producing $2m$ L-point sequences which are output to two $L \times m$ arrays. The first array is the solution of

$$A_i(f) = \left[\cos\left(\frac{2\pi i f}{N}\right) X_i(f) + \sin\left(\frac{2\pi i f}{N}\right) \tilde{X}_i(f)\right] \quad i \in [0, m) \quad , \tag{3.43}$$

and the second array is the solution of

$$B_i(f) = \left[\cos\left(\frac{2\pi i f}{N}\right) \tilde{X}_i(f) - \sin\left(\frac{2\pi i f}{N}\right) X_i(f)\right] \quad i \in [0, m) \tag{3.44}$$

from equation (3.30). This is the end of the second stage.

The final stage calculates the modified cosine and sine transforms of the rows of the two arrays, respectively, and sums the result to produce the permuted DHT, $X(pL + f)$. A block diagrams of the multirate DHT is shown in Figure 3.6. The MR DHT algorithm is given in Algorithm 3.2.

Algorithm 3.2: Multirate Discrete Hartley Transform

	Barrier 0: Start
Stage 1:	Load N data points by parsing N/m data points into $2m$ processors. (*This includes both the decimated data and the decimated and reversed data which go to separated groups of processors.*)
	Calculate the DHT of each subsequence
	Barrier 1:
Stage 2:	Swap data between DHT processor pairs
	Multiply by Cosine and Sine terms
	Output results to intermediate result array
	Barrier 2:
Stage 3:	Load rows of intermediate result array into processors
	Calculate modified Sine and Cosine transforms
	Output result
	Barrier 3: End

Like the multirate FFT, the multirate DHT performs identical operations in each channel on differing sets of data (all of the same length) meaning there is only one instruction set for a number of processors. This implies that the algorithm is suitable for implementation in a SIMD architecture [27].

3.5. Conclusion

In this chapter, a new multirate paradigm was introduced. The idea that multirate was limited to the area of filter bank design was discarded and replaced by a realization

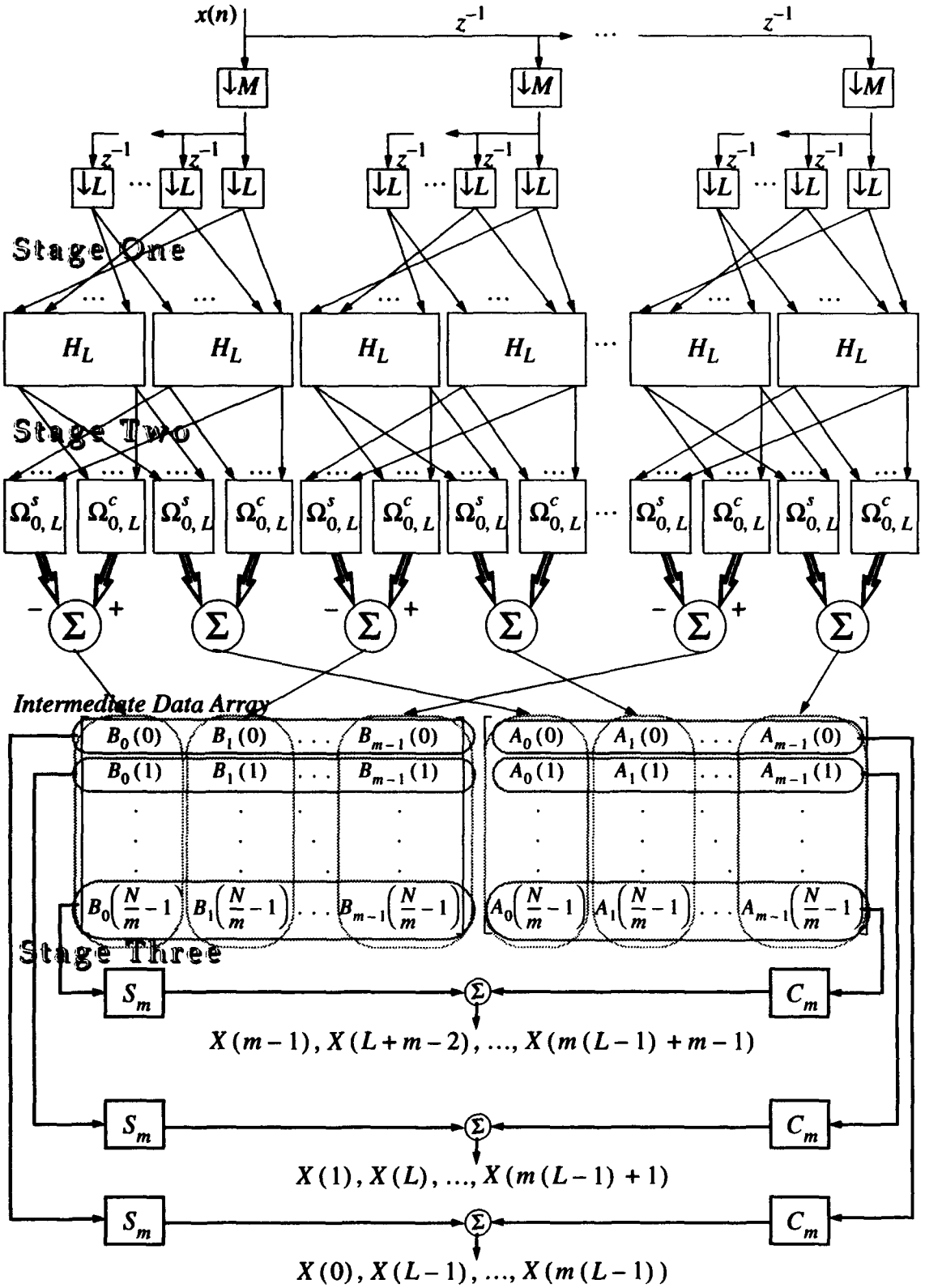


Figure 3.6. Block diagram of multirate Discrete Hartley Transform.

that multirate is a powerful divide and conquer strategy for the entire field of Numerical Linear Algebra.

Multirate, as a computational paradigm, was successfully applied to vector-vector, matrix-vector and matrix-matrix operations. Since these operation form the foundation of modern numerical linear algebra, it can be seen that multirate as a computational paradigm may be applied to all problems encountered throughout the field of numerical linear algebra.

Using the multirate numerical linear algebraic operations developed here, the new multirate paradigm was applied to two commonly used algorithms: the FFT and the DHT. The resulting multirate algorithms are parallel implementations of the FFT and DHT, respectively. They require slightly more computations to calculate; however, when processed in parallel, they can deliver dramatic increases in throughput.

It has now been demonstrated that multirate as a computational paradigm is suitable for all numerical linear algebraic problems including many encountered in signal processing. It remains to apply the paradigm to the problem of the GDTFD. The first step in this process is to design the computational tools which will allow any discrete kernel to be used with the multirate TFD. In Chapter 4, these tools are developed.

4. Kernel Design Techniques for Alias-Free Time-Frequency Distributions

4.1. Introduction

The kernels used for the GDTFD are designed in either the time-lag domain or the ambiguity domain. In order for them to be used with the MRTFD presented in the following chapters, they must be discretely represented in the time-lag domain. For certain kernels, this can be difficult. Therefore, in this chapter, three different methods are presented which allow kernels developed in either domain to be discretely represented in the other.

Recently, Jeong and Williams introduced a generalized method to calculate an Alias-Free Generalized Discrete Time-Frequency Distribution (AF-GDTFD) [29][30]. In this dissertation, the term AF-GDTFD will be used synonymously with the term GDTFD as the AF-GDTFD is the form of GDTFD which is of interest in this work. Rather than using only half of the possible sample points in the discrete bilinear signal as the Discrete TFD (DTFD) does, their method makes use of all the available samples of the bilinear signal. Since the DTFD uses only half of the available sample points, it has a discrete bandwidth of π and will suffer aliasing for a Nyquist sampled signal. The Jeong and Williams GDTFD is 2π periodic since it keeps all the sample points and is, therefore, alias-free for a Nyquist sampled signal.

The GDTFD is an $O(N^3)$ process and requires the continuous form of a kernel in the time-lag plane be known. The modifications to the basic GDTFD decrease the computational complexity to $O(N^2 \log N)$. They, also, allow the use of kernels designed in the ambiguity plane where the continuous form of the kernel in the time-lag plane is unavailable or difficult to obtain.

This chapter is divided into two sections. First, a description of the modifications made to the alias-free algorithm developed by Jeong and Williams [29] to create an

$O(N^2 \log N)$ algorithm is given, and second, a simple method to modify kernels designed in the ambiguity plane to the alias-free form is developed. The ability to design kernels in the time-lag domain will be critical to the development of the Multirate Time-Frequency Distribution.

4.2. A Faster GDTFD

Assume the number of samples, N, is a power of two. Then, the method reported by Jeong and Williams in [29] is an $O(N^3)$ process. With a slight modification, the GDTFD can be implemented in $O(N^2 \log N)$ computations. The faster form is based upon the Cohen's class using the form

$$C(t, \omega;\phi) = \iint A(\theta, \tau)\, \phi(\theta, \tau)\, e^{-j\tau\omega - j\theta t} d\tau d\theta \tag{4.1}$$

where $A(t, \tau) = \frac{1}{2\pi}\int f\left(u + \frac{\tau}{2}\right) f^*\left(u - \frac{\tau}{2}\right) e^{jut} du$ is the ambiguity function of f. The discrete form of (4.1) is given by

$$C(n, k;\phi) = \sum_{k=-N}^{N-1} \sum_{p=-\frac{N}{2}}^{\frac{N}{2}-1} \phi(p, \tau) \left(\sum_{u=-\frac{N}{2}}^{\frac{N}{2}-1} R_f(u, \tau)\, e^{\frac{j2\pi up}{N}} \right) e^{\frac{-j2\pi pn}{N}}\, e^{\frac{-j2\pi k\tau}{N}}. \tag{4.2}$$

To calculate the GDTFD, the first step is to determine the ambiguity function of f. It is found by calculating the inverse FFT of the rows of $R_f(u,k)$—an $O(N^2 \log N)$ process. Next, the product of the kernel and the ambiguity function is determined—an $O(N^2)$ process. Finally, the two-dimensional FFT is performed to find the GDTFD—an $O(N^2 \log N)$ process. Thus, the overall process is $O(N^2 \log N)$.

In comparison, the method given in [29] makes use of the form

$$C(n, k;\phi) \;\text{:-}\; \mathcal{F}\left[R_f^{AF}(n, \tau)\right]_{\tau \to k} \tag{4.3}$$

$$= \mathcal{F}\left[\sum_{m \in Z} f(m + \tau)\, f^*(m)\, \psi'(n - m, \tau)\right]_{\tau \to k}$$

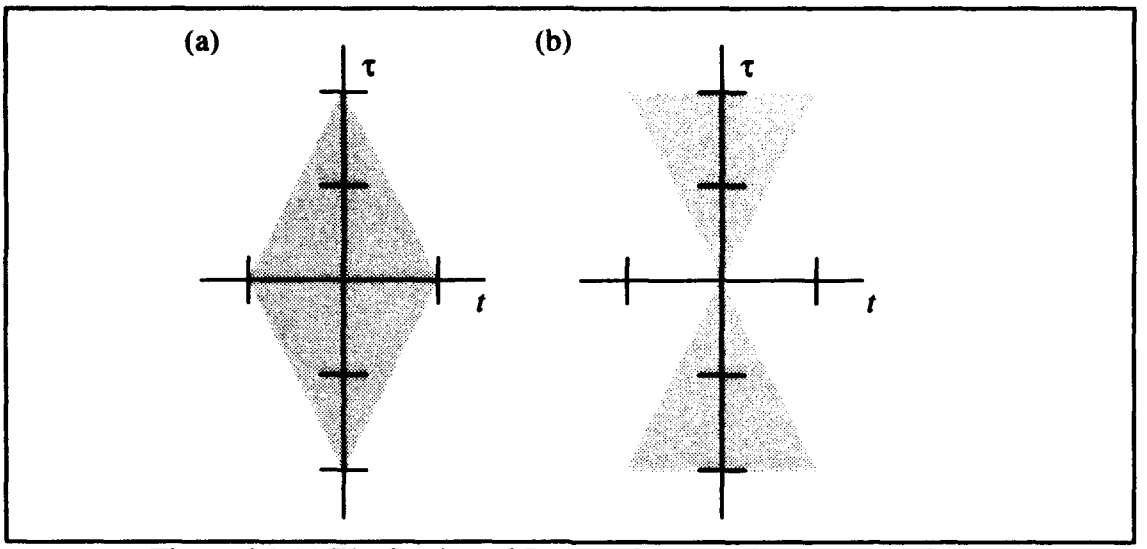


Figure 4.1 (a) The Region of Support for the Bilinear Form of the Signal (i.e. R_f). (b) The Region of Support for a Kernel of the "Bow-Tie" Class.

where ψ' is the appropriately sampled and shifted kernel in the time-lag plane and $\mathcal{F}$ is the Fourier transform operator. The calculation of the linear convolution within the brackets requires $O(N^3)$ operations to complete followed by an $O(N^2 \log N)$ operation to calculate the Fourier transform of the columns; thus, the overall process is dominated by the linear convolution and is $O(N^3)$.

Because of the fixed overhead associated with the FFT, for small N (e.g. $N < 16$) solving the linear convolution directly may be faster. Also, the support of the kernel plays a role as well. The support of R_f is diamond shaped (See Figure 4.1a.). If the kernel is of the "bow-tie" class (See Figure 4.1b.), then aliasing does not occur and the circular convolution implemented by the FFT creates no error. However, if the kernel extends beyond the "bow-tie" region, then aliasing will occur if the rows of the kernel and the bilinear signal are not zero padded. This will result in the cross-over point (i.e. the point at which the FFT method is faster than the original method) increasing by a factor of four.

One subtlety is encountered as a result of using the DFT. For the odd rows of the kernel, the sampling shifts for the kernel and the bilinear signal must be in opposite directions. For example, if

$$R_f(t,k) = \begin{cases} f\left(t+\frac{k}{2}\right)f^*\left(t-\frac{k}{2}\right) & k \text{ even} \\ f\left(t+\frac{k+1}{2}\right)f^*\left(t-\frac{k-1}{2}\right) & k \text{ odd} \end{cases} \tag{4.4}$$

then kernel, $\phi(p,k)$, which is the inverse Fourier transform in the p direction of a kernel function, ψ, defined in the t-k plane, is given by

$$\phi(n,k) = \begin{cases} \sum_{t=-N/2}^{N/2-1} \psi(t,k)\, e^{j2\pi tn/N} & k \text{ even} \\ \sum_{t=-N/2}^{N/2-1} \psi\left(t-\frac{1}{2},k\right) e^{j2\pi tn/N} & k \text{ odd} \end{cases} \tag{4.5}$$

This is now a discrete formulation which uses the complete sampling bandwidth and is 2π periodic instead of π periodic and can be implemented in $O(N^2 \log N)$.

4.3. A Simple Method to Design Alias-Free Kernels

One problem with the GDTFD is the calculation of an appropriately sampled kernel, ψ. Often, it is convenient to design kernels in the ambiguity plane where the kernel acts as a two-dimensional multiplicative filter. Unfortunately, that filter must be altered such that the values used are equivalent to the inverse Fourier transform of ψ on the hexagonal grid. This will be called the *shifted* kernel function, $\phi(\theta,\tau)$.

4.3.1. Method 1: The Continuous Kernel (CK) Method. Although Jeong and Williams address the continuous kernel in the time-lag domain [29], no work has previously been done with continuous kernel in the ambiguity plane. The most straightforward method of calculating the shifted $\phi(\theta,\tau)$ is to select a continuous ambiguity domain kernel, $\phi(\theta,\tau)$, which meets some user defined criteria and to take its Fourier transform in the θ direction. This yields the continuous time-lag domain kernel, ψ. The shifted sampled $\phi(n,k)$ (note this is now discrete) is then found via equation (4.3).

4.3.2. Method 2: Interpolation Method. The second method approaches the prob-

lem of finding the shifted $\phi(n,k)$ in a different manner. Note from Chapter 2, the outer product formulation is actually performing a convolution of a kernel defined in t-τ with the bilinear signal. Examine the convolution in more detail. Let a given even indexed row of the rotated outer product of the signal be represented by X[k], an odd indexed row be represented by X[k+1/2]. Further let an even indexed row of the kernel be given by Y[k] and for an odd row let it be Y[k-1/2]. The result of the convolution of the even rows of the kernel and bilinear signal is

$$X[k] * Y[k] = \mathcal{F}[x[n]\,y[n]] = Z[k], \tag{4.6}$$

and the convolutions of the odd rows of the kernel with the odd rows of the bilinear signal is

$$\begin{aligned} X\left[k+\frac{1}{2}\right] * Y\left[k-\frac{1}{2}\right] &= \mathcal{F}[e^{-j\pi n/N} x[n]\, e^{j\pi n/N} y[n]] \\ &= \mathcal{F}[x[n]\,y[n]] = Z[k]. \end{aligned} \tag{4.7}$$

Thus, the convolution of both the even and odd rows results in a function sampled only at integer values (i.e. rectangular sampling). Note that it is necessary in finite circular convolution for the shifts of X and Y to be in opposite direction.

The above result is the justification for attempting to find the shifted kernel $\phi(n,k)$. Suppose the kernel were not shifted. Equation (4.6) would be unchanged, but (4.7) would become

$$\begin{aligned} X\left[k+\frac{1}{2}\right] * Y[k] &= \mathcal{F}[e^{-j\pi n/N} x[n]\, y[n]] \\ &= \mathcal{F}[e^{-j\pi n/N} z[n]] = Z\left[k+\frac{1}{2}\right]. \end{aligned} \tag{4.8}$$

From this it is seen that the result of not calculating the shifted $\phi(n,k)$ is the convolution of the rotated outer product of the signal and kernel will still be on the hexagonally decimated rectangular grid.

This structure can be exploited by interpolating to fill in the missing elements, take the Fourier transform along τ and create a TFD which is 2π periodic and has twice the resolution in time; hence, the time axis in the examples for the interpolation method is twice the length of the frequency axis.

4.3.3. Method 3: Phase Shift Method. Another method which could be used to find the shifted $\phi(n,k)$ is to sample $\phi(\theta,\tau)$ on a rectangular grid at intervals of 1/2 along θ for τ odd, take its discrete Fourier transform and decimate the result. This is equivalent to having sampled $\psi(t,\tau = \text{odd integer})$ at intervals of 1/2. If ψ is decimated by taking the odd indexed samples (i.e. $\psi(t=0.5, \tau = \text{odd integer})$, $\psi(t = 1.5, \tau = \text{odd integer})$, ...) and take the inverse DFT of the resulting sequence, then the result is the shifted $\phi(n,k)$.

This method is exactly equivalent to the well known transform relationship,

$$e^{j2\pi ln/N} x[n] \leftrightarrow X[k-l], \tag{4.9}$$

where $x[n] \leftrightarrow X[k]$ represent a N point DFT pair [39]. As a result, this is called the phase shift method. In practice, the phase shift relationship is used rather than decimating the oversampled DFT and inverting.

With l=0.5, the shifted $\phi(n,k)$ can be easily be determined by starting with the unshifted $\phi(n,k)$ and modifying that as follows:

$$\phi(n,k) = \begin{cases} \phi(n,k) & k \text{ even} \\ e^{j\pi n/N}\phi(n,k) & k \text{ odd.} \end{cases} \tag{4.10}$$

4.4. Comparison of Methods.

The Butterworth kernel [40] demonstrates the difficulty that can arise using the CK method. The Butterworth kernel is given by

$$\phi(\theta,\tau) = \frac{1}{1+\left(\frac{\theta}{\theta_1}\right)^{2N}\left(\frac{\tau}{\tau_1}\right)^{2M}} \tag{4.11}$$

where τ_1 and θ_1 are constants greater than zero and N and M are integers greater than zero. Using contour integration, the Fourier transform of (4.11) is

$$\psi(t,\tau) = \lim_{z \to z_k} 2\pi j \theta_1^{2N} \left(\frac{\tau_1}{\tau}\right)^{2M} \left(\sum_{k=0}^{N-1} \frac{(z - z_k)\, e^{jz|t|}}{\theta_1^{2N} \left(\frac{\tau_1}{\tau}\right)^{2M} + z^{2N}} \right) \tag{4.12}$$

where

$$z_k = \theta_1 \left(\frac{\tau_1}{\tau}\right)^{M/N} e^{j\pi(2k+1)/2N}. \tag{4.13}$$

For a more detailed discussion of the calculation of the time-lag Butterworth kernel and a slightly different resulting form see [40].

While (4.12) does have a closed form solution, it does not follow that all kernels defined in the ambiguity plane have an inverse Fourier transform in the θ direction. Another problem occurs when $\psi(t,\tau)$ is sampled and the inverse discrete Fourier transform is taken. Since neither (4.11) or (4.12) have finite support, sampling the waveforms with a finite number of points over a finite window is going to introduce error, and for this case, the DFT magnifies the error. For example, the DFT of $\phi(n,k)$ where N is 128 bears only little resemblance to $\psi(p,k)$. (See Figure 4.4.) The result is, in fact, two different kernels which will asymptotically approach each other as the number and density of samples goes up. The question which naturally arises is which one is best? About the only thing that can be said is that it depends upon the particular signal, kernel and choice for N.

Unfortunately, the CK method requires the continuous form of the kernel in the time-lag plane. Thus, for a kernel defined in the ambiguity plane, if a closed form solution to the continuous Fourier transform does not exist, the CK method will not work. Further, in applications where the kernel is changing on a regular basis but cannot be predetermined, it may be impractical or impossible to use the CK method. As a result of the re-

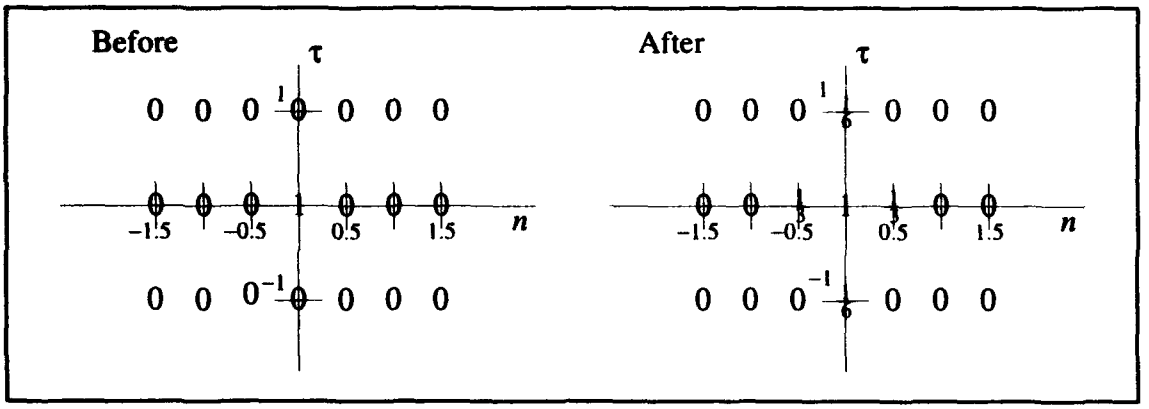


Figure 4.2. Result of Convolution of Impulse and Kernel (a) Before and (b) After Interpolation.

strictive requirement of needing the continuous form of the kernel in the time-lag plane, the CK method is not as robust as the Interpolation or the phase shift methods.

In the case of the Interpolation method, with two dimensional interpolation, some time and frequency resolution is lost. For example, suppose a weighted linear interpolation is used and the signal is the unit sample sequence. Then, the result of convolving the bilinear form of this signal with a kernel where $\psi(n,0) = 1$ for $n = 0$ and 0 otherwise is one at the origin and zero elsewhere. (See Figure 4.2a.) After interpolation, the impulse gets spread. Now, the impulse will show in the -0.5, 0 and 0.5 columns of the TFD. Further, the terms in the zero column will decay with τ of increasing magnitude rather than being a constant as it should. (See Figure 4.2b.) This is the worst case example for the interpolation method and demonstrates that the localization of a feature in the time-frequency plane can decrease by as much 50 percent. Localization performance could be improved by using more sophisticated types of interpolation. However, this comes with increased computational complexity.

The phase shift method does not require the continuous form of the kernel in either the ambiguity or time-lag plane; however, this method is only approximately equal to the shifted $\phi(n,k)$ found by (4.5) and is dependent upon the size of N. As N increases, the difference between (4.5) and (4.10) decreases. In practice, the transform relationship given in

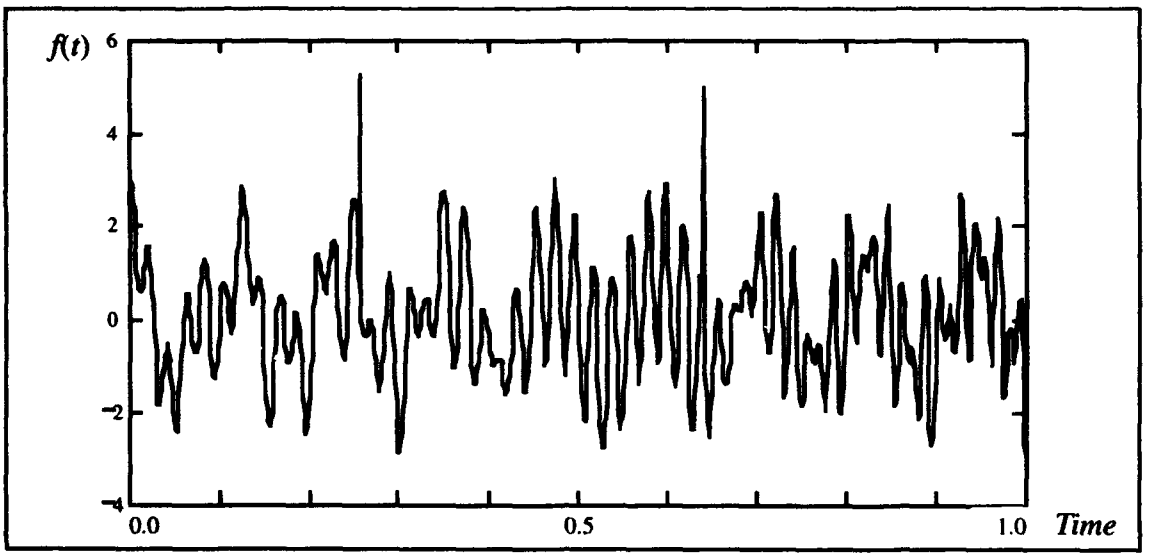


Figure 4.3. Multicomponent Signal Consisting of Two Tones, Two Impulses and a Chirp.

(4.9) is only approximate with an error that decays as $1/N$, but even for N as low as 128, the effect on the TFD is minor. See Table 4.1 at the end of this section for a side-by-side comparison of the three methods.

To demonstrate the differences between the methods discussed above, examine the result of applying each method to the Butterworth kernel given by (4.5) in the GDTFD of a multicomponent signal. The signal consists of two tones, one at 0.136π and the other at 0.777π, two impulses at $t = 0.25$ and $t = 0.625$, and a rising chirp starting at 0.18π with a slope of 0.562π. It is depicted in Figure 4.3.

If, in (4.12), $\theta_1 = \tau_1 = 10.0$ and $M = N = 1$, by the CK method, the kernel in the time-lag plane becomes

$$\psi(t,k) = \frac{100\pi}{k} e^{-100t/k} \tag{4.14}$$

If this is sampled appropriately and the inverse DFT is taken as per (4.5), the kernel which results is shown in Figure 4.4. Compare this result to the kernel generated by the shifted kernel calculated via the phase shift method. Here, the kernel is calculated in the ambiguity plane by sampling (4.5) and applying a phase shift as given by (4.7). (See Figure 4.4.)

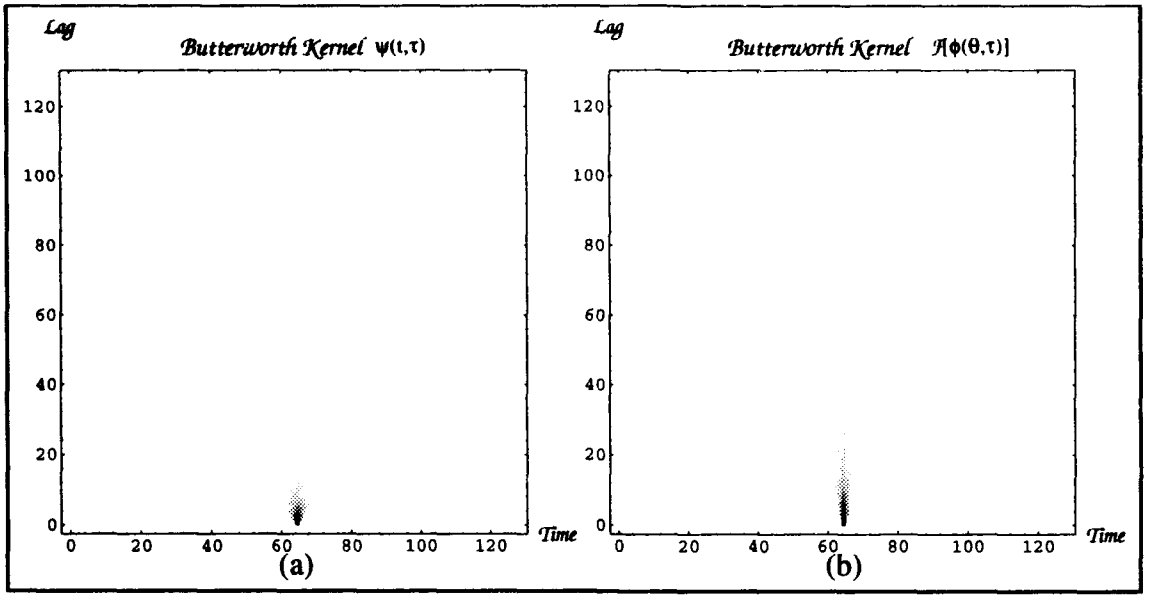


Figure 4.4. (a) Comparison of Discrete Butterworth Kernel Sampled in Time-Lag and (b) Kernel Sampled in Ambiguity Plane and Fourier transformed.

Although the continuous forms of the Butterworth kernel in the time-lag plane and ambiguity plane are equivalent when calculating the continuous TFD, when they are sampled and windowed that relationship no longer holds. Hence, the TFD's shown in Figure 4.5a and Figure 4.5c are actually two different TFD's which will approach each other as the size of the window and the density of the sampling increase.

The Interpolation method produces a TFD shown in Figure 4.5b. In this case, a weighted linear interpolation based on the four nearest neighbors in the time-lag domain is used. Note the interpolation is performed after the kernel and signal have been convolved but prior to taking the Fourier transform in the lag direction. As expected, the tones and impulses are more difficult to detect visually and are somewhat less localized in the Interpolated GDTFD than they are in either the Continuous Kernel GDTFD or phase shifted GDTFD.The difference between the GDTFD's created using the Continuous Kernel method and the Phase Shift method are the result of the kernels not being equivalent (i.e. they will not produce they same distribution since they are not the same themselves).

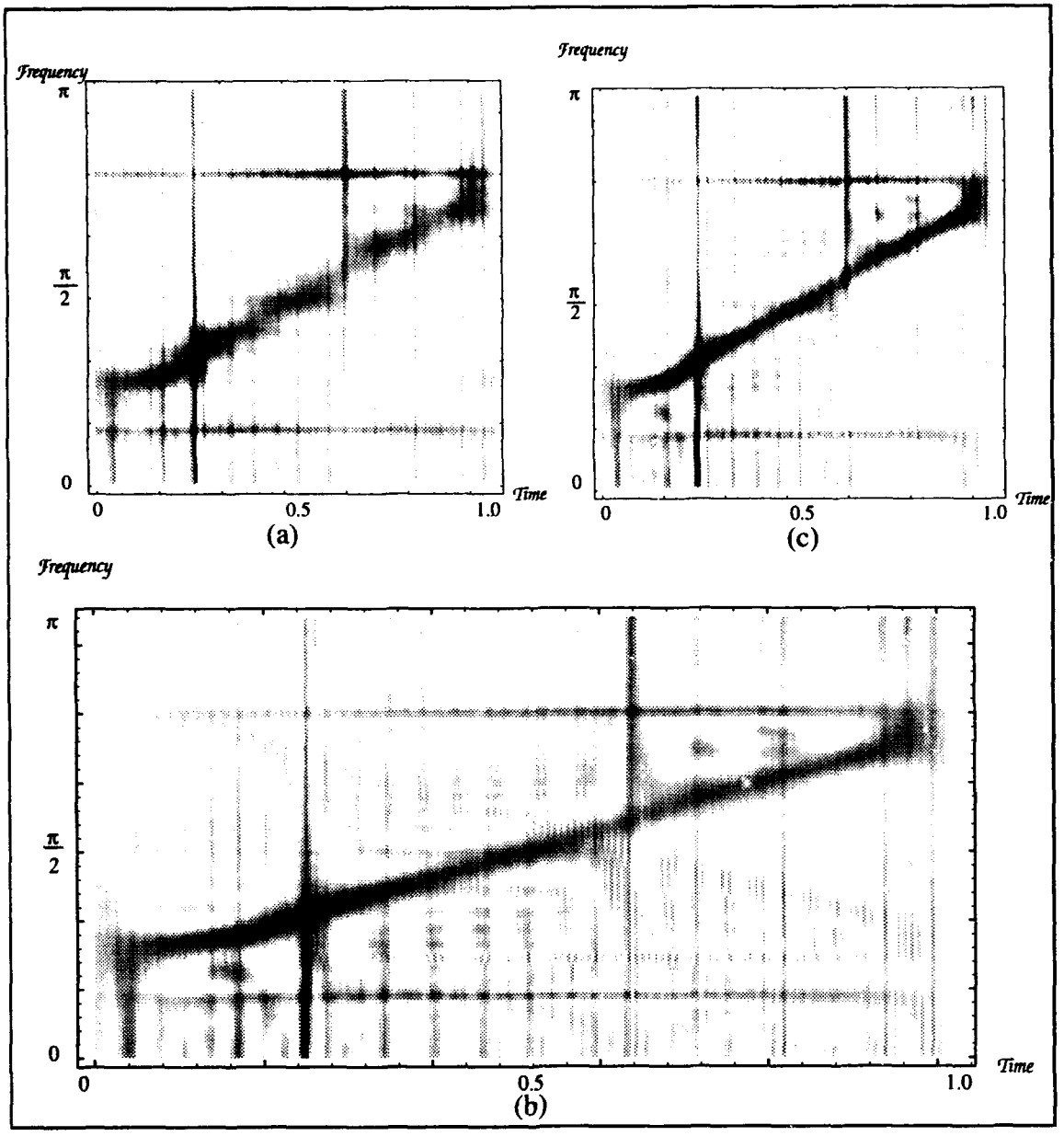


Figure 4.5. GDTFD Using the Butterworth Kernel via (a) the Continuous Kernel Method, (b) Interpolation Method and (c) Phase Shift Method.

4.5. Further Examples.

To demonstrate the versatility of the methods presented, they were also applied to the Binomial Reduced Interference Kernel developed by W. J. Williams and J. Jeong [56].

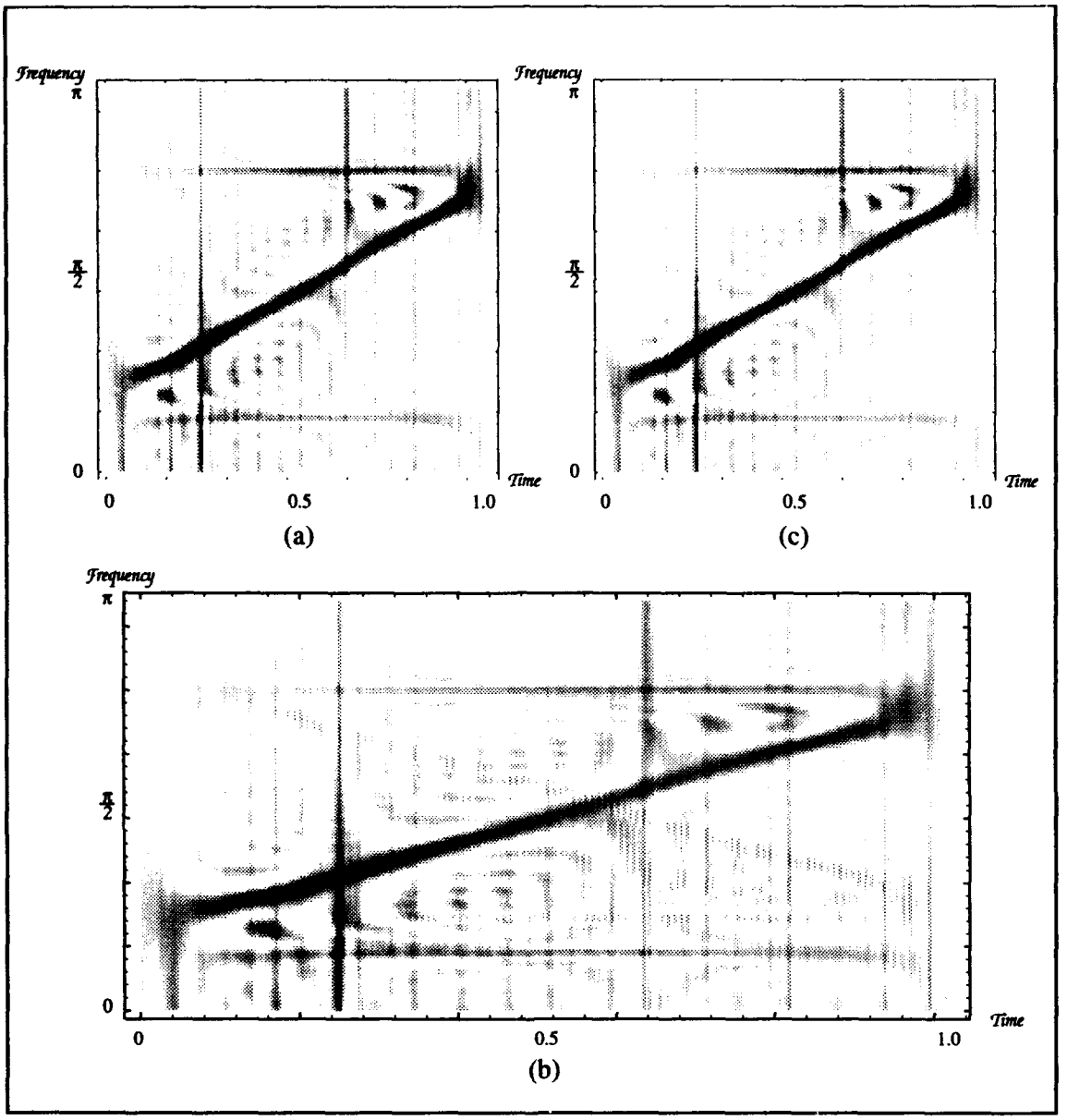


Figure 4.6. GDTFD Using the Binomial Kernel via (a) the Continuous Kernel Method (b) the Interpolation Method and (c) the Phase Shift Method.

The Binomial kernel provides a good trade-off between cross-term suppression and auto-term spreading. In the time-lag plane, the sampled form is

$$\psi(t,k) = \frac{1}{2^{|k|}} \sum_{i=0}^{|k|} \binom{|k|}{i} \delta\left(t + \frac{|k|}{2} - i\right), \tag{4.15}$$

and the ambiguity plane sampled form is

Table 4.1: Comparison of Alias-Free Methods

Comparison Criteria	Methods		
	Continuous Kernel	**Interpolation**	**Phase Shift**
Requires Continuous Form of Kernel in Time-Lag Domain	**Yes**	**No**	**No**
Accuracy	**Generates the correct result for the linear convolution if aliasing is avoided**	**Suffers from low pass filter effects which blur features in Time-Frequency plane**	**Encounters some error (very minor) in calculating the equivalent of the phase shifted linear convolution**
Cost of Computing	**O(N^2 *log N*)**	**O(N^2 *log N*) process that requires additional computation for interpolation (e.g. linear interpolation requires an additional O(N^2) computations)**	**O(N^2 *log N*)**
Truly Alias-Free?	**Yes, except for Discrete Wigner Distribution.**	**No. Aliasing is reduced but dependent upon interpolation method.**	**Yes**
Ambiguity Kernel Yields Same Distribution as Time-Lag Kernel	**No**	**Yes**	**Yes**

$$\phi(n,k) = \cos^{|k|}\left(\frac{\pi n}{N}\right) \qquad n = -\frac{N}{2}, \ldots, \left(\frac{N}{2}-1\right). \tag{4.16}$$

Figure 4.6 shows the GDTFD generated using the Continuous Kernel, Interpolation and Phase Shift methods.

As a final example, consider the case of the Wigner distribution. Suppose the three methods were applied to the Wigner distribution. Is it possible to generate an alias-free Wigner distribution? The answer is yes, but only for the Interpolation and Phase Shift methods. The CK method produces a standard *aliased* Discrete Wigner Distribution (DWD). This is the result of the hexagonal sampling yielding only zero terms for the odd

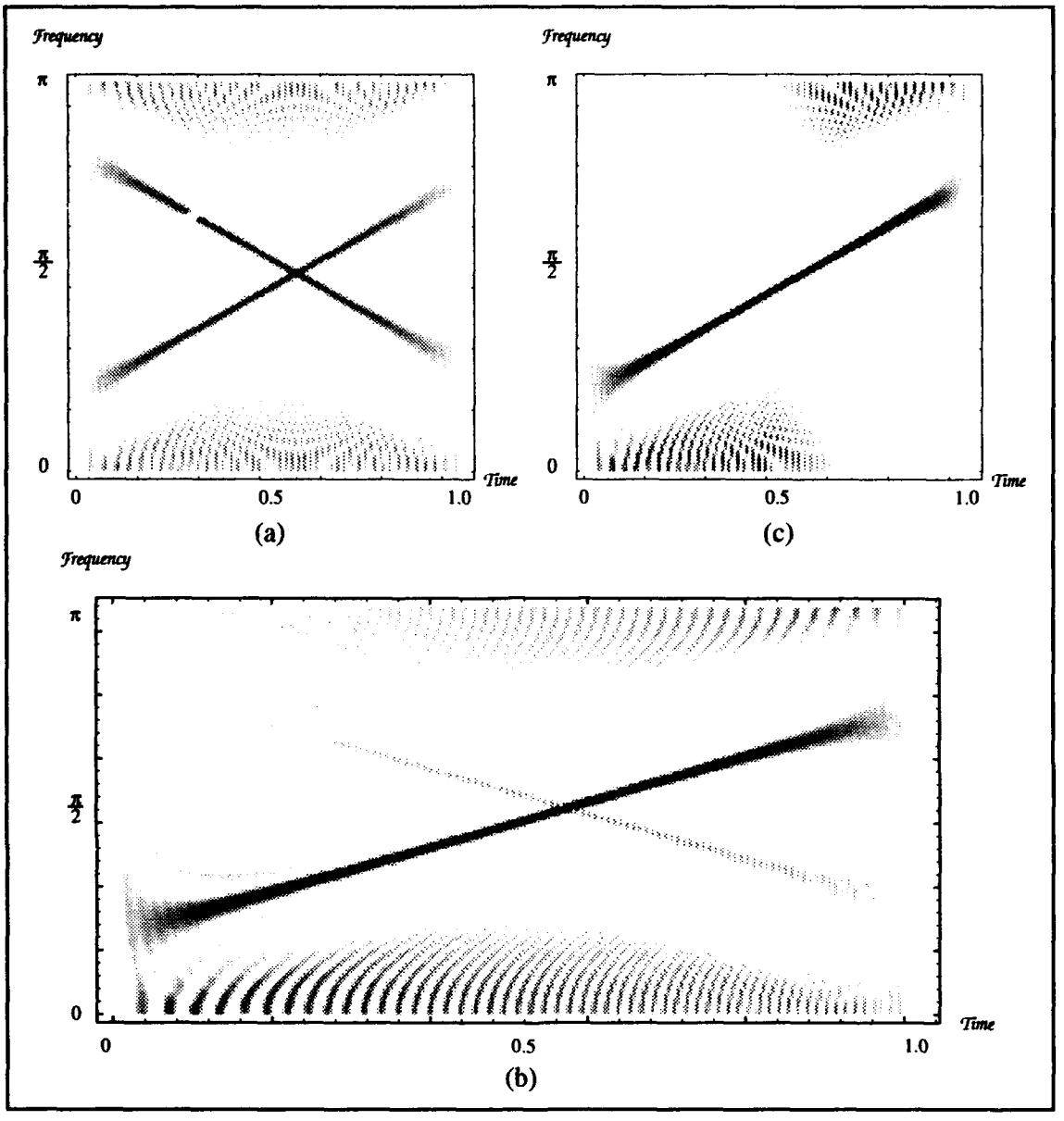


Figure 4.7. GDTFD Wigner of a Simple Chirp via (a) the Continuous Kernel (not Alias-Free) Method (b) the Interpolation Method and (c) the Phase Shift Method.

rows of the kernel; hence, the result of the convolution has non-zero values only at every other row. (The same as if each column is up-sampled.) Thus, the result is expected for an up-sampled signal when the Fourier transform of the columns is taken—aliasing. (See Figure 4.7.) In [29], Jeong and Williams propose a solution to this problem by defining the alias-free DWD (AF-DWD) with the kernel

$$\psi(t,k) = \begin{cases} \delta(t) & k \text{ even} \\ \frac{\delta}{2}\left(t+\frac{1}{2}\right)+\frac{\delta}{2}\left(t-\frac{1}{2}\right) & k \text{ odd.} \end{cases} \tag{4.17}$$

However, this is only an approximation to a alias-free Wigner. It imposes a two point moving average filter, but that is not a Wigner. In fact, the AF-DWD using this kernel does contain aliasing which can be seen as a small ridge running from the upper left corner toward the lower right. Cross-terms are evident along the top and bottom of the figure. The cross-terms in the upper right and lower left corners are the result of the interaction of the positive (0 to π) frequencies and the negative ($-\pi$ to 0) frequencies. The cross-terms in the upper left and lower right corners are the result of the interactions of the positive and negative frequency aliasing terms. The curved ridges seen in the center are cross-terms which result from the rectangular window which was applied to the chirp prior to calculating the distribution. (See Figure 4.8a.)

Consider the Phase Shift method. Define the AF-DWD by (4.10) where $\phi(n,k) = 1$, then the corresponding kernel in the time-lag domain is defined by

$$\psi(t,k) = \begin{cases} \delta(t) & k \text{ even} \\ \sum_{n=-N/2}^{N/2-1} e^{-j\pi n(2t-1)/N} & k \text{ odd.} \end{cases} \tag{4.18}$$

Figure 4.9 shows the impulse response of the odd rows for the case $N = 128$. As can be seen, its central peak is similar to the kernel in (4.17); however, other than that, all similarity is lost. Significantly, the AF-DWD calculated this way exhibits no aliasing. (See Figure 4.8b.)

The phase shift method produces a truly AF-DWD. It is a consistent method between odd and even rows. Furthermore, it generates the desired delta function for the even rows.

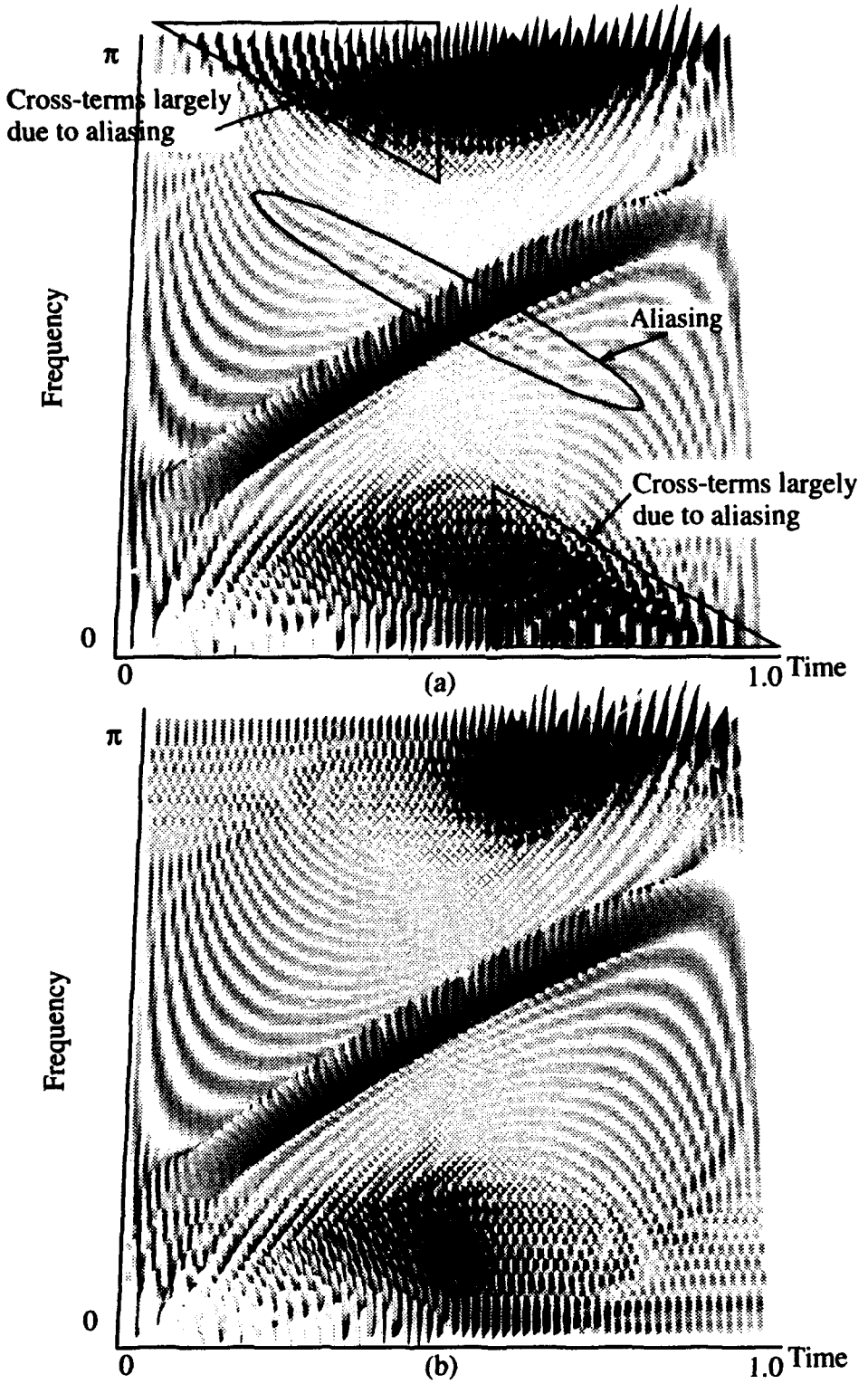


Figure 4.8. The Alias-Free Discrete Wigner Distribution of a Simple Chirp Using the Kernel Given by Jeong and Williams (a) and by the Ph ift Method (b).

As a result, it is claimed that this is a more accurate description of the kernel for the alias-free DWD and should be used rather than (4.17).

In the interpolation method, aliasing has not been completely avoided, but is present as a side-effect of the interpolation algorithm used; however, the peak magnitude of the aliasing terms are between 10 and 20 dB down relative to the primary signal.

The phase shift method produces an AF-DWD, but it does have cross terms present. The cross terms are seen in the lower left and upper right corners of Figure 4.8b and are a result of the interaction of the positive and negative frequencies (only the positive frequency is shown). These are expected when dealing with a real signal. The curved ridges in the center are also expected and are the result of the rectangular window which was applied to the chirp prior to calculating the AF-DWD.

4.6. Conclusion

In this chapter, a method was presented to reduce the computations necessary to calculate the GDTFD from $O(N^3)$ to $O(N^2 \log N)$ by using circular rather than linear convolution via FFT. Three methods were presented for this algorithm to move kernels between the ambiguity and time-lag domains.

The Continuous Kernel method is the most accurate of the three methods in determining the outcome of the interpolatic ever, it has the major drawback of requiring the continuous form of the kernel in the time-lag plane. Also, since the sampled $\psi(t,\tau)$ is not necessarily the DFT of the sampled $\phi(\theta,\tau)$, kernels may inadvertently be changed. In addition, it is often impractical or impossible to obtain the continuous form of the kernel. The CK method, therefore, has more restricted application than the other methods.

The Interpolation method has problems in regard to loss of localization in time as well as imposing extra low pass filtering along the frequency axis. Also, it requires an additional $O(N^2)$ computations as a minimum to implement. Selection of the interpolation algorithm used has a significant impact on the performance of the Interpolation method. In

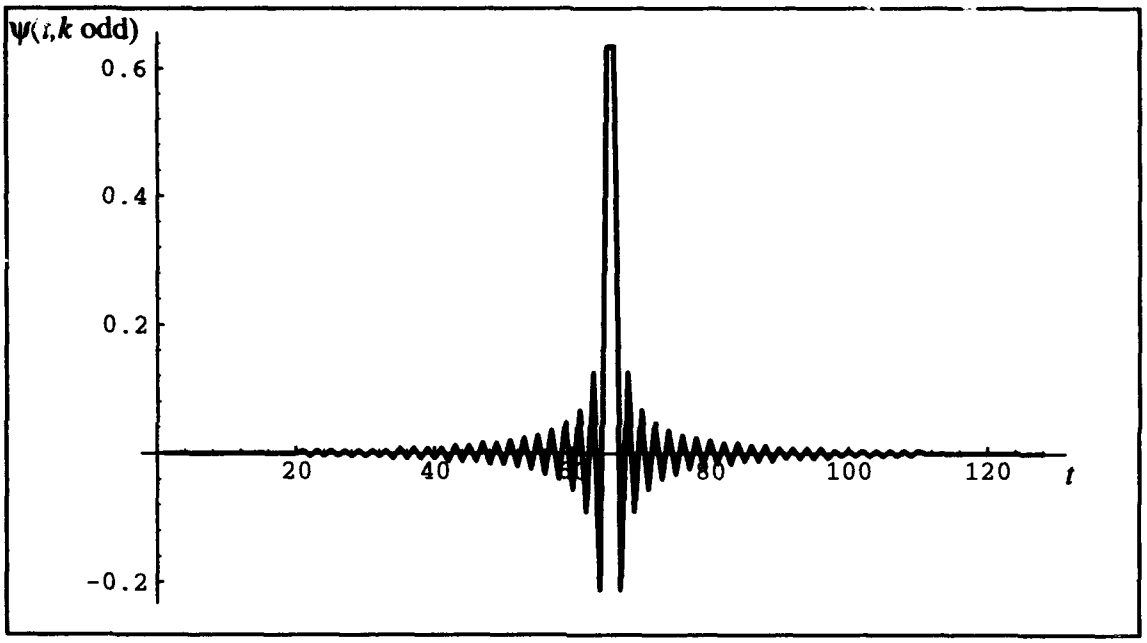


Figure 4.9. The Value of the Odd Rows of the Time-Lag Kernel of Alias-Free DWD (i.e. the Impulse Response of the Filter Implemented by the Odd Rows of the Time-Lag Kernel.

this chapter, the results are based on a linear interpolation of the four nearest neighbors. While this is a fast interpolation method, it does introduce some aliasing into the final TFD.

On the positive side, the Interpolation method does give a TFD which is largely alias-free up to the Nyquist frequency (meaning the aliasing effects are attenuated), and it has twice the resolution in time as the other two methods. It also has the strength of not needing the continuous form of the kernel in either the ambiguity or time-lag planes.

The most robust of the three methods is the Phase Shift method. It yields a reasonably accurate estimate of the shifted $\phi(n,k)$, providing a TFD very similar to that produced by the CK method. Because it does not suffer the problems associated with the Interpolation method and does not require the continuous form of the kernel in the ambiguity or time-lag planes, this is the most versatile of the three methods for calculating the alias-free form of a kernel designed in the ambiguity plane. It even allows the calculation of an alias-free Wigner distribution.

With the tools developed in this chapter it is now possible to easily use a kernel developed in the ambiguity plane or in the time-lag plane and still have an alias-free kernel. In the following chapters, fixed kernels defined in the time-lag plane are used exclusively to develop first decimated Time-Frequency Distribution in Chapter 5 and finally multirate Time-Frequency Distributions in Chapter 6, but these developments are predicated on the fact that it is now possible to easily obtain the alias-free form of a kernel in the time-lag domain no matter where the kernel was designed.

5. The Zak Transform and Decimated Time-Frequency Distributions

5.1. Introduction

A new Decimated Generalized Discrete Time-Frequency Distribution is developed using the kernel design techniques of Chapter 4 combined with the Zak transform and Weighted Spectrograms. These resulting decimated distributions will be used as building blocks to create MRTFD in Chapter 6.

In this chapter, it is shown that the discrete Zak transform is, with slight modification, a generalization of the STFT. The modification made to the Zak transform creates a new transform called the Windowed Zak Transform (WZT). The squared magnitude of the WZT is a generalization of the spectrogram which is called the Zak-Spectrogram (ZS). The connection between the Zak transform and the STFT makes it possible to create fast spectrograms which trade bandwidth for speed while maintaining the same frequency resolution. The link between the spectrogram and ZS hints at a more interesting, new, result—D-GDTFD. A D-GDTFD generates a distribution which has the same trade-offs seen between the ZS and the spectrogram (i.e. bandwidth for speed). Implementation of the D-GDTFD is done via a modification of the weighted spectrograms method.

The idea of D-GDTFD and its companion multirate implementation builds upon several different disciplines. It ties together a numerical method for calculating Gabor transforms (the Zak transform), Time-Frequency Distributions, Weighted Spectrograms and Multirate. Figure 5.1 shows a block diagram of the interrelationships between these fields and how they are combined to achieve a multirate D-GDTFD.

This chapter is divided into five sections. First, background material on the Windowed Zak transform, spectrogram, weighted spectrograms and Time-Frequency Distributions is given. Next, the Windowed Zak Transform is used to develop a new generalization

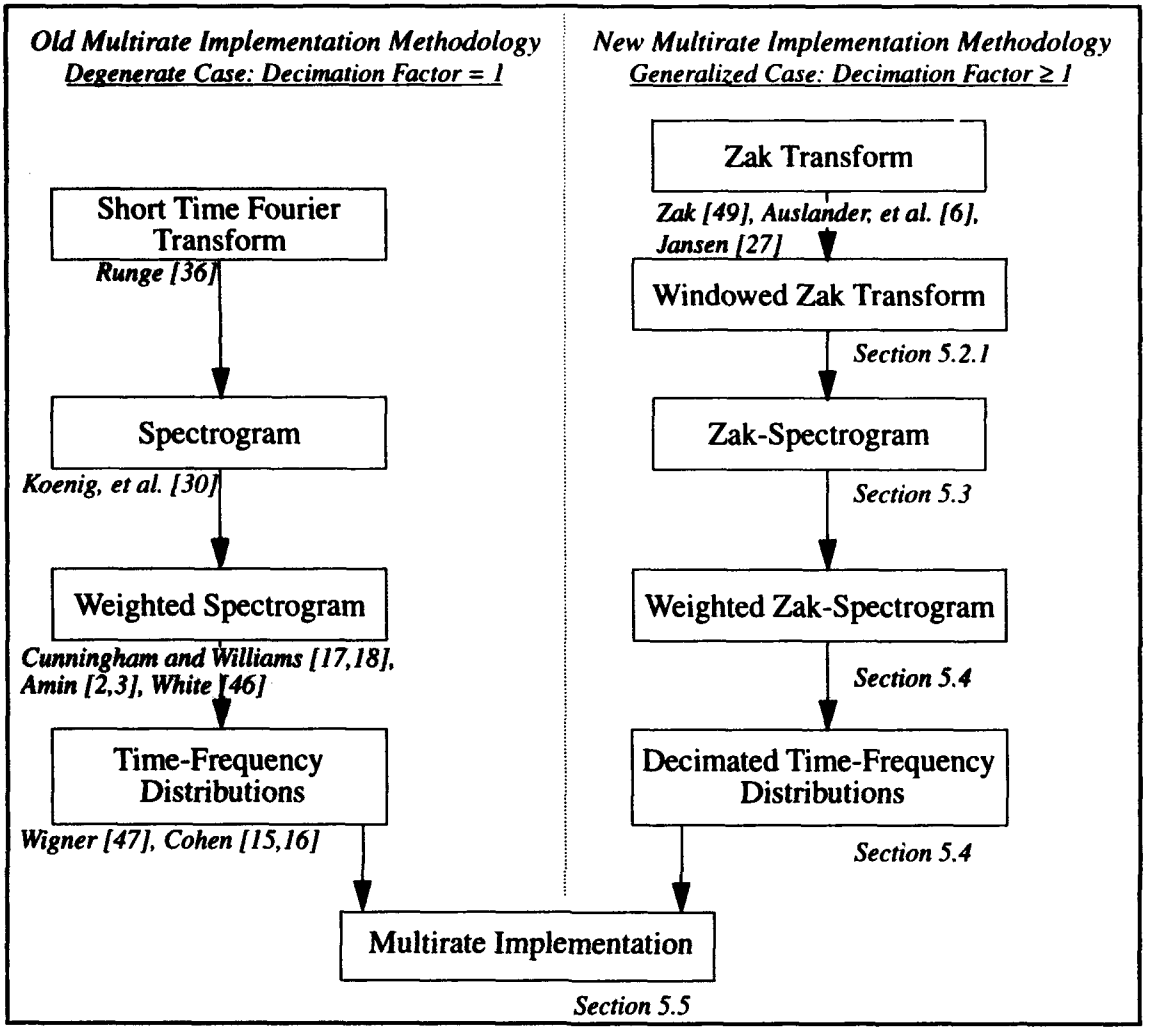


Figure 5.1. Interrelation of Logical Elements Discussed in This Chapter.

of the spectrogram, called the ZS. Then, arbitrary time-frequency distributions are characterized and extended using ZS's. After this, a possible multirate implementation of the D-GDTFD is presented. In the final section, an example is given to demonstrate the relationship between GDTFD, weighted spectrograms and the D-GDTFD.

5.2. Background

5.2.1. Windowed Zak Transform. The Zak transform was originally developed in the field of solid state physics [57]. Since that time it has been used to solve various problems in mathematics and physics. In the area of signal processing, the Zak transform has

been used of determining coefficients in Gabor transforms [6], but the physical meaning of the resulting coefficients have been called into question [41].

The Zak transform has been largely overlooked by the Time-Frequency Distribution (TFD) community, except for passing remarks that it is possible to produce a Wigner distribution (WD) using the Zak [28]; however, the problems associated with creating the discrete WD (DWD)—such as finite window effects, non-rectangular sampling of bilinear signal and aliasing (see, for example, Chapter 4)—were not addressed.

In this section, the definition of the Zak transform given in Chapter 2 is expanded. The existing definition assumes a rectangularly shaped window. A natural extension to this definition of the Zak transform allows for arbitrary windows and is called the Windowed Zak transform. Using the multirate implementation of the STFT given by Vaidyanathan [50], a multirate form of the Windowed Zak transform is presented.

5.2.1.1. Definition. The basic Zak transforms relies on the STFT using a rectangular window. Now, let the definition of the Zak transform to be broadened to include a non-rectangular window.

The definition of the STFT of some signal, f, is [39]

$$T_f(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\, h(\tau - t)\, e^{-j\tau\omega} d\tau = \int_{-\infty}^{\infty} f(\tau + t)\, h(\tau)\, e^{-j\tau\omega} d\tau \tag{5.1}$$

Letting $t = n\rho$, $\omega = k/N$ and $\tau = l\rho$, $l, n \in \mathbf{Z}$, the discrete STFT is

$$T_f(n,k) = \sum_{l=0}^{N-1} f([n+l]\rho)\, h(l\rho)\, e^{-j2\pi lk/N} \tag{5.2}$$

Without much effort, (5.2) can be rewritten as

$$WZ_f(n,k) = \sum_{l=0}^{L-1} f([lm+n]\rho)\, h(lm\rho)\, e^{-j2\pi lk/L} \tag{5.3}$$

where $N = Lm$. As can be seen, (5.3) is a generalization of the Zak transform which now

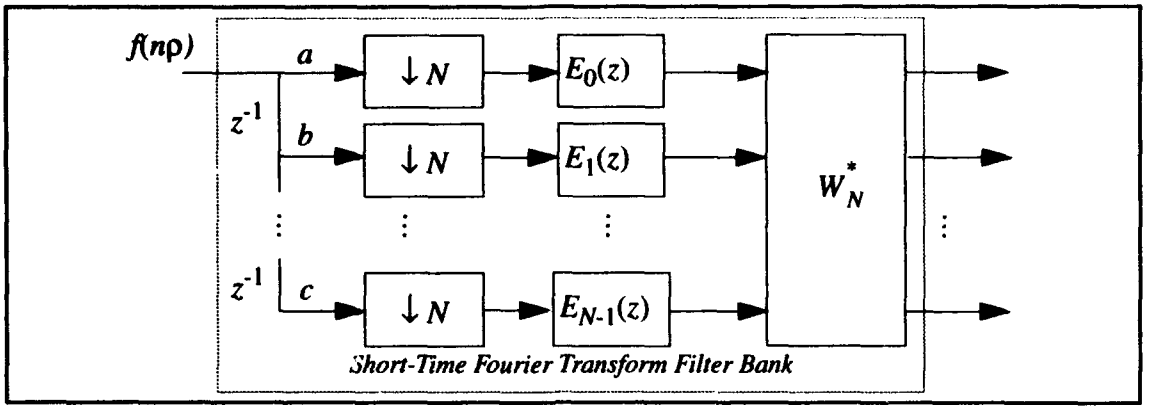


Figure 5.2. Block Diagram of Multirate Implementation of STFT.

includes an arbitrary window; thus, for $h(t) = rect(t)$, $Z_f(n,k) = WZ_f(n,k)$.This formulation is called the Windowed Zak Transform (WZT).

5.2.1.2. Multirate Interpretation. It is possible to consider the WZT in a multirate sense. This has an advantage of providing a direct means of implementation which reduces the required component speed for a given throughput. It provides an alternate interpretation both conceptually and mathematically for analysis of the Zak transform and WZT as nothing more than the result of decimation and filtering operations, and as will be seen ins section 5.4, it can be expanded to handle any discrete Time-Frequency Distribution.

Since both the Zak transform and WZT are based upon the discrete STFT, a multirate implementation of the STFT is examined first. In [50], Vaidyanathan presents a block diagram of the multirate implementation of the STFT (See Figure 5.2.). The multirate STFT takes the input signal, f, passes it through a delay chain such that at time $n = N$, $f(N\rho)$ is at a, $f([N-1]\rho)$ is at b and $f(\rho)$ is at c. The delay chain is followed by a bank of N fold decimators which pass only every N^{th} sample to the filters $E_k(z)$. The filters implement the window function in the STFT. For example, the filter $E_{N-k-1}(z)$ is just a multiplier whose value is $h(k\rho)$. The block W_N^* is the discrete Fourier transform coefficient matrix of dimension $N \times N$; thus, at every N^{th} clock cycle the system will output an N point

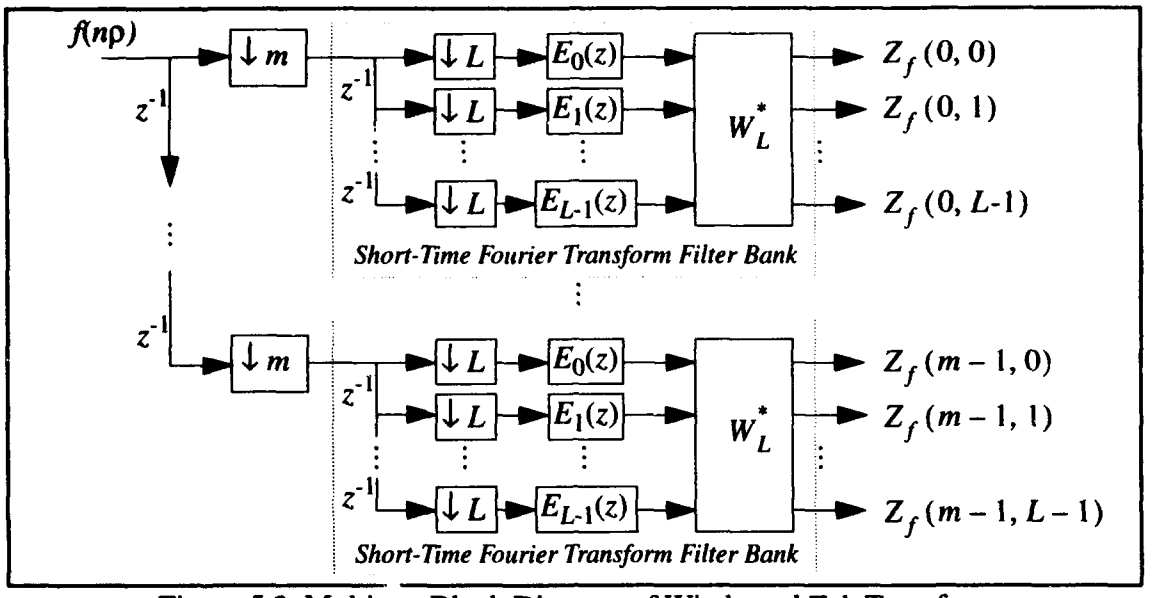


Figure 5.3. Multirate Block Diagram of Windowed Zak Transform.

Fourier transform.

With the system shown in Figure 5.2 as a building block, it is simple to extend the multirate implementation to include the WZ transform. Figure 5.3 shows the block diagram for this implementation. Now, instead of a single STFT filter bank, there is a set of filter banks preceded by an additional delay chain/decimator combination. The filters $E_k(z)$ are equivalent between parallel STFT filter banks as are the coefficients in the matrix W_L^*. Thus, a highly parallel multirate implementation of the Windowed Zak transform can be formed. At every N^{th} clock cycle this system will output an $m \times L$ WZ transform (i.e. m WZ transforms which are L long).

5.2.2. Spectrogram. In this section, the spectrogram is discussed. The spectrogram can be calculated in numerous ways. Two are discussed here. The first method for calculating the spectrogram is via Cohen's class of TFD [15]. It can be regarded as an outer product formulation and is discussed in 5.2.2.1. The second method takes magnitude square of the values of the STFT. This can be thought of as an inner product formulation, and it is discussed in 5.2.2.2. In section 5.2.2.3, a means of calculating GDTFD's based upon a

sum of weighted spectrograms is given. This is called the method of Weighted Spectrograms.

5.2.2.1. Outer Product Formulation. The spectrogram has been shown to be a time-frequency representation by Classen and Mecklenbräuker in [15]. In Appendix A, it is shown that in order for (2.2) (the outer product formulation) and the squared magnitude of the STFT to produce the same result, the time-lag kernel and the window in the STFT must be based upon a window which is real symmetric and/or imaginary anti-symmetric. In other words, the TFD can be written in the form seen in (2.2) where the kernel is defined as

$$\psi(u,\tau) = h\left(u+\frac{\tau}{2}\right)h^*\left(u-\frac{\tau}{2}\right) \tag{5.4}$$

and $h(t)$ is a real symmetric and/or imaginary anti-symmetric window function. This means ψ is equivalent to a Hermitian operator and the resulting distribution will be real.

5.2.2.2. Inner Product Formulation. The inner product formulation as shown in section 2.3.2 can be derived from the outer product by means of a double change of variables. When the kernel is of the form seen in (5.4), (2.2) can be rewritten as

$$T_f(t,\omega)\,T_f^*(t,\omega) = \left|\int_{-\infty}^{\infty} f(\tau+t)\,h(\tau)\,e^{-j\tau\omega}d\tau\right|^2 \tag{5.5}$$

which has the discrete form

$$T_f(n,k)\,T_f^*(n,k) = \left|\sum_{l=0}^{N-1} f(l+n)\,h(l)\,e^{-j2\pi lk/N}\right|^2. \tag{5.6}$$

One advantage of the inner product form over the outer product form is seen in the discrete case. The outer product form requires the kernel to be sampled on hexagonally decimated sampling grid to avoid aliasing [29]. This can cause some difficulties when designing kernels as was shown in Chapter 4; however, the inner product form is naturally alias-free. The sample points in $\tilde{\psi}(t_1, t_2)$, where t_1 and t_2 are integers, correspond to the

points in ψ when it has been sampled on the hexagonally decimated grid. Pictorially, this can be seen in Figure 2.3.

5.2.2.3. Arbitrary Time-Frequency Distributions Using Weighted Spectrograms. A generalized discrete TFD can be generated by calculating the sum of weighted spectrograms. The weighting factors are simply the eigenvalues of the kernel $\tilde{\psi}$, and the window functions in the spectrogram is the eigenvector corresponding to the eigenvalue [18]. In other words, (2.7) can be rewritten as

$$C_f(t, \omega; \psi) = \sum_{k=1}^{N} \lambda_k \left| \int_{-\infty}^{\infty} f(t+t_1)\, x_k^*(t_1)\, e^{-j2\pi\omega(t+t_1)}\, dt_1 \right|^2 \tag{5.7}$$

where λ_k is the k^{th} non-zero eigenvalue and x_k is the k^{th} eigenvector of $\tilde{\psi}$. With this formulation, any GDTFD can be written as the sum of weighted spectrograms.

5.3. Zak-Spectrogram

This section develops the ZS. First the ZS is defined in 5.3.1, and its connection to Time-Frequency Distributions is given in 5.3.2. Lastly, a multirate implementation is given for calculating the ZS in 5.3.3.

5.3.1. Definition and Properties. In [28], Jansen describes an ambiguity function based upon the continuous Zak transform. This work was expanded by Auslander, *et al.*, in [6] to include the finite discrete Zak transform. Auslander, *et al.*, dealt with the problem of calculating Gabor transforms and as such they consider only the cross-Zak transform. A study of the Zak transform and its relationship to TFD's has not previously been performed. It should be noted that Auslander's result [6] for the cross-ambiguity function and the work done by Jansen [28] lead to the supposition that $WZ_f(n,k)\, WZ_f^*(n,k)$ would result in a TFD.

If, in (5.3), $m = 1$, then $WZ_f(n,k)\, WZ_f^*(n,k)$ is nothing more than the discrete spectrogram. For $m > 1$, (5.3) becomes

$$WZ_f(n,k)\,WZ_f^*(n,k) = \left|\sum_{l=0}^{L-1} f[(lm+n)\rho]\,h[lm\rho]\,e^{-j2\pi lk/L}\right|^2 \tag{5.8}$$

which can be interpreted as a decimated spectrogram which will be called the ZS.

At each time, n, the ZS is related to the spectrogram by the relation of the STFT of a decimated and non-decimated signal. Since the STFT is being used, it is known that each filter output has a passband governed by the relation, 1/(sampling period × number of samples). In the case of the spectrogram, the sampling period is ρ and the number of samples is N. In the ZS, the sampling period is ρm and the number of samples is L; thus, for both the spectrogram and the ZS, the passband of the individual filters is exactly the same, $1/(\rho Lm) = 1/(\rho N)$. The decimation factor, m, instead of changing the passband of the filters, changes the bandwidth of the transform. The spectrogram has a bandwidth of $1/2\rho$ while the ZS has a bandwidth of $1/2\rho m$.

This is the fundamental relationship between the ZS and the spectrogram. Bandwidth is being traded for speed without reducing resolution.

5.3.2. Time-Frequency Interpretation. From Appendix A, it is known that for h symmetric or anti-symmetric, there is a corresponding outer product form of (5.8) which is given by

$$WZ_f(n,k)\,WZ_f^*(n,k) = \sum_{l=0}^{L-1}\sum_{u} f\left[\left(u+\frac{lm}{2}\right)\rho\right] f^*\left[\left(u-\frac{lm}{2}\right)\rho\right]$$

$$\times\, h\left[\left(u-n+\frac{lm}{2}\right)\rho\right] h^*\left[\left(u-n-\frac{lm}{2}\right)\rho\right] e^{-j2\pi lk/L} \tag{5.9}$$

This is nothing more than a GDTFD with a kernel defined as

$$\psi_h(n,l) = h\left[\left(n+\frac{lm}{2}\right)\rho\right] h^*\left[\left(n-\frac{lm}{2}\right)\rho\right], \tag{5.10}$$

and a signal decimated by m at each time interval.

There is, therefore, justification for calling the ZS a TFD. Specifically, the ZS is a

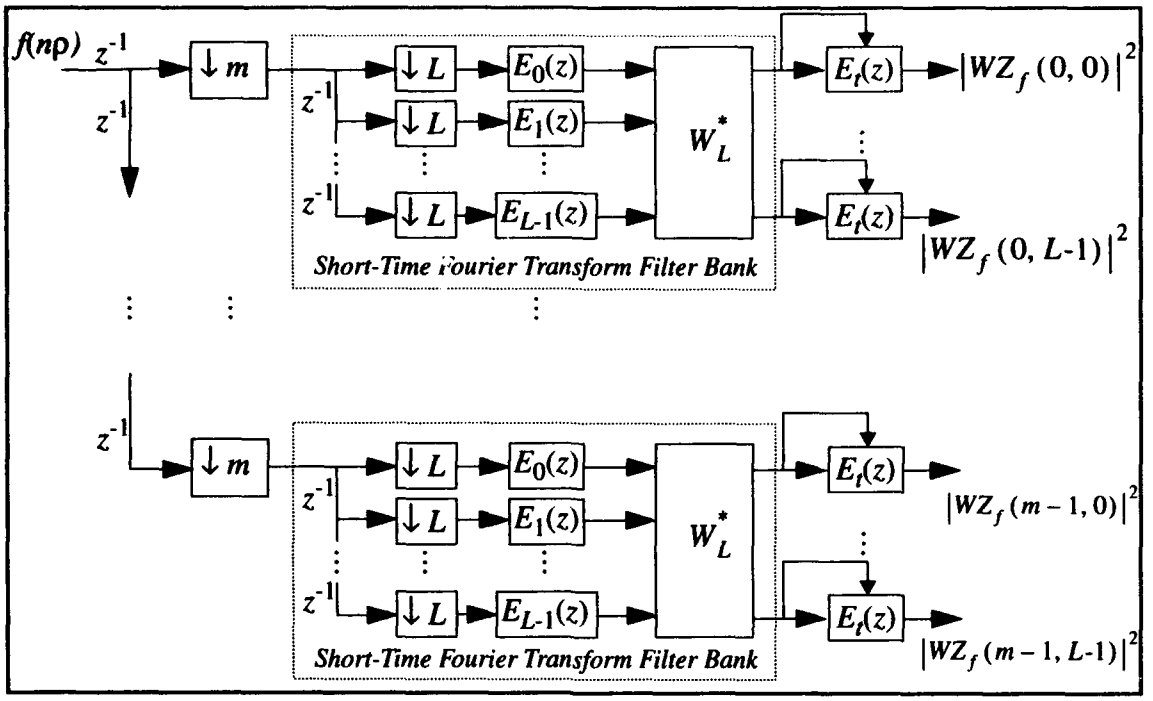


Figure 5.4. Multirate Block Diagram of Zak-Spectrogram

member of a new general class of TFD's which is called the Decimated Time-Frequency Distributions.

5.3.3. Multirate Interpretation. The ZS can be implemented using the multirate techniques discussed earlier. By a simple extension of Figure 5.3, the multirate implementation of the ZS can be obtained. The modification adds a time varying filter at the output of the WZT. The filter is a one point filter whose coefficient is dependent upon the input such that the real input equals the real coefficient and the imaginary input equals the negative of the imaginary coefficient. The output at each time, n, is then the ZS. See Figure 5.4.

Using this multirate implementation reduces the required speed of the components of the system by a factor of m less than a multirate spectrogram. Further it reduces the storage requirement for the coefficients from N filter coefficient to L and the number of complex exponential coefficients in the discrete Fourier transform from $N \times N$ to $L \times L$. Overall, the ZS requires $m^2 + m$ less storage than the spectrogram.

5.4. Generalization of ZS to Arbitrary GDTFD

This section builds upon the idea of the ZS and the weighted spectrogram to produce a Decimated Time-Frequency Distribution. Two ways the ZS can be extended to cover all of Cohen's class using the weighted spectrogram method are by Eigenvalue Decomposition [2][3][18] (covered in 5.4.1) and by Singular Value Decomposition (SVD) (covered in 5.4.2). Both methods can be used to extend the ZS concept to the GDTFD. This extension will be called the Decimated GDTFD (D-GDTFD). The next subsection discusses the similarities and differences between the Eigenvalue and Singular Value Decomposition methods (section 5.4.3).

5.4.1. Extension of Zak-Spectrogram via Eigenvalue Decomposition. The eigenvalues and eigenvectors of the discrete operator, $\tilde{\psi}$, can be used to calculate the GDTFD by means of the discrete form of (5.7),

$$C_f(n,k;\psi) = \sum_{i=1}^{r} \lambda_i \left| \sum_{l=0}^{N-1} f[(l+n)\rho]\, x_i^*(l\rho)\, e^{\frac{-j2\pi kl}{N}} \right|^2 \tag{5.11}$$

where N is the dimension of the operator (i.e. $\tilde{\psi}$ is $N \times N$), $r \le N$, r is the number of non-zero eigenvalues, λ_i is the i^{th} eigenvalue and x_i is the corresponding eigenvector. Letting $N = Lm$ and applying (5.2) to (5.11), the formula for the Decimated GDTFD results,

$$C_f(n,k;\psi) = \sum_{i=1}^{r} \lambda_i \left| \sum_{l=0}^{L-1} f[(lm+n)\rho]\, x_i^*(lm\rho)\, e^{\frac{-j2\pi lk}{L}} \right|^2 . \tag{5.12}$$

This yields a TFD which is decimated by a factor of m and has a digital bandwidth of $\pm\pi/m$ vice $\pm\pi$ of the normal alias-free GDTFD.

5.4.2. Extension of Zak-Spectrogram via Singular Value Decomposition. One problem with the Eigenvalue Decomposition method is, in general, eigenvalues and eigenvectors do not provide very much information about the operator. On the other hand, Singular Value Decomposition (SVD) does. The SVD method is based upon that proposed by White [54] and Amin [3] for use in creating approximate kernels. This section extends the

application of the SVD to D-GDTFD. It should also be noted that there has recently been interest in designing and using asymmetric kernels [5] for which the SVD is ideally suited.

A well known property of the SVD is its decomposition of any $L \times m$ rectangular operator into an $L \times L$ orthogonal matrix, U, an $L \times m$ diagonal matrix, Σ, and an $m \times m$ orthogonal matrix, V. The product of these matrices equals the original matrix, i.e. $A = U\Sigma V^{\dagger}$ ($\dagger$ indicates complex conjugate transpose). This may also be written as

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^{\dagger} \tag{5.13}$$

where r is the rank of A, σ_j is the j^{th} singular value corresponding to $[\Sigma_{jj}]$, $u_j \in \Re^L$ is the j^{th} column vector of U and $v_j \in \Re^m$ is the j^{th} column vector of V. (For more information, see, for example, [24].)

Starting with a general form of the inner product operator, $\langle \tilde{\psi} f, f \rangle$, take the SVD and recast the equation as follows

$$\begin{aligned} \langle \tilde{\psi} f, f \rangle &= \langle \sum_{j=1}^{r} \sigma_j u_j v_j^{\dagger} f, f \rangle \\ &= \sum_{j=1}^{r} \sigma_j \langle f, u_j \rangle^* \langle f, v_j \rangle \end{aligned} \tag{5.14}$$

Now, add the time shift and frequency shift operators defined in (2.8).

$$\begin{aligned} &\langle \tilde{\psi} S_{-t} M_{-\omega} f, S_{-t} M_{-\omega} f \rangle \\ &= \sum_{i=1}^{r} \sigma_i \langle f(n+l)\, e^{-j2\pi k(n+l)}, u_i \rangle^* \langle f(n+l)\, e^{-j2\pi k(n+l)}, v_i \rangle \\ &= \sum_{i=1}^{r} \sigma_i \left[\sum_{l=0}^{N-1} f(l)\, v_i^*(l-n)\, e^{-\frac{j2\pi lk}{N}} \right] \left[\sum_{l=0}^{N-1} f(l)\, u_i^*(l-n)\, e^{-\frac{j2\pi lk}{N}} \right]^* \\ &= C_f(n, k; \psi) \end{aligned} \tag{5.15}$$

Equation (5.15) provides a new general formulation for the GDTFD which depends upon

the product of the STFT using two potentially different window functions. In the degenerate case of $u_i = v_i$ or more generally, for the Hermitian kernels discussed in Section 5.4.3, the weighted spectrogram results.

The advantages of the SVD TFD are: (1) it is easy to obtain the condition number, (2) it is easy to create an approximate operator for which the condition number is automatically known and (3) the SVD based TFD opens up a whole new arena of approximate operators which in turn provides insight into the characteristics of the original operators.

Extending the ZS via the SVD TFD is just as easy as it was via the Eigenvalue Decomposition. By performing the same substitution, the D-GDTFD via the SVD TFD is

$$C_f(n,k;\psi) = \sum_{i=1}^{r} \sigma_i \left[\sum_{l=0}^{L-1} f(lm)\, v_i^*(lm-n)\, e^{-\frac{j2\pi lk}{L}} \right] \times \left[\sum_{l=0}^{L-1} f(lm)\, u_i^*(lm-n)\, e^{-\frac{j2\pi lk}{L}} \right]^* \tag{5.16}$$

5.4.3. Relationship of Eigenvalue and Singular Value Decomposition. If the class of kernels is restricted to those which are Hermitian about the lag axis in the time-lag domain, i.e. $\psi(t,\tau) = \psi^*(-t,\tau)$, Singular Value Decomposition becomes Eigenvalue Decomposition. A brief explanation of this important relationship follows.

A kernel which is real symmetric and/or imaginary anti-symmetric about the lag axis has an inner product representation which is Hermitian such that $A = A^\dagger$. For a Hermitian matrix, A, it is known that there exists a unitary operator, Q, (i.e. a complex matrix which has orthonormal column vectors) and a real diagonal operator, Λ, such that $A = Q\Lambda Q^\dagger$ [48]. The diagonal elements of Λ are the eigenvalues (always real) of A and the columns of Q are the corresponding eigenvectors.

A being Hermitian implies $AA^\dagger = A^\dagger A = A^2$. Since A can be rewritten by a similarity transform, $A^2 = Q\Lambda^2 Q^\dagger$. The values Λ^2 are then the eigenvalues of $AA^\dagger$ and $A^\dagger A$. The

eigenvalues of $AA^\dagger$ and $A^\dagger A$ are the square of the singular values, σ_i, of A [24]. From this, it can be seen that for Hermitian operators, $\sigma_i = |\lambda_i|$ is always the result [48]. This implies in this case that all of the information obtainable from the SVD can be interpreted directly from the Eigenvalue Decomposition.

The relationship between the singular values and eigenvalues implies a tie between the columns of the orthonormal operators U and V in the SVD (where $A = U\Sigma V^\dagger$) and the eigenvectors in Q. Let $V = Q$, and define a new diagonal operator, S, such that $s_i = sign(\lambda_i)$. Then, $A = VS\Sigma V^\dagger$. Since VS is also orthonormal and spans the same space as V (which is the same as U since $AA^\dagger = A^\dagger A$), it is possible to define $U = VS$; thus, the Singular Value Decomposition, $A = U\Sigma V^\dagger$ where $\sigma_i = |\lambda_i|$, $V = Q$ and $U = QS$, can be derived from the Eigenvalue Decomposition. Therefore, for kernels real symmetric and/or imaginary antisymmetric about the lag axis (i.e. $\psi(t,\tau) = \psi^*(-t,\tau)$) (5.15) can be simplified to

$$C_f(n,k;\psi) = \sum_{i=1}^{r} s_i \sigma_i \left| \sum_{l=0}^{N-1} f(l)\, v_i^*(l-n)\, e^{-\frac{j2\pi lk}{N}} \right|^2 \qquad (5.17)$$

$C_f(n,k;\psi)$ in (5.17) will always be real-valued; however, this is not necessarily true for $C_f(n,k;\psi)$ in (5.15). It is therefore a sufficient condition for real-valuedness of $C_f(n,k;\psi)$ if the operator is Hermitian. Further, if s_i is restricted to positive values only, the resulting distribution will always be positive [18].

Thus, for the case of a real distribution where the kernel must be Hermitian, the Singular Value Decomposition and the Eigenvalue Decomposition provide the same information.

5.5. Implementation Considerations

In this section, some of the details involved in implementing a decimated TFD are discussed. In subsection 5.5.1. , a decimated kernel is developed which could be used to implement the decimated TFD in the same fashion as any other distribution. In subsection

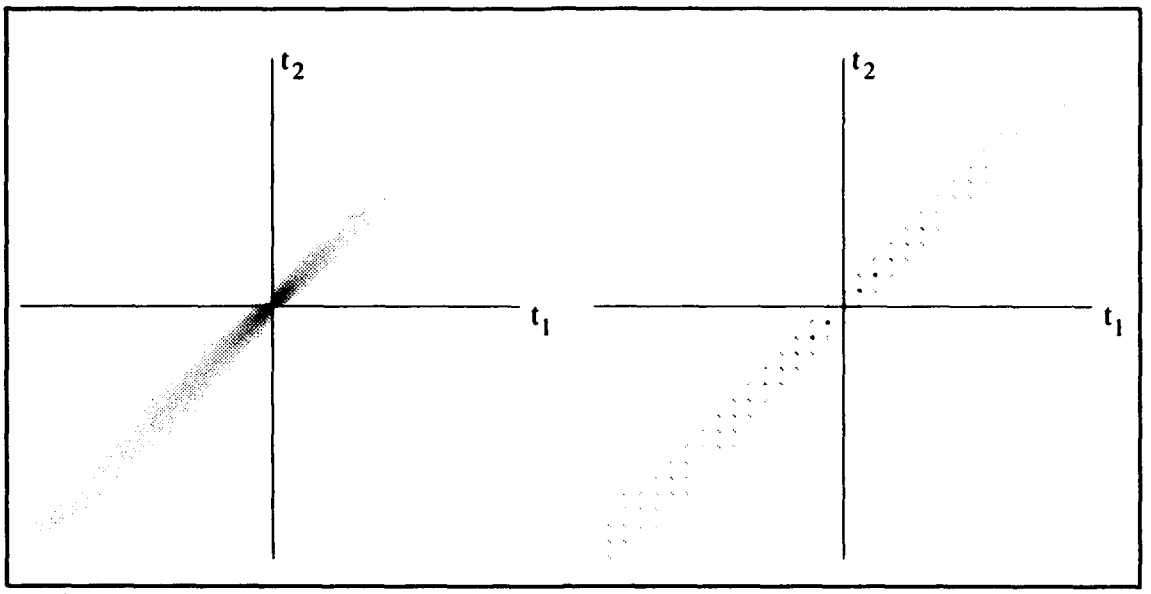


Figure 5.5a. The Original Binomial Kernel

Figure 5.5b. The Decimated Binomial Kernel

5.5.2. , a multirate implementation is proposed which takes advantage of the decimated structure of the ZS using a weighted ZS approach.

5.5.1. Decimated and Non-Decimated Kernels. Consider Equation (5.16). A new kernel can be created using (5.16) via (5.13). The new kernel can be thought of as a decimated and upsampled version of the original kernel. This follows from the decimation performed in (5.16).

There are two ways to look at the kernel created by (5.16). First, a decimated kernel operating on a decimated signal. For example, suppose the level of decimation is four (i.e. $m = 4$). Then, every fourth value in the t_1 and t_2 directions in the kernel is taken and formed into a new kernel. If the original kernel was $N \times N$, the new kernel is $N/4 \times N/4$. This kernel is then applied to the decimated signal which also must be decimated by four and has length $N/4$.

The second way to look at the new kernel is to consider (5.16) as having the window functions v and u which have been decimated and then upsampled. The dimension of the new kernel is the same as the original and the signal need not be decimated. It does cause

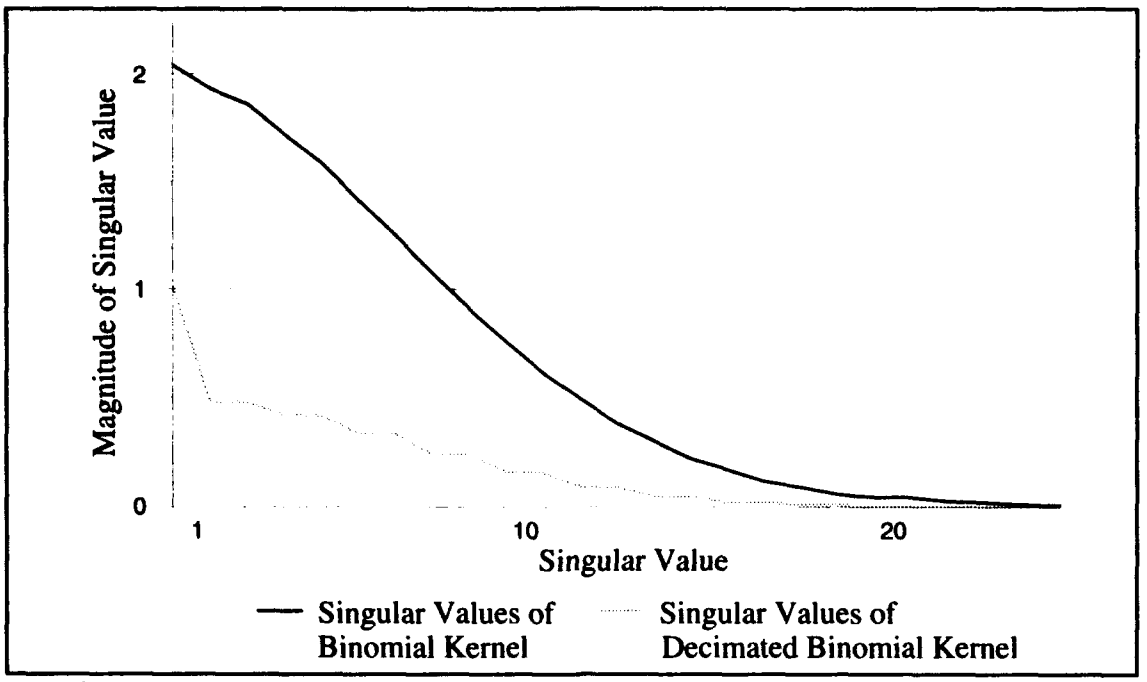


Figure 5.6. The First 25 Singular Values for the Normal Binomial Kernel and the Decimated Binomial Kernel.

the result of (5.16) to be periodic with period $2\pi/m$ rather than 2π, but in practice, the m - 1 periodic images need not be calculated.

By this second method, a new kernel is created which can operate on the same signal as the original kernel, or put another way it operates in the same dimension l_2 space as the original; therefore, this kernel is considered the Decimated GDTFD kernel. In Figure 5.5a, an example of an original kernel is shown. In this case, it is the Binomial kernel [56]. In Figure 5.5b, the new kernel is shown. As can be seen, it is considerably more sparse than the original containing $1/16^{th}$ as many non-zero terms, and it is the Decimated GDTFD kernel.

While the original kernel and the new kernel are obviously related, they are decidedly non-similar (in a mathematical sense), and hence, have different Eigenvalue and Singular Value Decompositions. This point is driven home by an analysis of the singular values of the examples in Figure 6. If the two kernels were similar, the singular values would be the same; however, as Figure 5.6 shows, they are not. The significance of this is that a

method to create a new set of kernels from ones previously defined has been formed. The new kernels are truly different, as evidenced by their different singular values, from their "parent," but they behave the same over the bandwidth $\pm\pi/m$.

5.5.2. Possible Implementation Strategy. One possible way of implementing the D-GDTFD is by means of multirate techniques. By using multirate methods, it is possible to define a strategy which makes use of parallel design to reduce the clock rate of the individual computational elements; thus, for a given clock rate the throughput of the system is significantly higher than standard sequential design techniques. The implementation presented here is based upon (5.16) and the previous multirate examples.

A multirate implementation of the D-GDTFD consists of parallel ZS's with some modification. Each Windowed Zak uses a different set of windows corresponding to the right and left singular vectors. The WZ based on the left singular vector is conjugated and multiplied by the WZ using the right singular vectors and the singular value. The result of each branch is summed to create the final result. For Hermitian operators, the filters $E^R_{i,j}(z) = E^L_{i,j}(z)$ are the values v_{ji} or u_{ji} of the right or left singular vectors, respectively, and σ_i is the singular value multiplied by s_i from (5.17). In other words, for Hermitian operators, the window functions in the STFT are identical and the weights correspond to the eigenvalues. Figure 5.7 presents a block diagram of one possible multirate implementation.

5.6. Example

It is instructive to examine the performance of the D-GDTFD using weighted ZS's and to compare it to the GDTFD created using weighted spectrograms. The weighted spectrogram approach serves as our basis for comparison and represents the current state-of-the-art. It must be emphasized that the spectrogram is a degenerate case of the ZS (i.e. the spectrogram is the ZS with a decimation factor of one). For clarity the ZS with a decimation factor of one is referred to as the spectrogram and as the ZS for decimation factors

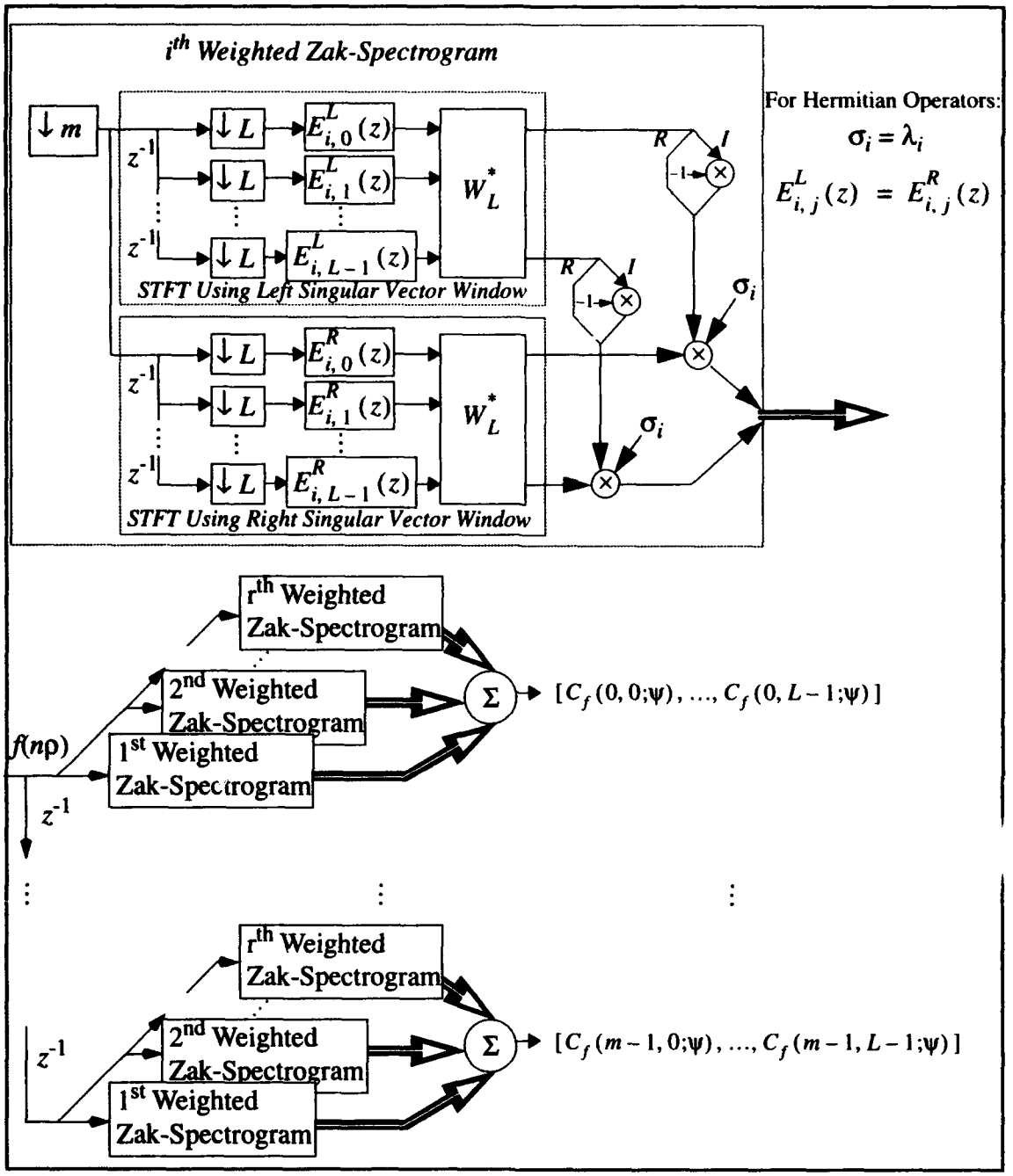


Figure 5.7. Multirate Implementation of Decimated Generalized Discrete Time-Frequency Distribution.

other than one. In this example, a comparison is made between the TFD generated by the weighted spectrogram and TFD created by the weighted ZS with a decimation factor of four.

Four types of distributions will be seen here: the Binomial GDTFD, the approximate

Binomial GDTFD, the decimated Binomial GDTFD and the approximate decimated Binomial GDTFD. The Binomial GDTFD is nothing more than the classical form of the alias-free discrete TFD using a Binomial kernel developed by Williams and Jeong [56]. The approximate Binomial GDTFD is created by first decomposing the Binomial kernel via SVD. Then, the approximate TFD is calculated by means of (5.17) where r is selected to be less than the dimension of the kernel. For example, for $r = 1$, only the spectrogram which is generated by using the singular vector, v_1, as the window function would be used to calculate the approximate distribution. This singular vector is associated with the largest singular value. For $r = 2$, the previous spectrogram multiplied (or weighted) by the largest singular value and the spectrogram created using singular vector v_2 weighted by the second largest singular value are summed to create an improved approximation to the Binomial GDTFD. For successively better approximations, r is increased until σ_{r+1} is zero or r is equal to the dimension of the kernel. At this point, the distribution obtained by the weighted spectrogram method is no longer an approximation but the true distribution.

The decimated Binomial GDTFD is calculated by using

$$C_f(n,k;\psi) = = \sum_{i=1}^{r} s_i \sigma_i \left| \sum_{l=0}^{L-1} f(lm)\, v_i^*(lm-n)\, e^{-\frac{j2\pi lk}{L}} \right|^2 \qquad (5.18)$$

where $m = 4$ and $L = 32$. For the true distribution, r is the dimension of the kernel and $N = Lm = 128$. Equation (5.18) is equivalent to calculating the distribution using the kernel in Figure 5.5b where only the frequencies between $\pm\pi/4$ are calculated. The frequencies above and below are not needed since they are periodic images of those between $\pm\pi/4$. The approximate decimated Binomial GDTFD is calculated in exactly the same fashion as the approximate Binomial GDTFD except (5.18) is used instead of (5.17).

Now, suppose there is a bandlimited signal consisting of a rising chirp and a constant tone where the chirp has twice the amplitude of the tone. Taking the GDTFD of the signal using the Binomial kernel, the resulting TFD is shown in Figure 5.8. This is, in effect, the

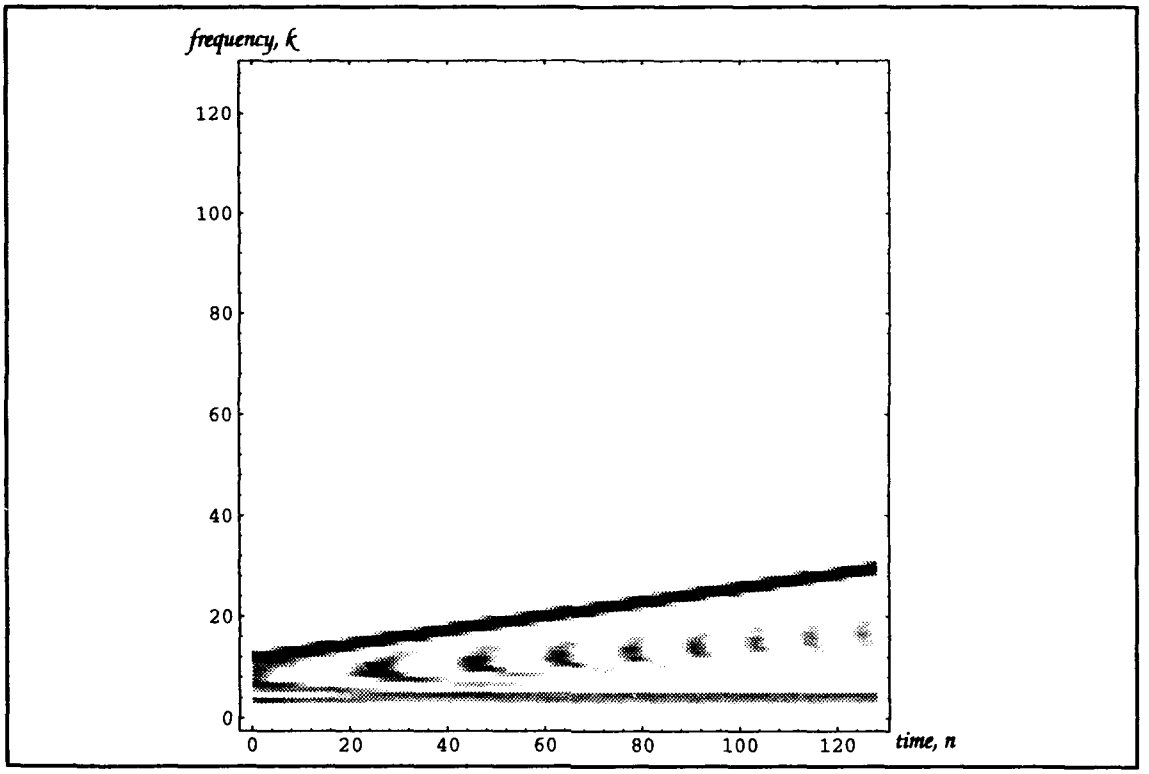


Figure 5.8. Binomial TFD of Bandlimited Signal

"*approved solution*" and is the basis of comparison for this example. The signal is bandlimited as can be seen. What cannot be seen is small cross-terms which reside in the region above $\pi/4$. These terms tend to be less than 10^{-2}.

Next, calculate the approximate Binomial and decimated Binomial GDTFD's. To start, let $r = 1$. Then, the approximate Binomial and decimated Binomial distributions would be the spectrogram and ZS based upon the singular vector v_1 as the window function. The distributions are calculated via (5.17) and (5.18), respectively. The window function is shown in Figure 5.9a. The resulting approximate Binomial distribution is seen in Figure 5.9b and the approximate decimated Binomial distribution is seen in Figure 5.9c.

To create a better approximation, additional weighted spectrograms and weighted ZS's are added. In this example, the spectrograms and ZS's using the singular vectors v_2 and v_3 as the window functions are weighted by their respective singular values (the second and third largest) and added to the previous result (which is weighted by the largest

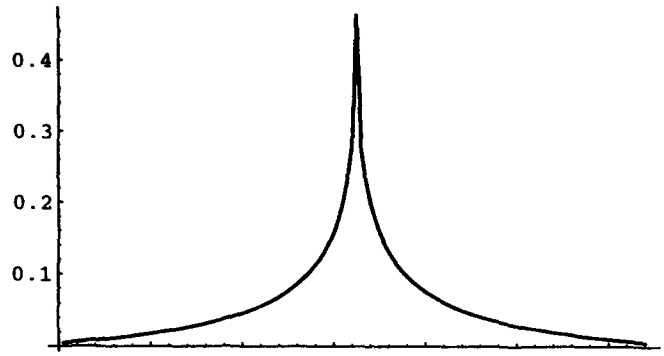


Figure 5.9a. Window Function Corresponding to the Largest Singular Value (i.e. Singular Vector v_1).

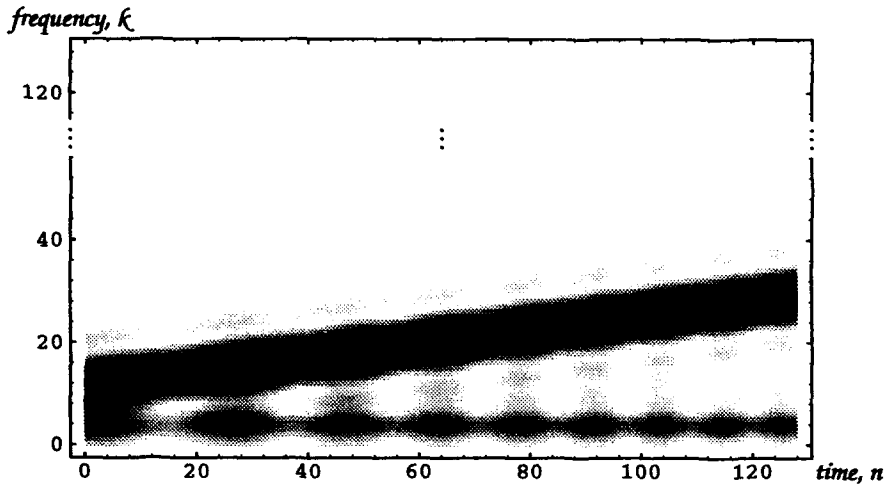


Figure 5.9b. Approximate Binomial GDTFD Using One Weighted Spectrogram with the Singular Vector v_1 Shown in Figure 5.9a as the Window Function.

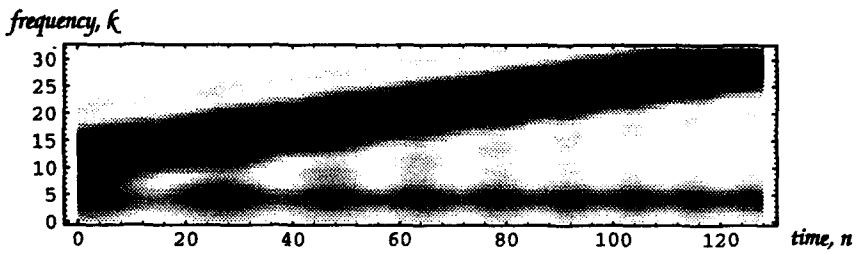


Figure 5.9c. Approximate Decimated Binomial GDTFD Using One Weighted Zak-Spectrogram with the Singular Sector v_1 Shown in Figure 5.9a as the Window Function.

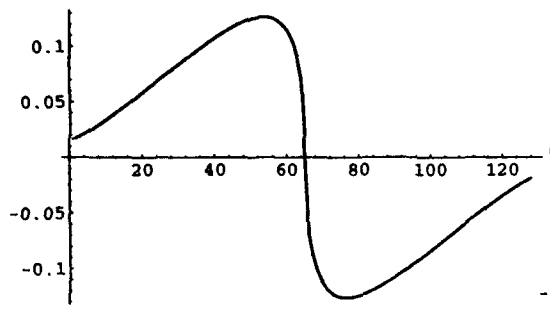


Figure 5.10a. Window Function Corresponding to Second Largest Singular Value.

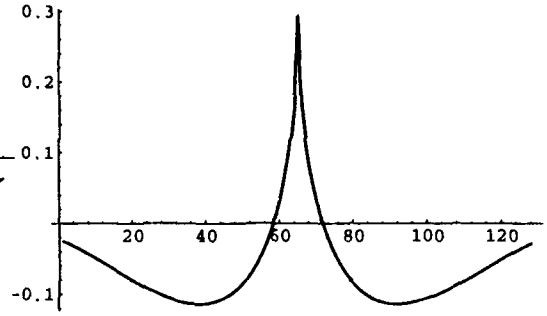


Figure 5.10b. Window Function Corresponding to Third Largest Singular Value.

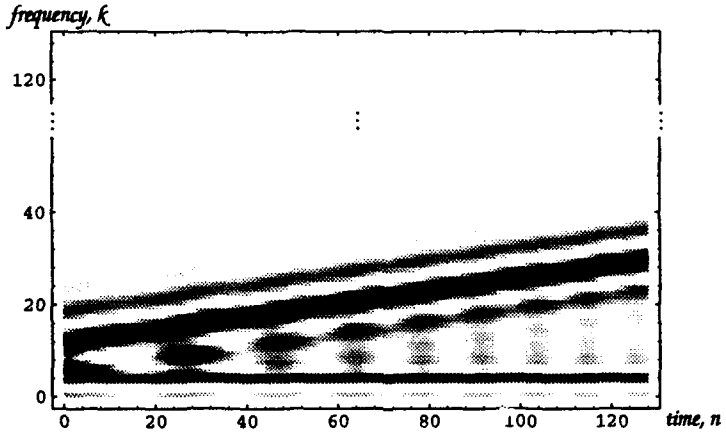


Figure 5.10c. Approximate Binomial GDTFD Using Three Weighted Spectrograms Based upon the Window Functions Seen in Figures 5.9a, 5.10a and 5.10b.

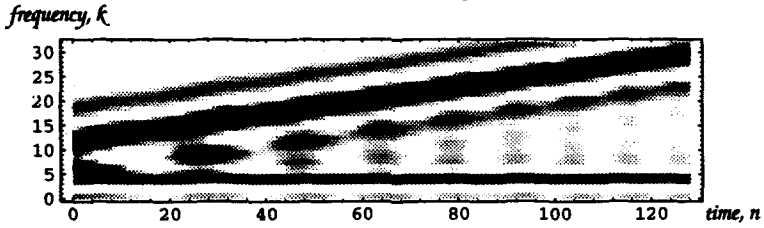


Figure 5.10d. Approximate Decimated Binomial GDTFD Using Three Weighted Zak-Spectrograms Based upon the Window Functions Seen in Figures 5.9a, 5.10a and 5.10b.

singular value). Figures 5.11a and 5.11b show the singular vectors v_2 and v_3, respectively. Figures 5.11c and 5.11d are the approximate Binomial and decimated Binomial distributions. These distributions are the result of r being set equal to three in (5.17) and (5.18).

As the value of r increases the closeness of the approximation will improve in a L_2 sense and in a L_∞ sense as well since the L_2 and L_∞ norms are related by $\|x\|_\infty \leq \|x\|_2$ [24]. Before showing the final approximation in Figure 5.14, it must first be determined how

many weighted spectrograms and weighted ZS's are necessary for a "good" approximation. To determine this, the effect on the error in the approximation of additional weighted spectrograms and weighted ZS's must be examined.

For this example, the instantaneous L_2 error is calculated over all the computed frequencies at a given time, n,

$$\text{Inst } L_2 \text{ error } = L_2(n) = \left(\sum_k [C_f(n,k;\psi) - C_f^r(n,k;\psi)]^2 \right)^{1/2} \tag{5.19}$$

then the average L_2 error over the frame is computed by

$$\text{Avg } L_2 \text{ error } = \frac{1}{128} \sum_{i=1}^{128} L_2(i) \tag{5.20}$$

where the frame is the interval $1 \le n \le 128$, $C_f(n,k;\psi)$ is a particular instant in time of the Binomial GDTFD or the decimated Binomial GDTFD depending upon the comparison being made and $C_f^r(n,k;\psi)$ is a particular instant in time of the r^{th} approximate Binomial or decimated Binomial GDTFD.

The L_∞ error is the maximum pointwise error over the entire frame for all computed frequencies, i.e.

$$L_\infty \text{ error } = \max_{n,k}(\left| C_f(n,k;\psi) - C_f^r(n,k;\psi) \right|) \tag{5.21}$$

Using (5.20) and (5.21), the error between the Binomial GDTFD and the approximate Binomial GDTFD is examined. This represents the baseline error performance for the existing technique. Figure 5.11 shows the average L_2 and L_∞ error for $r = 1,\ldots,40$. It also shows the maximum excursion of the L_2 error over the frame. These are represented as crosses. The top of the cross is the largest L_2 error seen over the frame of 128 instants in time. The bottom of the cross is the smallest L_2 error seen over the frame, and the horizontal bar in the cross is the average of the L_2 error over the frame. The magnitude of these values is read off the left vertical axis. The maximum L_∞ error of all the columns in the frame is plotted as a gray line. The scale for this plot is read off the left vertical axis. As

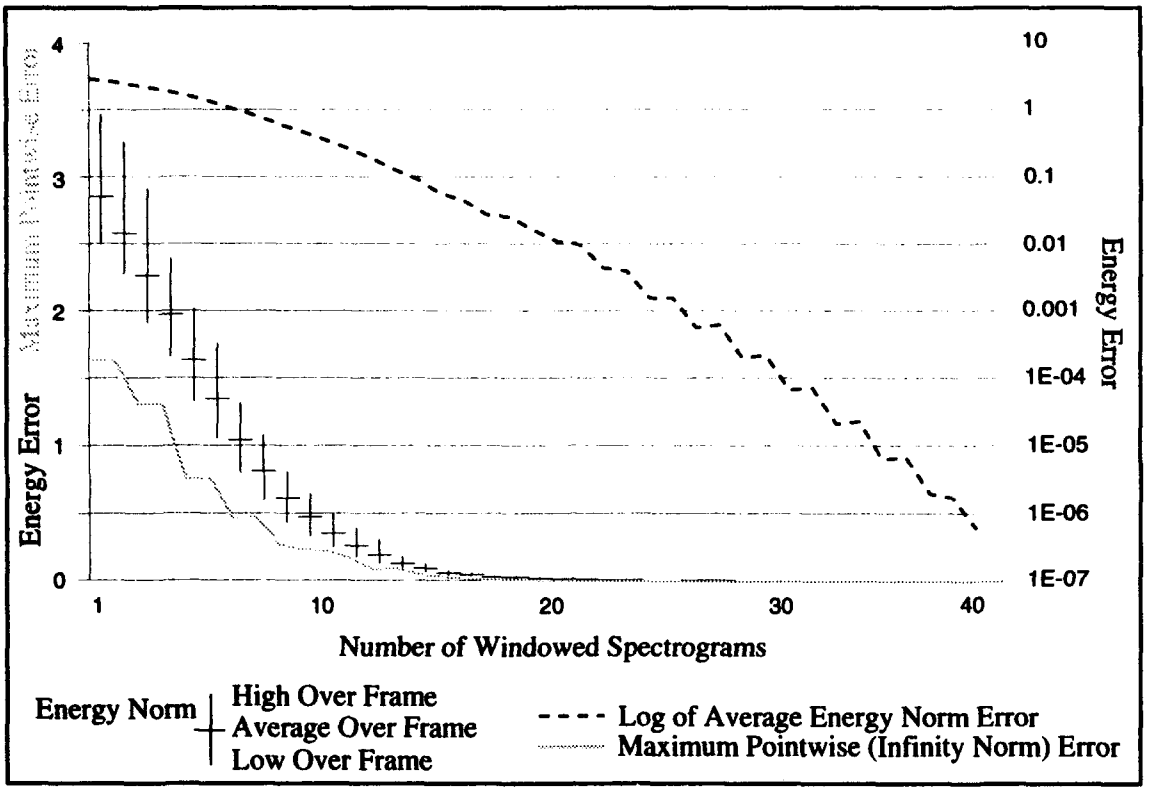


Figure 5.11. The L_2 and L_∞ Error Between the Binomial GDTFD and the Approximate Binomial GDTFD as a Function of the Number of Spectrograms Being Summed.

can be seen, both the L_2 and L_∞ error decrease rapidly as successive weighted spectrograms are added (i.e. as r in (5.17) is increased). The dashed line is the logarithm of the average of the L_2 error of each column of the distribution. The scale of this plot is read off the right vertical axis.

It is expected, based upon an analysis of the kernel in Figure 5.5b, that the error between the decimated Binomial GDTFD and the approximate decimated Binomial GDTFD will perform similarly to that seen in Figure 5.11. Figure 5.12 shows the error of between these distributions. As expected, the error decreases as successive weighted ZS's are added (i.e. as r in (5.17) is increased).

The final error of interest is the error between the Binomial and decimated Binomial GDTFD over the region of mutual support. To examine this, the error between the Binomial GDTFD and the approximate decimated Binomial GDTFD was calculated. The result

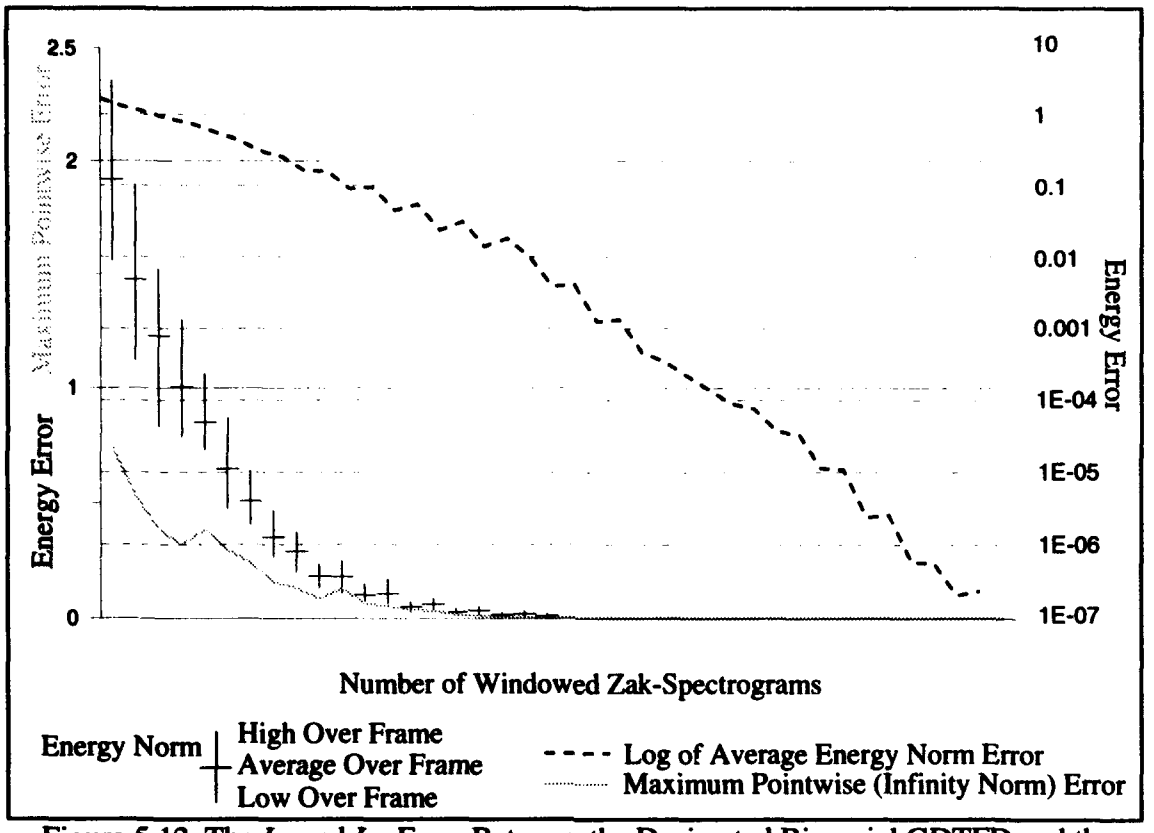


Figure 5.12. The L_2 and L_∞ Error Between the Decimated Binomial GDTFD and the Approximate Decimated Binomial GDTFD as a Function of the Number of Weighted Zak-Spectrograms.

of these calculations is shown in Figure 5.13. While the error between the two does decrease with additional weighted ZS initially, it can be seen that the error approaches a limit. In the figure, only the first 40 weighted ZS's are shown, but there is no significant improvement as additional weighted ZS's are included; thus, the Binomial and decimated Binomial GDTFD, while producing results which are close, do not produce convergent results.

This should not be surprising. As shown in section 5.5.1. , the kernels are related but are not the same. Because the distributions have different bandwidths, the signal components will interact differently. That is, the signal separation, especially between positive and negative frequencies, will be different; hence, the cross-terms will be affected. This will impact the error between the decimated and non-decimated distributions.

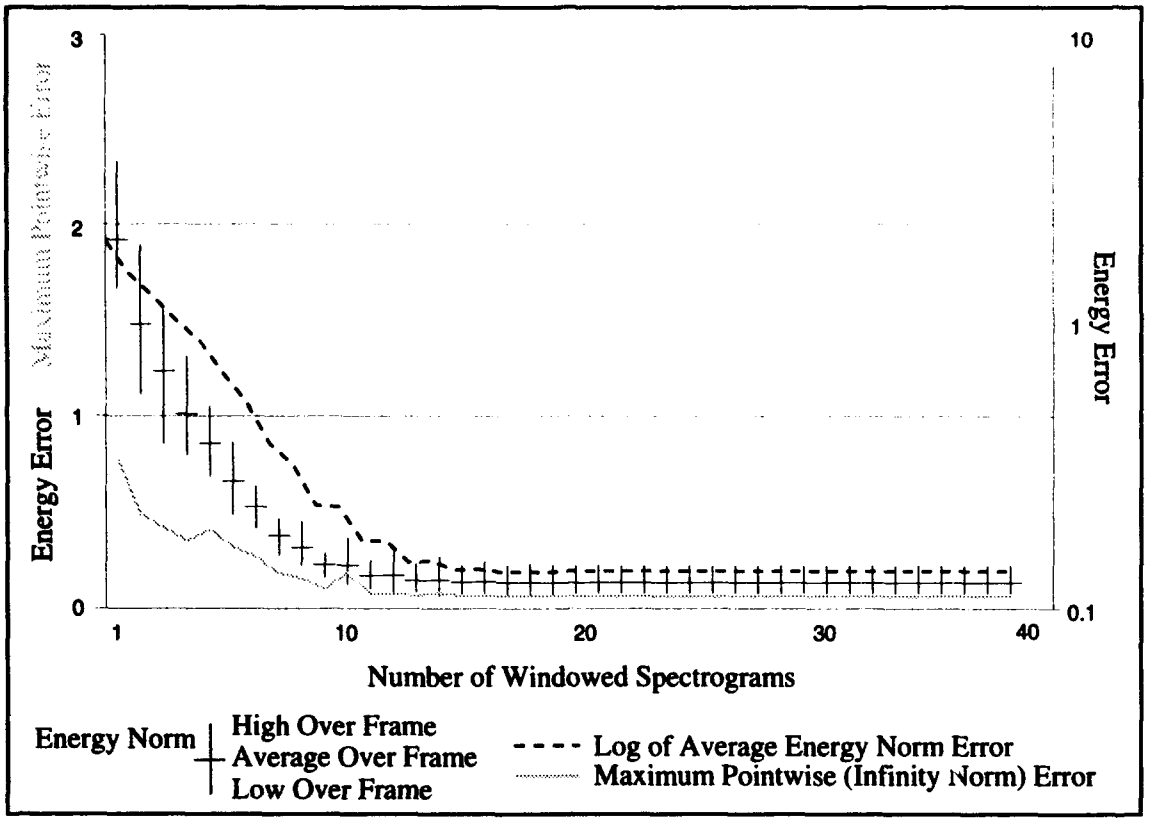


Figure 5.13. The L_2 and L_∞ Error Between the Binomial GDTFD and the Approximate Decimated Binomial GDTFD as a Function of the Number of Weighted Zak-Spectrograms Being Summed.

The beauty of the SVD and Eigenvalue Decomposition methods is there is no need to calculate all of the weighted spectrograms/ZS's. For some error threshold, a suitable approximation can be calculated. For the purposes of this chapter, an arbitrary error threshold of $\sim 10^{-4}$ was selected. This corresponds to somewhere around 30 weighted spectrograms in Figure 5.11 and 20 weighted ZS's in Figure 5.12. Again somewhat arbitrarily, the value of $r = 31$ is selected as the approximation cut-off. It will definitely result in the distributions having errors of less than 10^{-4} since the 32^{nd} singular value is less than 10^{-4}. Figure 5.14a shows the final approximation to the Binomial GDTFD, and Figure 5.14b shows the final approximation to the decimated Binomial GDTFD. The L_∞ error between these two distributions is approximately 0.06.

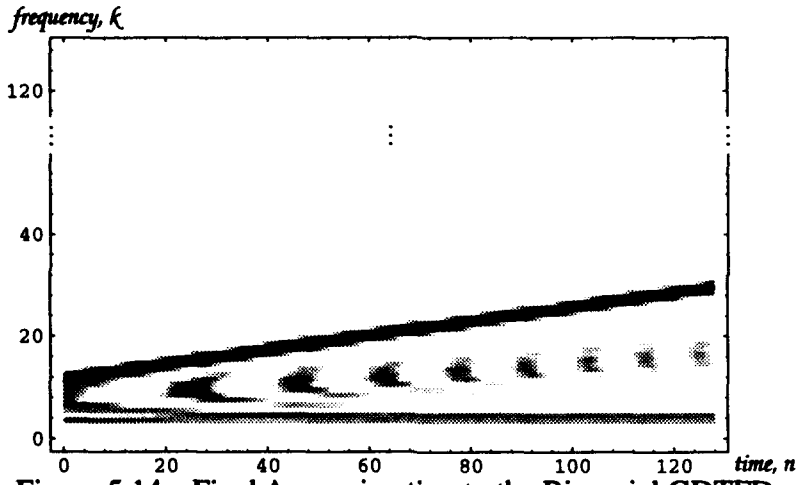


Figure 5.14a. Final Approximation to the Binomial GDTFD Using 31 Weighted Spectrograms.

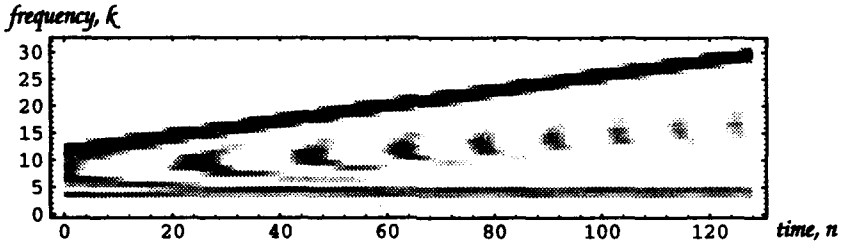


Figure 5.14b. Final Approximation to the Decimated Binomial GDTFD Using 31 Weighted Spectrograms.

5.7. Conclusion

In this chapter, it has been shown that the Zak transform is, with the Windowed Zak modification, a generalization of the Short-Time Fourier Transform (STFT). From this, it was shown that the ZS is a generalization of the spectrogram. The connection between the ZS and the spectrogram makes it possible to create fast spectrograms which trade bandwidth for speed while maintaining the same frequency resolution. The ZS was then used *along with the idea of weighted spectrograms* (via both Eigenvalue Decomposition and SVD) to create Decimated Time-Frequency Distributions.

The power of this formulation lies in its connection with existing multirate techniques and the concept of using small processing building blocks to implement an arbi-

trary TFD. Multirate techniques provide a means to implement the transforms using slower speed devices operating in parallel to achieve the same throughput of standard computational techniques. For a decimation factor of m, there is a m fold increase in throughput (or speed of calculation). The corresponding reduction in discrete bandwidth is from 2π for the GDTFD to $2\pi/m$ for the D-GDTFD. An important attribute of the D-GDTFD is that it requires significantly less storage than the GDTFD. The D-GDTFD requires only $1/m^2$ of the storage of the GDTFD.

Finally, an example based upon the Binomial distribution was given to show how the ZS and spectrogram could be used to produce D-GDTFD and GDTFD respectively. The error between the approximate GDTFD and the GDTFD was examined as was the error between the approximate D-GDTFD and D-GDTFD and the error between the approximate D-GDTFD and the GDTFD.

It has now been shown that a D-GDTFD does exist and can be created from any discrete kernel. Chapter 6 will discuss the final step in creating the MRTFD: recombining D-GDTFD's into a GDTFD.

6. Multirate Time-Frequency Distributions

6.1. Introduction

New Multirate Time-Frequency Distributions (MRTFD) are developed using the multirate computational paradigm (Chapter 3) combined with the techniques to create a discrete kernel in the time-lag domain (Chapter 4) and the ability to decimate a Time-Frequency Distribution (Chapter 5). The first method is based upon using the inner product form of the GDTFD together with the Singular Value Decomposition of the kernel [Singular Value Decomposition Multirate Time-Frequency Distribution (SVD MRTFD) algorithms]; while the second method is based upon the convolutional interpretation of the outer product form of the Generalized Discrete TFD [Circular Convolution Multirate Time-Frequency Distribution (CC MRTFD)].

6.1.1. Baseline—Time-Frequency Algorithms. This section will present the algorithms which will serve as a baseline for the rest o the paper. The first algorithm discussed is the fast TFD algorithm of Cunningham and Williams [19]—the only fast TFD algorithm to appear in the literature. The second algorithm is the straightforward parallel adaptation of the method of Weighted Spectrograms [18].

6.1.1.1. Cunningham and Williams Fast TFD. The fast algorithm of Cunningham and Williams is based upon an approximation of (2.10) in which only the r eigenvectors associated with the largest eigenvalues are used ($r \ll N$). It is given by [19]

$$C_f^r(t, \omega;\psi) = \sum_{k=1}^{r} \lambda_k \left| \int_{-\infty}^{\infty} f(t+t_1)\, x_k^*(t_1)\, e^{-j2\pi\omega(t+t_1)}\, dt_1 \right|^2 . \tag{6.1}$$

The error is bound by the magnitude of the largest eigenvalue not used, i.e.

$$\sup_{t,\omega} \left| C_f^N(t, \omega;\psi) - C_f^r(t, \omega;\psi) \right| \le |\lambda_{r+1}| \|f\|^2 . \tag{6.2}$$

At this point it is necessary to make an assumptions which will be used throughout this paper. The type of FFT used has a direct impact on the number of computations necessary to implement any GDTFD. To be sure that the comparison made herein are on a level playing field, all the algorithms will be assumed to be designed using the same FFT algorithm as a building block. The split radix-2 FFT described by Sorensen, et al, in [46] requires $N \log_2 N - 3N + 4$ multiplies and $3N \log_2 N - 3N + 4$ additions to implement. It represents the state-of-the-art and, as such, it will be the basic FFT building block used. It will also be assumed that all the algorithms will compute $2N$ frequencies at N instants of time. This is done so inner produce methods based upon the Weighted Spectrogram approach, which calculate individual instants of time, can be compared with outer product methods which calculate all N instants of time at $2N$ frequencies. Using the split-radix 2 FFT of Sorensen to calculate all the instants and frequencies, the computational cost of the Cunningham and Williams Fast TFD is then $rN(2N\log_2 N + 8N + 4)$ multiplications and $rN(6N\log_2 N + 14N + 4) - 4N^2$ additions. The value, r, is the number of eigenvalues (and hence spectrograms) that the algorithm uses to approximate the GDTFD.

6.1.1.2. The Parallel Weighted Spectrogram Time-Frequency Distribution (PWS TFD). For maximum throughput, the parallel version of this algorithm calculates each weighted spectrogram in a separate processor. If the kernel is not full rank or an approximation is used, the value N is replaced by r where $r < N$. The algorithm is a two stage process where the first stage calculates the individual spectrograms and the second stage sums all the spectrograms together.

The cost to compute the first stage is then driven by the number of real multiplies and additions needed to calculate the spectrogram. For $2N$ frequencies, the number of multiplies and additions is given in the column titled, "Stage One." The cost of the second stage is driven by the number of spectrograms which must be summed together. There are

no multiplications necessary and the number of additions is given in the column titled, "Stage Two."

The time it takes to calculate a distribution is dependent upon the greatest number of calculations which must be performed in each stage and the number and types of main memory transactions which must be done. The memory transactions are extremely device dependent and are beyond the scope of this paper. In Table 4, the Longest Path is the total number of multiplies and additions needed to calculate a single column of the GDTFD in the most heavily loaded processor in each stage. This figure will be used as a device independent measure of the throughput of the algorithm.

The Parallel Complexity of the algorithm is given by the cost of the Longest Path times the number of parallel processors necessary to calculate $2N$ frequencies at N instants of time, simultaneously. The Sequential Complexity is the number of multiplies and additions necessary to implement the algorithm on a sequential machine.

Table 6.1: Computational Costs of the PWS TFD

	Stage One	Stage Two	Longest Path	Parallel Complexity	Sequential Complexity
×	$2N\log_2 N + 8N + 4$	0	$2N\log_2 N + 8N + 4$	$N^2(2N\log_2 N + 8N + 4)$	$N^2(2N\log_2 N + 8N + 4)$
+	$6N\log_2 N + 14N + 4$	$2N\log_2 N$	$8N\log_2 N + 14N + 4$	$N^2(8N\log_2 N + 14N + 4)$	$N^2(6N\log_2 N + 14N + 4) + 2N^2\log_2 N$

6.2. Singular Value Decomposition MRTFD

In this section, a MRTFD based upon a modification of the SVD technique introduced in Chapter 5 is presented. The SVD MRTFD is based upon the inner product formulation of the GDTFD, and as a result, it can be used to calculate the GDTFD at a single instant in time. As such, it is called an isolated column method. In section 6.2.1, the theoretical framework for the SVD MRTFD is presented. Then, in section 6.2.2, the important

special case of a Hermitian kernel is considered. Finally, the resulting algorithm and its associated computational complexity are discussed in sections 6.2.3 and 6.2.4, respectively.

6.2.1. Algorithm Development–A General SVD-Based Multirate TFD. The first step in creating a SVD MRTFD is to decompose the inner product operation into a sum of smaller operations. Consider the inner product form of the discrete TFD,

$$C_f(n,k;\psi) = \langle \tilde{\psi}(t_1,t_2)\, x(t_1), x(t_2) \rangle \tag{6.3}$$

where

$$x(t) = f(n+t)\, e^{-j2\pi k \frac{(n+t)}{N}} \tag{6.4}$$

where $n \in \mathbf{Z}$, and $k, t_1, t_2 = 0,1,2,\ldots,N\text{-}1$.

Let t_1^o and t_1^e represent the odd and even terms of t_1, respectively. Then, $C_f(n,k;\psi)$ can be rewritten in terms of t_1^o and t_1^e as

$$C_f(n,k;\psi) = \langle \tilde{\psi}(t_1^o,t_2)\, x(t_1^o) + \tilde{\psi}(t_1^e,t_2)\, x(t_1^e), x(t_2) \rangle, \tag{6.5}$$

or equivalently as

$$C_f(n,k;\psi) = \langle \tilde{\psi}(t_1^o,t_2)\, x(t_1^o), x(t_2) \rangle + \langle \tilde{\psi}(t_1^e,t_2)\, x(t_1^e), x(t_2) \rangle. \tag{6.6}$$

Similarly, replacing $x(t_2)$ with the sum $x(t_2^0) + x(t_2^e)$, the inner product form of the GDTFD can be rewritten as

$$\begin{aligned} C_f(n,k;\psi) &= \langle \tilde{\psi}(t_1^o,t_2^o)\, x(t_1^o), x(t_2^o) \rangle + \langle \tilde{\psi}(t_1^o,t_2^e)\, x(t_1^o), x(t_2^e) \rangle \\ &+ \langle \tilde{\psi}(t_1^e,t_2^o)\, x(t_1^e), x(t_2^o) \rangle + \langle \tilde{\psi}(t_1^e,t_2^e)\, x(t_1^e), x(t_2^e) \rangle \end{aligned} \tag{6.7}$$

Equation (6.7) could be rewritten in the form

$$C_f(n,k;\psi) = \left\langle \begin{bmatrix} \tilde{\psi}(t_1^e,t_2^e) & \tilde{\psi}(t_1^o,t_2^e) \\ \tilde{\psi}(t_1^e,t_2^o) & \tilde{\psi}(t_1^o,t_2^o) \end{bmatrix} \begin{bmatrix} x(t_1^e) \\ x(t_1^o) \end{bmatrix}, \begin{bmatrix} x(t_2^e) \\ x(t_2^o) \end{bmatrix} \right\rangle \tag{6.8}$$

In the preceding, the odd-even sampling performed on the kernel and signal is equivalent to decimating the signal by two in both the t_1 and t_2 directions and decimating the kernel by

$$\mu = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = 2I_2 \tag{6.9}$$

(see, for example, [14] or [50], for more detail).

A more general case for arbitrary square decimation will now be developed. The decimation matrix for square decimation is given by

$$\mu = \begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix} = mI_2. \tag{6.10}$$

Now, assume there is an arbitrary square kernel, $A_{N\times N}$, such that its dimension, N, is devisable by m such that let $N/m = L \in \mathbf{Z}$. Define a block matrix, A', as was done in (3.4) and (3.5). The blocks in (3.4) are called the decimated kernels because, in practice, each block matrix is the result of two dimensional decimation. They operate on the decimated signal, each decimated kernel in a different multirate channel. Equation (3.4) is represented in block form as a notational convenience, but it is important to remember that each block will be processed as a separate entity.

The signal must also be rearranged to take advantage of the block structure. Returning to $x(t)$ as defined in (6.4), the signal is rearranged as was done in (2.15) to create $x'(t)$. This is the decimated signal.

The generalized square decimation algorithm (or the square multirate algorithm) solves

$$C_f(n,k;A) = \langle A'(t_1,t_2)\,x'(t_1), x'(t_2)\rangle. \tag{6.11}$$

If $m = 2$, then (6.11) and (6.8) are identical. Further, if $A = \tilde{\psi}$, then (6.11) is a MRTFD.

To form the SVD MRTFD, the Singular Value Decomposition of each block of the decimated kernel must be calculated. Each block of (3.4) when applied to the sub-vector, $x_i(t)$, defined in (2.16) becomes

$$\langle A_{a,b}(t_1,t_2)\,x_a(t_1), x_b(t_2)\rangle = \qquad (6.12)$$

$$\sum_{j=1}^{r} \sigma_j^{a,b} \langle x_b(t_2), u_j^{a,b}(t_2)\rangle^* \langle x_a(t_1), v_j^{a,b}(t_1)\rangle$$

Substituting the definition for $x_i(t)$ as defined in (6.4), letting $A = \tilde{\psi}$ and summing over all possible values of a and b, yields the equation for the generalized square decimation DTFD,

$$C_f(n,k;\psi)$$

$$= \sum_{a=1}^{m-1}\sum_{b=1}^{m-1} e^{\frac{j2\pi k(a-b)}{N}} \sum_{i=1}^{r_{a,b}} \sigma_i^{a,b} \left(\sum_{t=0}^{L-1} (v_i^{a,b}(t))^* f(mt+a)\, e^{-\frac{j2\pi kt}{L}} \right)$$

$$\times \left(\sum_{t=0}^{L-1} (u_i^{a,b}(t))^* f(mt+b)\, e^{-\frac{j2\pi kt}{L}} \right)^* \qquad , \qquad (6.13)$$

or equivalently,

$$C_f(n,k;\psi) = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1} e^{\frac{j2\pi(a-b)k}{N}} \sum_{i=1}^{r_{a,b}} \sigma_i X_{v_i,a}^{a,b}(k)\, (X_{u_i,b}^{a,b}(k))^* \qquad (6.14)$$

where

$$X_{w_i,\alpha}^{a,b}(k) = \sum_{t=0}^{L-1} (w_i^{a,b}(t))^* f(mt+\alpha)\, e^{-\frac{j2\pi kt}{L}} \qquad (6.15)$$

and $w_i^{a,b}$ represents either $v_i^{a,b}$ or $u_i^{a,b}$. Note that $X_{w_i,\alpha}^{a,b}(k)$ is periodic with period $L = N/m$ (i.e. $X_{w_i,\alpha}^{a,b}(k) = X_{w_i,\alpha}^{a,b}(pL+k)$, $p \in \mathbb{Z}$). Using this observation together with (6.14), (6.13) can be rewritten as

$$C_f\left(n, p\frac{N}{m} + k_1;\psi\right) = \tag{6.16}$$

$$\sum_{a=0}^{m-1}\sum_{b=0}^{m-1} e^{\frac{j2\pi(a-b)p}{m}}\left(e^{\frac{j2\pi k_1(a-b)}{N}}\sum_{i=1}^{r_{a,b}}\sigma_i X_{v_i,a}^{a,b}(k_1)\,(X_{u_i,b}^{a,b}(k_1))^*\right)$$

where $p = 0, 1,\ldots, m-1$ and $k_1 = 0, 1,\ldots, L-1$.

Alternately, letting $d = a - b$ and $c = a + b$, equation (6.16) could be written as

$$\begin{aligned}
&C_f(n, pL + k_1;\psi)\\
&= \sum_{c=0}^{m-1} e^{\frac{j2\pi cp}{m}}\left(e^{\frac{j2\pi k_1 c}{N}}\sum_{d=0}^{m-1-c}\sum_{i=1}^{r_{d+c,d}}\sigma_i^{d+c,d}X_{v_i,d+c}^{d+c,d}(k_1)\,(X_{u_i,d}^{d+c,d}(k_1))^*\right)\\
&+ \sum_{c=0}^{m-1} e^{\frac{j2\pi cp}{m}}\left(e^{\frac{j2\pi k_1 c}{N}}\sum_{d=0}^{m-1-c}\sum_{i=1}^{r_{d,d+c}}\sigma_i^{d,d+c}X_{v_i,d}^{d,d+c}(k_1)\,(X_{u_i,d+c}^{d,d+c}(k_1))^*\right)\\
&- \sum_{d=0}^{m-1}\sum_{i=1}^{r_{d,d}}\sigma_i^{d,d}X_{v_i,d}^{d,d}(pL+k_1)\,(X_{u_i,d}^{d,d}(pL+k_1))^*
\end{aligned} \tag{6.17}$$

This is easier to implement as it makes the outer summation a DFT which can, of course be solved by taking the FFT. This is the SVD MRTFD for the *general* case, no assumption have been made as to the form of the kernel.

Equation (6.17) can be thought of as summing the result (6.13) along each diagonal of (3.4). This results in a single L point vector for each diagonal. The vectors form an array which has columns indexed by $-m + 1$ to $m - 1$ representing each diagonal and rows indexed by 0 to $L - 1$. If the array is broken into two arrays $[0,L) \times (-m,0]$ and $[0,L) \times [0,-m)$ (the zero column is duplicated) and the DFT of the rows are taken and summed together, the result is almost the GDTFD for time n. Since the zero column was included twice, it must be subtracted to produce the true GDTFD.

6.2.2. A SVD MRTFD for Hermitian Kernels. It is possible to significantly improve the performance of the SVD MRTFD algorithm by assuming that the kernel is Her-

mitian. Only a Hermitian kernel will produce a strictly real TFD, and realness is a very common constraint placed upon TFD's.

Given that the kernel is Hermitian, the sub-kernels along the main diagonal of (3.4) are also Hermitian. This can be seen by examining the elements of (3.5) when $i = j$. Since the elements of A are Hermitian, i.e. $a_{i,j} = a^*_{j,i}$, the diagonal elements of (3.4) will also be Hermitian since $a_{i+lm,\,j+km} = a^*_{i+km,\,j+lm}$ when $i = j$.

The off-diagonal elements of (3.4), on the other hand, will not be Hermitian, but each sub-kernel will be the complex conjugate transpose of transpose elements of A' (i.e. $A_{i,j} = A^{\dagger}_{j,i}$ where † indicates complex conjugate transpose). This can be seen by examining the elements of $A_{i,j}$ and $A_{j,i}$. The elements of $A_{i,j}$ are $a_{i+bm,\,j+cm}$ and the elements of $A_{j,i}$ are $a_{j+bm,\,i+cm}$ where $b, c \in [0,\ldots,N/m - 1]$. It follows that the elements of $A^{\dagger}_{j,i}$ are $a^*_{j+cm,\,i+bm}$ but this is the same as $a_{i+bm,\,j+cm}$ since $a_{i,j} = a^*_{j,i}$ is given; thus, $A_{i,j} = A^{\dagger}_{j,i}$. This property makes it unnecessary to compute both the sub-kernels above and below the main diagonal.

From the relationship $A_{i,j} = A^{\dagger}_{j,i}$, it is possible to show that the singular values of the transpose element of A' are the same. This combined with the fact that the left singular vectors of A are the eigenvectors of $AA^{\dagger}$ and the right singular vectors are the eigenvectors of $A^{\dagger}A$, it can be seen that the left (right) singular vectors of $A_{i,j}$ and the right (left) singular vectors of $A_{j,i}$ are the same. This can be used to reduce the complexity of (6.17).

Consider (6.13) for just the case of a transpose pair of sub-kernels (i.e. $A_{i,j}$ and $A_{j,i}$),

$$\sum_{i=1}^{r_{a,b}} \sigma_i \left(\sum_{t=0}^{L-1} (v_i^{a,b}(t))^* f(mt+a)\, e^{\frac{-j2\pi kt}{L}} \right) \left(\sum_{t=0}^{L-1} (u_i^{a,b}(t))^* f(mt+b)\, e^{\frac{-j2\pi kt}{L}} \right)^* + \sum_{i=1}^{r_{a,b}} \sigma_i \left(\sum_{t=0}^{L-1} (v_i^{b,a}(t))^* f(mt+b)\, e^{\frac{-j2\pi kt}{L}} \right) \left(\sum_{t=0}^{L-1} (u_i^{b,a}(t))^* f(mt+a)\, e^{\frac{-j2\pi kt}{L}} \right)^* \tag{6.18}$$

This is equivalent to

$$\sum_{i=1}^{r_{a,b}} \sigma_i \left(\sum_{t=0}^{L-1} (v_i^{a,b}(t))^* f(mt+a) e^{\frac{-j2\pi k}{L}} \right) \left(\sum_{t=0}^{L-1} (u_i^{a,b}(t))^* f(mt+b) e^{\frac{-j2\pi k}{L}} \right)^*$$

$$+ \qquad (6.19)$$

$$\sum_{i=1}^{r_{b,a}} \sigma_i \left(\sum_{t=0}^{L-1} (u_i^{a,b}(t))^* f(mt+b) e^{\frac{-j2\pi k}{L}} \right) \left(\sum_{t=0}^{L-1} (v_i^{a,b}(t))^* f(mt+a) e^{\frac{-j2\pi kt}{L}} \right)^*$$

Since the second part of the sum in (6.19) is just the complex conjugate of the first part, only the real part of one of them must be calculated and the result multiplied by two.

The resulting improved algorithm is given by

$$\begin{aligned} & C_f(n, pL+k_1; \psi) \\ &= 2\mathrm{Re}\left[\sum_{c=0}^{m-1} e^{\frac{-j2\pi cp}{m}} \left(e^{\frac{j2\pi k_1 c}{N}} \sum_{d=0}^{m-1-c} \sum_{i=1}^{r_{d+c,d}} \sigma_i^{d+c,d} X_{v_i, d+c}^{d+c,d}(k_1) \left(X_{u_i, d}^{d+c,d}(k_1)\right)^* \right) \right] \\ &- \sum_{d=0}^{m-1} \sum_{i=1}^{r_{d,d}} \sigma_i^{d,d} X_{v_i, d}^{d,d}(pL+k_1) X_{u_i, d}^{d,d}(pL+k_1)^* \end{aligned} \qquad (6.20)$$

This is the SVD MRTFD for Hermitian kernels. It is important to note that for the sub-kernels along the main diagonal of (3.4), the eigenvalue decomposition may be used in place of the SVD—resulting in a reduction by a factor of two of the number of STFT's that need to be performed for these sub-kernels.

6.2.3. The SVD MRTFD Algorithm. The algorithm given in this section will implement the SVD MRTFD for Hermitian kernels. The algorithm for the general case given in section 6.2.1 can be obtained by a very simple modification of the one presented below. These modifications will be briefly discussed at the end of this section.

Using the singular vectors as windows and the singular values as weights, the first stage calculates each STFT pair and multiplies them together for each sub-kernel off the main diagonal. For the main diagonal sub-kernels, the corresponding eigenvectors are

used as window functions and the eigenvalues are used as weights. Stage two calculates the sum of the each sub-block. The result is a $[0, L-1] \times [0, m-1]$ matrix. In the third stage, the FFT of each row is taken, the real portion is multiplied by two and each k_1^{th} row is subtracted by the constant (which is also real) representing the k_1^{th} element of the summation of the blocks of the main diagonal. If the summation stage is implemented using a binary tree structure, then it is possible to sum the L outputs of m processors in the time it would take one processor to perform $L\log_2 m$ additions.

For maximum throughput, this algorithm requires $r_{a,b}$ processors for each block; thus, a total of $r_{a,b}(m^2 + m)/2$ processors are necessary. The calculation of the main diagonal (via eigenvalues) is slightly different than the remainder of the blocks (via SVD).

Algorithm 6.1: SVD MRTFD for Hermitian Kernels

	Barrier 0: Start
Stage 1:	Load each of the $r_{a,b}(m^2 + m)/2$ processors with the appropriate portions of the signal
	Calculate the two STFT's (one in the case of the main diagonal)
	Pointwise multiply the result (spectrogram for the main diagonal)
	Barrier 1:
Stage 2:	Sum each block and along each diagonal
	Output the result of the sum of each diagonal as a column in a matrix
Stage 3:	Load the rows of the matrix into L processors
	Take the FFT of each row
	Multiply the real part by two
	Subtract by the main diagonal
	Barrier 3: End

In the general case of the SVD MRTFD algorithm, $r_{a,b}m^2$ processors would be required for maximum throughput. These would be loaded with the appropriate sub-kernels and the two STFT's would be calculated for all of the blocks. This includes the main diag-

onal sub-kernels which are no longer Hermitian, and as such, they cannot make use of eigenvalues and eigenvectors. All other operations would be the same as in the algorithm for Hermitian kernels.

6.2.4. Computational Costs of the Improved Algorithm. The computational costs described in this section are for the SVD MRTFD for Hermitian kernels. The modifications to these results for the general case will be briefly discussed at the end of the section.

The longest path in the first stage of the algorithm is the number of additions and multiplications it takes to compute two L point FFT's and a L point complex multiply. The cost of stage two is the cost to compute the sums of the blocks and the diagonals. Stage three has a cost driven by a m point FFT and a m point real multiply and add.

Table 6.2: Computational Costs of the SVD MRTFD

	Stage One	Stage Two	Stage Three	Longest Path
×	$2\frac{N}{m}\log_2\left(\frac{N}{m}\right)+4\frac{N}{m}+8$	0	$m\log_2 m-2m+4$	$2\frac{N}{m}\log_2\left(\frac{N}{m}\right)+4\frac{N}{m}$ $+m\log_2 m-2m+12$
+	$6\frac{N}{m}\log_2\left(\frac{N}{m}\right)+2\frac{N}{m}+8$	$\frac{N}{m}\log_2(r_{a,b}m)$	$3m\log_2 m-2m+4$	$6\frac{N}{m}\log_2\left(\frac{N}{m}\right)+2\frac{N}{m}$ $+\frac{N}{m}\log_2(r_{a,b}m)$ $+3m\log_2 m-2m+12$

Table 6.2: Computational Costs of the SVD MRTFD (continued)

	Parallel Complexity	Sequential Complexity
×	$r(m^2+m)\left(\frac{N}{m}\log_2\left(\frac{N}{m}\right)+2\frac{N}{m}\right.$ $\left.+\frac{m}{2}\log_2 m-m+6\right)$	$rN\left(m\log_2\left(\frac{N}{m}\right)+2m+2\right)$ $+N\log_2 m-2N+4\frac{N}{m}+4rm^2$
+	$r\left(\frac{m^2+m}{2}\right)\left(6\frac{N}{m}\log_2\left(\frac{N}{m}\right)+2\frac{N}{m}\right.$ $+\frac{N}{m}\log_2(r_{a,b}m)$ $\left.+3m\log_2 m-2m+12\right)$	$rN\left(3m\log_2\left(\frac{N}{m}\right)+m+1\right)$ $+N\log_2 m-2N+4\frac{N}{m}+4rm^2$

In order to compare this new algorithm to the baseline algorithms in section 6.1.1, it will be assumed that the memory access times and associated communications costs are negligible compared to the computational costs. The time it takes to compute the parallel baseline and SVD MRTFD with Hermitian kernels are displayed relative to the time it takes to compute the Cunningham and Williams fast algorithm. In Figure 6.1, the line indicating the fraction of the time it takes to compute the parallel baseline compared to the Cunningham and Williams baseline is labeled "Parallel Baseline Algorithm." The parallel baseline is calculated as detailed in section 6.1.1.2 where each weighted spectrogram is calculated in a separate processor. The fastest performance possible by this algorithm is the time it takes to calculate a $2N$ point weighted spectrogram.

The relative time it takes to compute the SVD MRTFD compared to the Cunningham and Williams baseline is given for two levels of decimation, m. As can be seen, as the level of decimation is increased the relative time it takes to calculate the distribution decreases. This is due to the increased number of parallel paths and the reduced size of the problem each path must solve. The total number of additions and multiplications will rise somewhat with increasing levels of decimation. The plot compares the total number of multiplies and additions for the three algorithms.

The speed of the Parallel Baseline and SVD MRTFD are largely unaffected by the number of singular values and eigenvalues weights which are used. Since, for maximum throughput, it is assumed that there are as many parallel paths as desired, additional singular values and eigenvalues weights will only slightly increase the number of additions performed in the second stage, and compared to the cost of the overall algorithms, these additions can be neglected Thus, the time to compute these algorithms compared to the Cunningham and William baseline are unaffected by the accuracy of the Parallel Baseline and the SVD MRTFD.

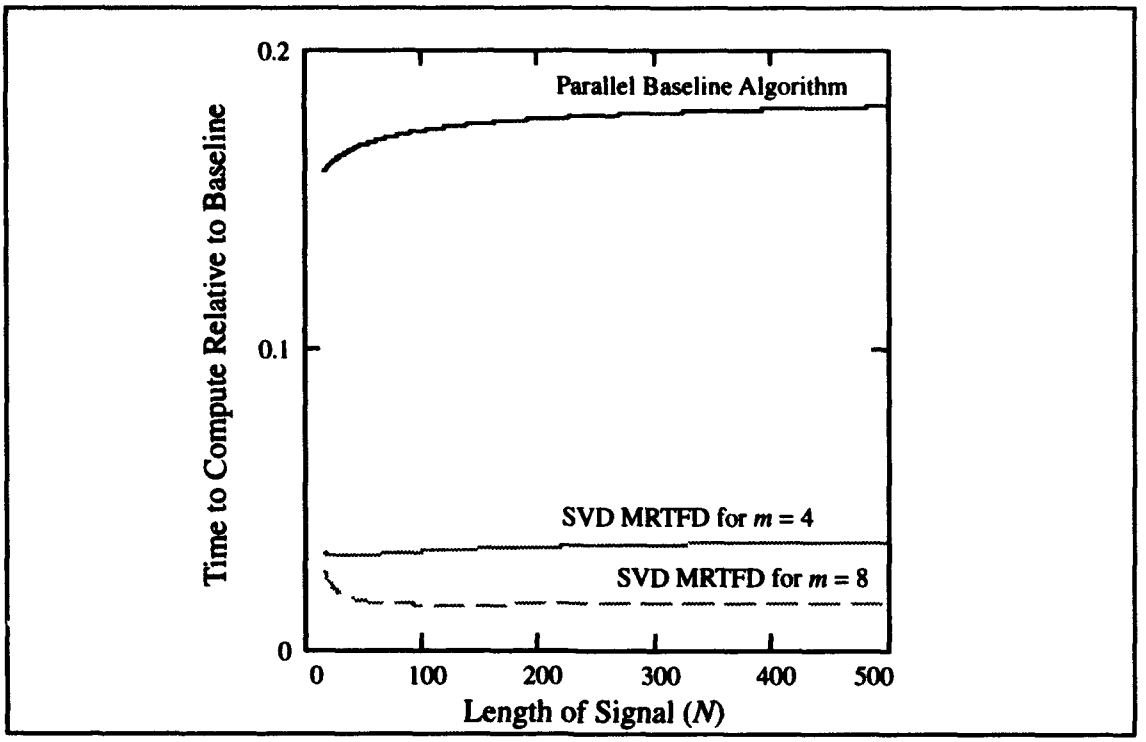


Figure 6.1. Comparison of Cost to Compute Parallel Baseline and SVD MRTFD with Normalized Cost of Cunningham and Williams Baseline Algorithm.

For the case of a signal which is 128 point long, the SVD MRTFD with a decimation factor of eight can have and increase throughput of over 50 fold relative to the baseline Cunningham and Williams algorithm. For a decimation factor of 16, the throughput can increase by a factor of 100.

The cost of the general (non-Hermitian) SVD MRTFD is only slightly larger than the SVD MRTFD with Hermitian kernels. If it is assumed that there are as many parallel paths as desired, then the cost is roughly the same between the two algorithms; however, the general SVD MRTFD will require twice as many parallel paths to achieve that throughput.

6.3. Circular Convolution Multirate Time-Frequency Distribution

The CC MRTFD is based upon the outer product form of the GDTFD, and as such, it is a two stage algorithm. It is a block Time-Frequency method calculating every column (instants of time) in a block, simultaneously. First, it performs a multirate circular convo-

lution of the kernel and the bilinear signal using MR FFT's. In other words, the algorithm uses multirate techniques to implement the circular convolution of the rows. It then uses the MR FFT to calculate the Fourier transform of the columns.

This section has three parts: first, the CC MRTFD theoretical background is introduced in section 6.3.1 In section 6.3.2, the algorithm is given, and then, in section 6.3.3, the computational cost of the algorithm is discussed.

6.3.1. The Circular Convolution MRTFD Algorithm Theory. Consider the outer product formulation of the GDTFD given by

$$C(n, f;\phi) = \sum_{k=-N}^{N-1} \sum_{p=-\frac{N}{2}}^{\frac{N}{2}-1} \phi(p,k) \left(\sum_{u=-\frac{N}{2}}^{\frac{N}{2}-1} R_f(u,k)\, e^{\frac{j2\pi up}{N}} \right) e^{\frac{-j2\pi pn}{N}}\, e^{\frac{-j\pi kf}{N}} \tag{6.21}$$

where

$$R_s(u,k) = f\left(u+\frac{k}{2}\right) f^*\left(u-\frac{k}{2}\right) \tag{6.22}$$

and

$$\phi(p,k) = \begin{cases} \sum_{t=-N/2}^{N/2-1} \psi(t,k)\, e^{j2\pi tp/N} & k \text{ even} \\ \sum_{t=-N/2}^{N/2-1} \psi\left(t-\frac{1}{2},k\right) e^{j2\pi tp/N} & k \text{ odd} \end{cases} . \tag{6.23}$$

Equation (6.21) can be thought of as a two stage processes. The first stage calculates the circular convolution of the row of the hexagonally decimated kernel, ψ, and the bilinear signal, R_f. Each row is independent and can be broken off as a separate process. The output of this stage is placed in a rectangular array indexed by n and k such that $N \in \{-N/2,\ldots, N/2-1\}$ and $k \in \{-N, \ldots, N-1\}$. The second stage takes the output of the circular convolutions which now lie on a rectangular grid and calculates the Fourier transform of each column, n.

The main computational building block in the CC MRTFD is the MR FFT which was presented in Chapter 3. The MR FFT is a two stage process. A convolution built upon the MR FFT is, in general, a four stage process. If the kernel is fixed and can be calculated a priori, the MR FFT based circular convolution is a three stage process.

To perform the multirate circular convolution, first zero pad the rows of the kernel to prevent convolution aliasing and perform the MR FFT of the kernel (this will be precomputed a priori for fixed kernels). Next, the MR FFT of the rows of the bilinear signal are calculated followed by the pointwise multiplication of the transformed kernel and transformed bilinear signal, i.e.

$$\begin{bmatrix} X(0)Y(0) & X\left(\frac{N}{m}\right)Y\left(\frac{N}{m}\right) & \cdots & X\left[(m-1)\frac{N}{m}\right]Y\left[(m-1)\frac{N}{m}\right] \\ X(1)Y(1) & X\left(\frac{N}{m}+1\right)Y\left(\frac{N}{m}+1\right) & \cdots & X\left[(m-1)\frac{N}{m}+1\right]Y\left[(m-1)\frac{N}{m}+1\right] \\ \vdots & \vdots & & \vdots \\ X\left[\left(\frac{N}{m}-1\right)\right]Y\left[\left(\frac{N}{m}-1\right)\right] & & \cdots & X(N-1)Y(N-1) \end{bmatrix} \tag{6.24}$$

where $X(i)$ and $Y(i)$ is a particular Fourier coefficient of the signal and kernel, respectively. If each row of (6.24) is local to a single processor, then there is no synchronization requirement and the inverse MR FFT can be directly calculated.

The inverse MR FFT is performed by first calculating the inverse FFT of the rows of (6.24) followed by the application of the inverse phase shift. At this point a barrier is encountered and the second stage is complete. The result of this operation is the Zak transform of the convolution of a row of the kernel and bilinear signal. The final step is to calculate the inverse FFT of the columns to produce the circular convolution of a single row of the kernel and the bilinear signal. The complete process is illustrated in Figure 6.2.

This method for calculating a multirate circular convolution is applied to all the rows of the bilinear signal and the kernel. Each of the $2N$ multirate convolutions is a three stage

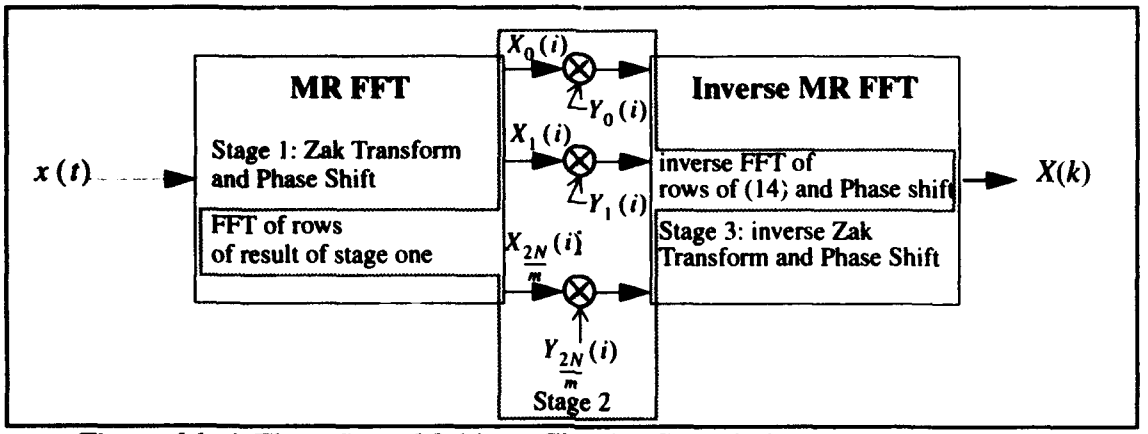


Figure 6.2. A Three Stage Multirate Circular Convolution. *Y* Represents the Fourier Coefficients of the Kernel.

process; however, each convolution is independent and all must be completed prior to taking the Fourier transform of the columns.

Once all the convolutions have been calculated, it is time to perform the Fourier transform of the columns. The columns are calculated using the multirate FFT. The two stages of the MR FFT are considered to be sub-stages, and the Fourier transform of the all the columns is Stage Two. With the completion of Stage Two, the GDTFD is done.

6.3.2. The CC MRTFD Algorithm. The loading of the individual processors averages far less than 100 percent with this algorithm. If throughput is maximized, then the $2N$ convolutions require $\sqrt{2N}$ processors each resulting in a total of $(2N)^{3/2}$ processors for the first stage; however, only N MR FFT need to be done in the second stage which suggests that for maximum throughput a total of $N\sqrt{2N}$ processors are needed. Thus, half of the processor are idle during the second pass; however, the second pass is not as long as the first so the total idle time for the processors is less than 50 percent. A system which uses $(2N)^{3/2}$ processor will be called the fast CC MRTFD since this configuration would maximized throughput.

An alternative algorithm can be defined which maximizes processor utilization. If the input is broken into two $N \times 2N$ arrays which are processed sequentially prior to calculating the Fourier transform of the column, none of the processors will be idle during the

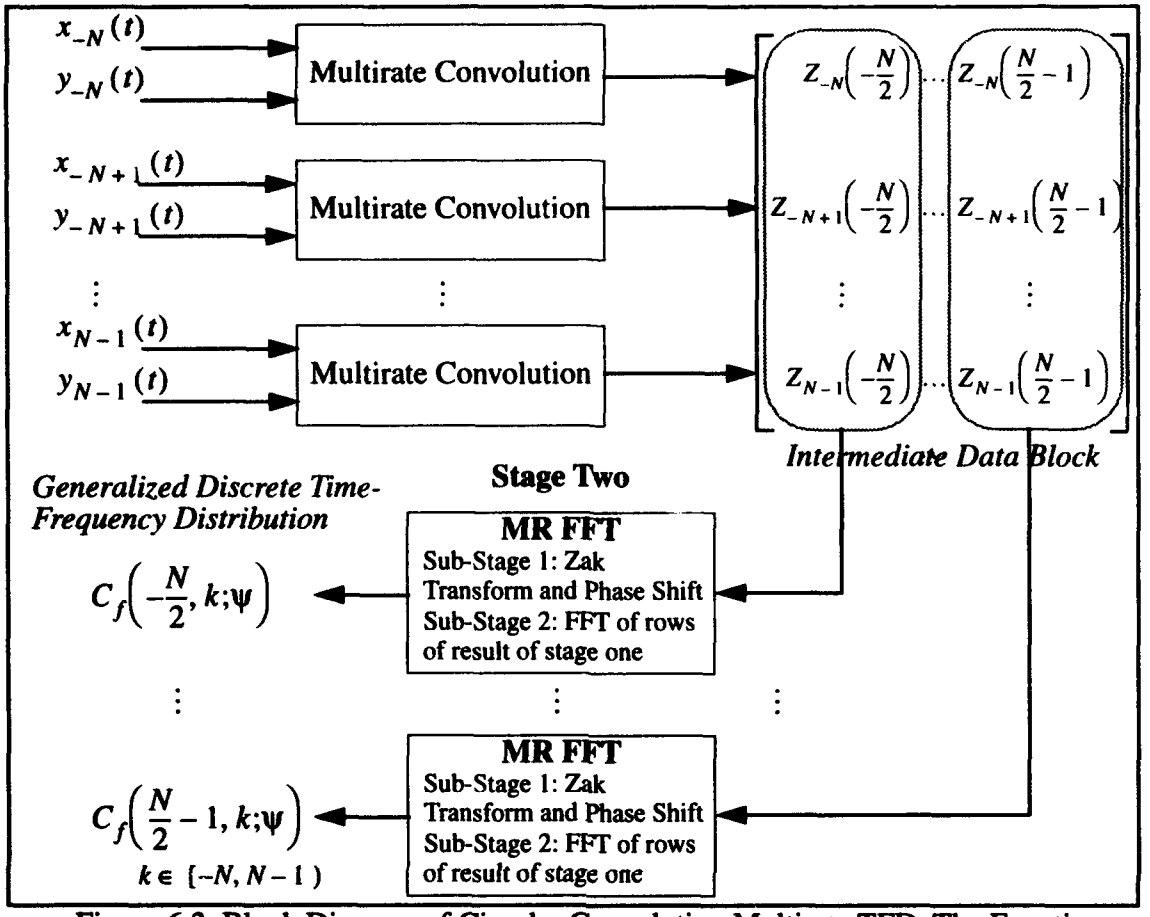


Figure 6.3. Block Diagram of Circular Convolution Multirate TFD. The Function $x_i(t)$ and $y_i(t)$ Refer to the i^{th} Row of the Bilinear Signal and Kernel, Respectively.

second stage. This algorithm will increase the time it takes to compute the GDTFD by slightly more than 50 percent. This algorithm will be called the efficient CC MRTFD since it maximizes processor utilization. A block diagram of this algorithm is shown in Figure 6.3.

It is assumed that $m = \sqrt{2N}$ in the both algorithms to prevent excess idle time in the processors during each MR FFT. The algorithms flow as shown in Algorithm 3.2

6.3.3. The Computational Cost of the Circular Convolution MRTFD. The cost for the longest path in stage one is the cost of one $2N$ point MR convolution. Assuming $m = \sqrt{2N}$ and the MR FFT of the kernel has been pre-calculated, the cost of the longest path is twice the cost of a MR FFT plus $\sqrt{2N}$ point complex multiply for the fast algo-

Algorithm 6.2: CC MRTFD Algorithms

	Fast CC MRTFD		Efficient CC MRTFD
	Barrier 0: Start		*Barrier 0: Start*
Stage 1:	Load $2N$ rows of bilinear signal into $\sqrt{2N}$ processors per row	Stage 1:	Load first N rows of bilinear signal into $\sqrt{2N}$ processor per row
	Each set of $\sqrt{2N}$ processors perform a MR convolution		Each set of $\sqrt{2N}$ processors perform a MR convolution
	Place result in temporary storage		Place result in temporary storage
			Load second N rows of bilinear signal into $\sqrt{2N}$ processor per row
			Each set of $\sqrt{2N}$ processors perform a MR convolution
			Place result in temporary storage
	Barrier 1:		*Barrier 1:*
Stage 2:	Load the $-N/2$ to $N/2-1$ columns of the temporary storage into the $\sqrt{2N}$ processors per column	Stage 2:	Load the $-N/2$ to $N/2-1$ columns of the temporary storage into the $\sqrt{2N}$ processors per column
	Each set of $\sqrt{2N}$ processors perform a MR FFT		Each set of $\sqrt{2N}$ processors perform a MR FFT
	Barrier 2: End		*Barrier 2: End*

rithm and twice this figure for the efficient algorithm. The cost of the second stage for both algorithms is identical and is equivalent to the longest path of a single MR FFT. The longest path for the CC MRTFD is the sum of the first and second stages. The Parallel Complexity is simply the Longest Path times the number of processors: in this case, $(2N)^{3/2}$ for the fast algorithm and $N\sqrt{2N}$ for the efficient algorithm. The Sequential Complexity is the same for both algorithms and is the twice the sequential cost of a MR convolution

plus a MR FFT times $N\sqrt{2N}$.

Table 6.3: Computational Costs of the Fast CC MRTFD

	Stage One	Stage Two	Longest Path	Parallel Complexity	Sequential Complexity
×	$2\sqrt{2N}\log_2 N - \sqrt{2N} + 10$	$\sqrt{2N}\log_2 N - 2\sqrt{2N} + 5$	$3\sqrt{2N}\log_2 N - 3\sqrt{2N} + 15$	$2N(6N\log_2 N - 6N + 15\sqrt{2N})$	$N(10N\log_2 N - 8N + 10\sqrt{2N} + 15)$
+	$6\sqrt{2N}\log_2 N + 3\sqrt{2N} + 10$	$3\sqrt{2N}\log_2 N + 5$	$9\sqrt{2N}\log_2 N + 3\sqrt{2N} + 15$	$2N(18N\log_2 N + 6N + 15\sqrt{2N})$	$N(30N\log_2 N + 12N + 10\sqrt{2N} + 15)$

Table 6.4: Computational Costs of the Efficient CC MRTFD

	Stage One	Stage Two	Longest Path	Parallel Complexity	Sequential Complexity
×	$4\sqrt{2N}\log_2 N - 2\sqrt{2N} + 20$	$\sqrt{2N}\log_2 N - 2\sqrt{2N} + 5$	$5\sqrt{2N}\log_2 N - 4\sqrt{2N} + 25$	$N(10N\log_2 N - 8N + 25\sqrt{2N})$	$N(10N\log_2 N - 8N + 10\sqrt{2N} + 15)$
+	$12\sqrt{2N}\log_2 N + 6\sqrt{2N} + 20$	$3\sqrt{2N}\log_2 N + 5$	$15\sqrt{2N}\log_2 N + 6\sqrt{2N} + 25$	$N(30N\log_2 N + 12N + 25\sqrt{2N})$	$N(30N\log_2 N + 12N + 10\sqrt{2N} + 15)$

As was done in the case of the SVD MRTFD, a comparison of the time it takes to compute the CC MRTFD with the baseline algorithms must be performed. Again, the assumption is made that the time to compute is directly proportional to the number of multiplies and additions needed to calculate the longest path and communications costs are negligible.

In order to compare the CC MRTFD algorithm to the baseline algorithms found in section 6.1.1, it will be assumed that the memory access times and associated communications costs are negligible compared to the computational costs. The time it takes to compute the parallel baseline and the fast and efficient CC MRTFD's are displayed relative to the time it takes to compute the Cunningham and Williams fast algorithm. In Figure 6.4, the line indicating the fraction of the time it takes to compute the parallel baseline com-

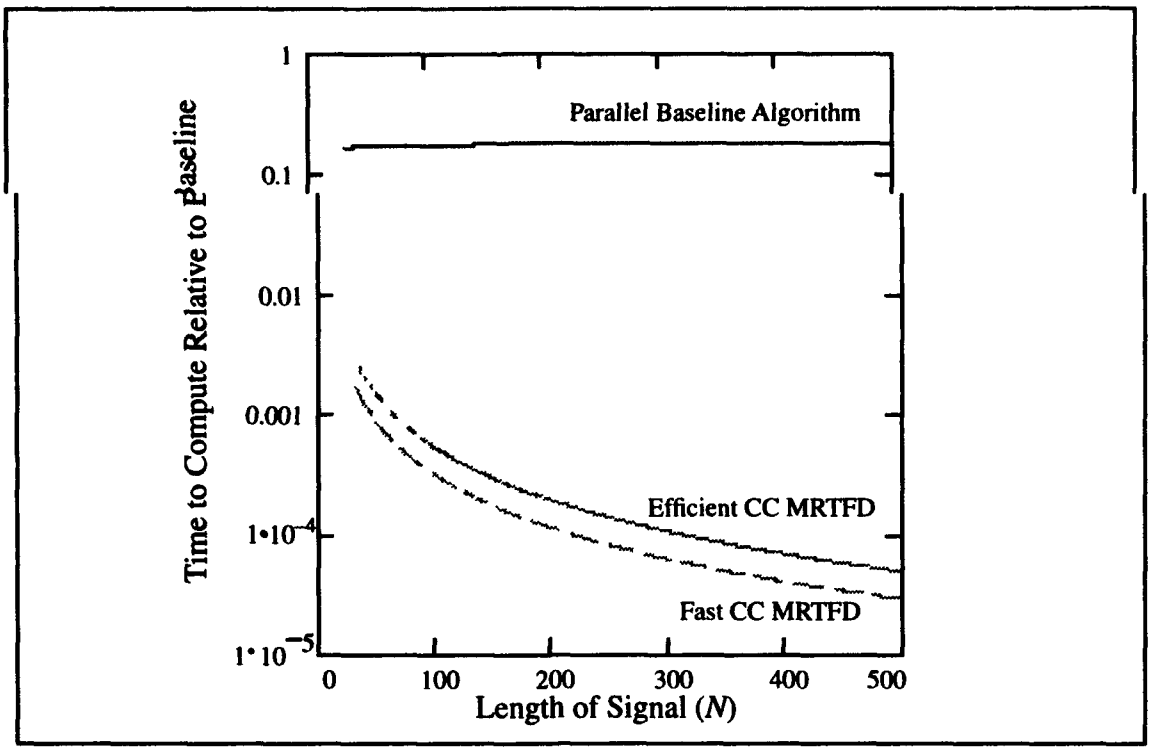


Figure 6.4. Comparison of Efficient CC MRTFD and Fast CC MRTFD to Parallel Baseline Algorithm and Normalized Cunningham and Williams Baseline Algorithm.

pared to the Cunningham and Williams baseline is labeled, "Parallel Baseline Algorithm." The parallel baseline is calculated as detailed in section 6.1.1.2 where each weighted spectrogram is calculated in a separate processor. The fastest performance possible by this algorithm is the time it takes to calculate a $2N$ point weighted spectrogram.

Figure 6.4 shows the potential speed up of the CC MRTFD over the Cunningham and Williams and parallel baselines. Note that unlike the SVD MRTFD which is depicted with constant values of m, the CC MRTFD uses a decimation factor, m, which changes with the length of the signal since $m = \sqrt{2N}$.

The CC MRTFD produces a distribution which is equivalent to having kept all the singular values and eigenvalue weights. The Parallel Baseline's throughput is largely unaffected by increased accuracy, as before. Thus, both the CC MRTFD and Parallel Baseline

provide methods for calculating the actual GDTFD while the Cunningham and Williams Baseline provides only an approximation.

For the case of a 128 point long signal, the potential speed up of the CC MRTFD over the baseline is on the order of 1000 fold for the efficient CC MRTFD and slightly more for the fast CC MRTFD. The decimation factor which is used for this length signal is 16 which implies 2048 processors are needed for maximum throughput for the efficient CC MRTFD and 4096 processors are needed for maximum throughput for the fast CC MRTFD to achieve this increase.

6.4. Comparison of SVD MRTFD and CC MRTFD

If the application to which the MRTFD is applied requires the time-frequency distribution at each possible instant of time, the CC MRTFD algorithm is the most logical choice. When calculating a block of data the CC MRTFD can be over an order of magnitude faster than the SVD MRTFD for the same number of parallel processors. The SVD MRTFD is appropriate when only a fraction of the distributions at the possible instants of time need to be calculated.

The performance of the SVD MRTFD can be improved by taking an approximation to the distribution; however, as the decimation factor rises the benefit associated with the approximation decreases. As the decimation factor increases, the size of the sub-kernels decrease. An approximation works by keeping a set number of singular values or eigenvalues and excluding the remainder. The performance is improved since fewer STFT's and/or spectrograms must be calculated, but for smaller sub-kernels, the number of singular values or eigenvalues excluded decreases; hence, the throughput improvement of the approximation decreases.

When deciding which algorithm to use the deciding factor is generally whether or not the distribution needs to be calculated at each instant of time. If it does, then the CC MRTFD is the clear choice. If it does not, then the SVD MRTFD may be appropriate. A

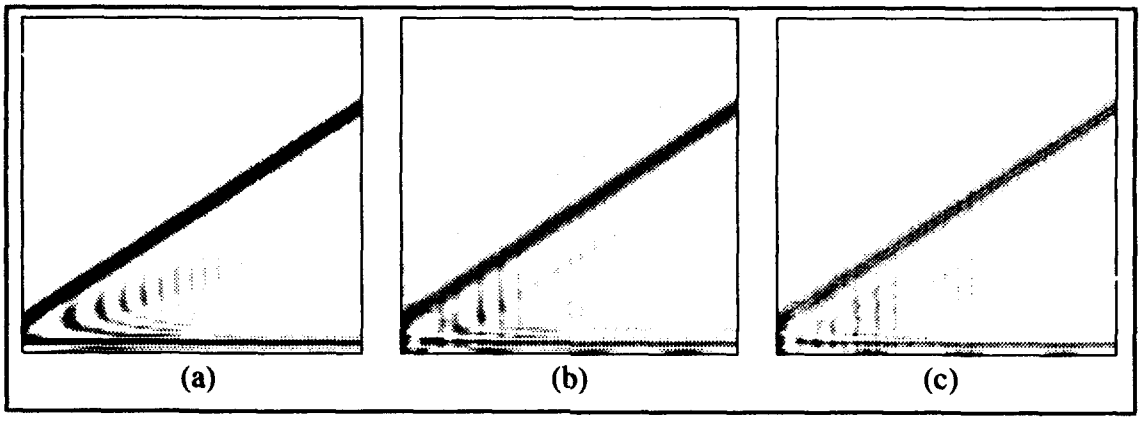


Figure 6.5. Comparison of Distribution Calculated by (a) CC MRTFD, (b) Cunningham and Williams Baseline Using Seven Eigenvalues and a Condition Number of 1.5, (c) SVD MRTFD Approximation Using a Condition Number of 1.5.

good estimate of the point at which the SVD MRTFD becomes the appropriate choice is when less than one-in-ten instants in time need to be calculated. The exact cross-over point depends upon the length of the signal, the size of the kernel, the decimation factor and the number of available processors. To determine the cross-over point, the Longest Path times the number of parallel paths needed for a given implementation of the CC MRTFD and SVD MRTFD must be compared.

6.5. Example

In this section an example is given which compares the Cunningham and Williams fast algorithm, the SVD MRTFD and CC MRTFD for the Binomial kernel. The signal being analyzed is composed of a rising chirp and a tone. The chirp has twice the magnitude of the tone.

In Figure 6.5a, the distribution generated by the CC MRTFD algorithm is shown. It is identical to the distribution that would be created by means of the Alias-Free Generalized Discrete Time-Frequency Distribution (See Chapter 4 or [29].), Weighted Spectrograms where all non-zero eigenvalues are used [18] or by the SVD MRTFD when all non-zero singular values are used.

The GDTFD calculated via the baseline Cunningham and Williams fast algorithm is shown in Figure 6.5b. This algorithm obtains its speed, primarily, by reducing the number of eigenvalues that are kept and, hence, the number of weighted spectrograms which must be calculated. Figure 6.5b is the distribution which results when only the seven largest eigenvalues are kept. This is equivalent to limiting the condition number of the kernel approximation to 1.5. For this data set, the resulting distribution is very close to the actual distribution in Figure 6.5a, but it is an approximation. The l_2 error of the approximation is bound by the largest magnitude of the eigenvalues not included. In this case, the error is bound by 0.667.

Figure 6.5c shows the approximation to the distribution which is obtained when not all of the singular values are kept in the SVD MRTFD algorithm. One of the advantages of the SVD MRTFD is the ability to control both the number of singular values kept in each sub-kernel and the decimation factor, which in turn affects both throughput and the number of non-zero singular values. To more accurately compare with Figure 6.5b, all the blocks of the SVD MRTFD have been limited to an l_2 error bound of 0.667 (a condition number of 1.5). The number of singular values kept in each of the sub-kernels ranges from one to nine depending upon how quickly the magnitudes of the singular values decay. This effect is known as deflation. It should be noted that this distribution is an approximation like the baseline fast algorithm, but it is not the same approximation.

6.6. Conclusion

Two new fast computational methods have been demonstrated for the calculation of Generalized Discrete Time-Frequency Distributions. The computational paradigm underlying the algorithms is multirate. The SVD MRTFD method is based upon the inner product formulation of the GDTFD and as such it can be used to calculate the frequency content of a signal for a particular instant in time. Even for modest decimation value of eight, throughput can be increased by as much as 50 fold over the current state-of-the-art Cun-

ningham and Williams fast algorithm and by an order of magnitude over the parallel baseline algorithm. The SVD MRTFD is less computationally expensive when calculating selected columns of the GDTFD, while the CC MRTFD is less expensive when calculating every column of the GDTFD.

7. Conclusion

7.1. Summary and Findings

The most significant and fundamental result presented in this dissertation is the new computational paradigm for multirate. Viewing multirate as a computational paradigm extends the multirate benefits (reduced required speed of computational elements, reduced cost, and increased throughput) to a much wider class of problems than just the design and implementation of filter banks. Specifically, multirate can be applied to any problem which can be expressed as a set of vector-vector, matrix-vector or matrix-matrix operations or any combination of these three types of operations [34]. It can and does lead to fast and parallel algorithms by revealing the underlying parallelism and the inherent recurrent nature of a particular problem.

Considering multirate as a computational model opens new avenues for multirate as a divide and conquer formalism. In this application, multirate could be used to solve any problem in numerical linear algebra. More importantly to the field of signal processing, there is a very large class of problems which, at heart, are numerical linear algebra problems. Thus, in signal processing, multirate can be used both as a means to design and implement filter banks and as an underlying computational paradigm for other types of problems. For example, multirate was applied to the Fast Fourier Transform (FFT) and Discrete Hartley Transform (DHT) to produce fast, parallel versions of these well know signal processing algorithms[34]. In fact, the remainder of the dissertation was an extended example of this paradigm.

Creation of a Multirate Time-Frequency Distribution (MRTFD) algorithm opens the door for a new class of fast and parallel algorithms for Time-Frequency Analysis [33]. The first algorithms in this new class are the Singular Value Decomposition MRTFD (SVD

MRTFD) and the Circular Convolution MRTFD (CC MRTFD) which demonstrate the potential to increase the throughput of a Generalized Discrete Time-Frequency Distribution (GDTFD) by well over an order of magnitude. Specifically, the two MRTFD's presented here yield increases in throughput, compared to the Cunningham and Williams baseline algorithm (the fastest algorithm reported to date in the literature), of 50 fold for even a modest decimation factor of eight. Unlike the Cunningham and Williams baseline algorithm, the SVD MRTFD and CC MRTFD do not trade accuracy for speed. As the decimation factor associated with the MRTFD increases, the parallelization of the algorithm increases and, as a result, the potential throughput of the MRTFD increases [33].

The SVD MRTFD is based upon the inner product formulation of the GDTFD and is based upon the Singular Value Decomposition of the kernel. It demonstrates the capability to produce the GDTFD for a single instant of time significantly faster than any existing algorithm. The improvement is strictly a function of the decimation factor and the number of parallel processing paths available.

The SVD MRTFD has the advantage of being able to allow engineering trade-offs between the number of processors in an implementation, the throughput of the system and the accuracy of the GDTFD produced. If there are insufficient parallel processing paths to implement the SVD MRTFD using every singular value, then an approximation can be made to reduce the number of parallel paths without increasing the time it takes to computed the GDTFD via the SVD MRTFD. If each sub-kernel in the SVD MRTFD is approximated by limiting the condition number of the sub-kernels, it is possible to reduce the number of Short-Time Fourier Transforms (STFT) necessary to implement the SVD MRTFD. As the condition number is decreased, the approximation to the GDTFD will become less precise, but the number of parallel processing paths will shrink. The exact reduction is dependent upon the specific kernel being used.

The CC MRTFD is based upon the outer product form of the GDTFD and calculates N instants in time simultaneously. It does this faster than any existing algorithm including the SVD MRTFD. Compared to the Cunningham and Williams baseline algorithm, the CC MRTFD is on the order two orders of magnitude faster given several hundred parallel processing paths. The CC MRTFD has the advantage of quickly calculating the exact GDTFD but must do so in blocks N long.

To create the MRTFD, it was necessary to create another new time-frequency analysis tool—the Decimated Generalized Discrete Time-Frequency Distribution (D-GDTFD). Its development was the culmination of the development of a new collection of Zak transform based time-frequency analysis tools. First, it was shown that the discrete Zak transform, with the addition of a window, becomes a generalization of the STFT. This new transform was called the Windowed Zak Transform (WZT). Building upon the WZT, the Zak-Spectrogram (ZS) was created and shown to be a generalization of the spectrogram. Lastly, the ZS was used in combination with the method of weighted spectrograms (via both eigenvalue decomposition and SVD) to create D-GDTFD. These distributions trade bandwidth for speed. For a decimation factor of m, there is an m fold increase in throughput (or speed of calculation). The corresponding reduction in discrete bandwidth is from 2π for the GDTFD to $2\pi/m$ for the D-GDTFD. An important attribute of the D-GDTFD is that it requires significantly less storage than the GDTFD. The D-GDTFD requires only $1/m^2$ of the storage of the GDTFD [35][38].

To allow the use of alias-free kernels designed in either the ambiguity or time-lag domains, new methods were created to allow the easy (and fast) numerical calculation of kernels in either domain from a kernel defined in the other domain regardless of the plane in which it was designed [36][37].

7.2. Recommendations

An interesting area of research which could now be done is to examine the problem of adaptive kernels from the perspective of multirate as the computational paradigm. Current adaptive kernel techniques require the repeated calculation of the entire distribution as the kernel is iteratively adapted to the signal. The Multirate Time-Frequency Distribution approach suggests the notion of selectively adapting the decimated kernels. Thus, the size of each optimization being performed is reduced permitting increased performance for each iteration of the kernel. Perhaps an interesting variation of this idea would be to adapt individual window functions in either the weighted spectrogram or SVD approach instead of adapting the entire kernel.

A second area for further research is the application of multirate to the multidimensional Time-Frequency Distribution. The throughput problems encountered with the Time-Frequency Distributions of one dimensional signal are greatly magnified with the addition of more dimensions in the input signal. Multirate, in this case multidimensional multirate, provides one approach to improving the performance of a multidimensional TFD. One of the benefits of multidimensional multirate is the ability to select optimum sampling structures or latices that were not available in the one dimensional case. This can greatly increase the efficiency of the algorithm reducing the overall cost, and the resulting algorithm can also be implemented in a parallel architecture to obtain the type of performance benefits detailed in this dissertation.

Appendix A. Spectrograms and GDTFD's

The spectrogram is given by

$$S_f(t,\omega) = |F_t(\omega)|^2 = \left| \int_{-\infty}^{\infty} f(\tau)\, h(\tau - t)\, e^{-j2\pi\tau\omega} d\tau \right|^2$$

$$= \left[\int_{-\infty}^{\infty} f(\tau)\, h(\tau - t)\, e^{-j2\pi\tau\omega} d\tau \right] \left[\int_{-\infty}^{\infty} f(\tau)\, h(\tau - t)\, e^{-j2\pi\tau\omega} d\tau \right]^* \tag{1}$$

Using the transforms pair given in [11], $x^*(-t) \leftrightarrow X^*(\omega)$, allows (1) to be rewritten as

$$S_f(t,\omega) = \left[\int_{-\infty}^{\infty} f(\tau)\, h(\tau - t)\, e^{-j2\pi\tau\omega} d\tau \right] \left[\int_{-\infty}^{\infty} f^*(-\tau)\, h^*(-\tau - t)\, e^{-j2\pi\tau\omega} d\tau \right] \tag{2}$$

Note that the convolution and the Fourier transform have the following relationship

$$x(t) * y(t) = \int_{-\infty}^{\infty} x(u)\, y(t-u)\, du = \mathrm{F}^{-1}_{\omega \to t}[X(\omega)\, Y(\omega)] \tag{3}$$

Applying the Fourier transform to the right hand side of (3), leads to an equation of the form found in (2) implying the spectrogram can be rewritten as

$$S_f(t,\omega) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(\xi)\, h(\xi - t)\, f^*(\xi - \tau)\, h^*(\xi - \tau - t)\, d\xi \right] e^{-j2\pi\tau\omega} d\tau \tag{4}$$

Applying a change of variables to the interior integral, (4) becomes

$$S_f(t,\omega) = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f\left(u + \frac{\tau}{2}\right) f^*\left(u - \frac{\tau}{2}\right) h\left(u + \frac{\tau}{2} - t\right) h^*\left(u - \frac{\tau}{2} - t\right) du \right] e^{-j2\pi\tau\omega} d\tau \tag{5}$$

Compare this result with the convolutional form of the Generalized Time-Frequency Distribution using a kernel based upon the bilinear form of the window, $w(t)$.

$$C_f(t, \omega; W) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} R_f(u, \tau)\, W(t-u, \tau)\, e^{-j2\pi\tau\omega} du d\tau \tag{6}$$

where $R_f(u, \tau) = f\left(u + \frac{\tau}{2}\right) f^*\left(u - \frac{\tau}{2}\right)$ and $W(u, \tau) = w\left(u + \frac{\tau}{2}\right) w^*\left(u - \frac{\tau}{2}\right)$. Placing the definitions into (6) yields,

$$\begin{aligned} C_f(t, \omega; W) = & \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f\left(u + \frac{\tau}{2}\right) f^*\left(u - \frac{\tau}{2}\right) \\ & \times w\left(-u + \frac{\tau}{2} + t\right) w^*\left(-u + \left(-\frac{\tau}{2}\right) + t\right) e^{-j2\pi\tau\omega} du d\tau \end{aligned} \tag{7}$$

Equations (5) and (7) bear a marked similarity. The only difference lies in the variables for h and w. On inspection, it can be seen that the variables of h are the negative of w^* and the variables of h^* are the negative of w. This implies that if h is symmetric or anti-symmetric and equal to w, then (5) and (7) are identical. In practice, this is not a severe restriction on the windows used for spectrograms.

BIBLIOGRAPHY

[1] S. G. Akl, The Design and Analysis of Parallel Algorithms, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[2] Amin, M. G., "Performance Comparison of Wigner-Ville Spectrum Estimators Using Least-Squares Approximation of Kernels," *Proc. ISSPA*, Gold Coast Australia, vol. 2, 1990.

[3] ————, "Time-Frequency Spectrum Analysis and Estimation for Non-Stationary Random Processes," in Time-Frequency Signal Analysis: Methods and Applications, B. Boashash (ed.), Longman Cheshire, Melbourne, Australia, 1992.

[4] E. Anderson, et al, LAPACK User's Guide, SIAM, Philadelphia, 1992.

[5] F. Auger, "Some Simple Parameter Determinations Rules for the Generalized Choi-Williams and Butterworth Distributions," *IEEE Signal Processing Letters*, Vol. 1, No. 1, pp 9–11, January 1994.

[6] L. Auslander, I. C. Gertner and R. Tolimieri, "The Discrete Zak Transform Application to Time-Frequency Analysis and Synthesis of Nonstationary Signals," *IEEE Trans. on Sig. Proc.*, Vol. 39, No. 4, pp. 825–835, April 1991

[7] L. Auslander and R. Tolimieri, "Is Computing with Finite Fourier Transforms Pure or Applied Mathematics," *Bulletin of the American Mathematical Society*, Vol. 1, pp 847-897, 1979.

[8] B. Boashash, ed. Time-Frequency Signal Analysis: Methods and Applications, John Wiley & Sons, New York, 1992.

[9] B. Boashash and A. Reilly, "Algorithms for Time-Frequency Signal Analysis," Time-Frequency Signal Analysis: Methods and Applications, ed. B. Boashash, John Wiley & Sons, New York, 1992.

[10] P. J. Boles and B. Boashash, "Application of the Cross-Wigner-Ville Distribution to Seismic Data Processing," Time-Frequency Signal Analysis: Methods and Applications, ed. B. Boashash, John Wiley & Sons, New York, 1992.

[11] R. N. Bracewell, The Fourier Transform and Its Applications, McGraw-Hill Book Company, New York, 1978.

[12] G. Brassard and P. Bratley, Algorithmics: Theory and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[13] T. Brotherton, T. Pollard and D. Jones, "Application of Time-Frequency and Time-Scale Representations to Fault Detection and Classification," *IEEE International*

Symposium on Time-Frequency and Time-Scale Analysis, pp. 95–98, October 1992.

[14] T. Chen and P. P. Vaidyanathan, "Recent Developments in Multidimensional Multirate Systems," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 3, No. 2, pp. 116-137, April 1993.

[15] T. A. C. M. Classen and W. F. G. Mecklenbräuker, "The Wigner Distribution—A Tool for Time-Frequency Signal Analysis, Part III: Relations With Other Time-Frequency Signal Transforms," *Philips J. Res.*, Vol. 35, No. 6, pp. 372–389, 1980.

[16] L. Cohen, "Time-Frequency Distributions–A Review," *Proceedings of the IEEE*, Vol. 77, No. 7, pp. 941-981, 1989.

[17] ————, "Generalized Phase-Space Distribution Functions," *J. Math. Phys.*, Vol. 7, pp. 781-786, 1966.

[18] G. S. Cunningham and W. J. Williams, "Kernel Decomposition of Time-Frequency Distributions," *IEEE Transactions on Signal Processing*, Vol. 42, No. 6, June 1994.

[19] ————, "Fast Implementation of Time-Frequency Distributions," *IEEE International Symposium on Time-Frequency and Time-Scale Analysis*, pp 241–244, October 1992.

[20] J. J. Dongarra, J. Du Croz, S. Hammarling and I. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, Vol. 16, pp. 1–17, 1990.

[21] J. J. Dongarra, J. Du Croz, S. Hammarling and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, Vol. 14, pp. 1–17, 1988.

[22] J. Fang, L. Atlas and G. Bernard, "Advantages of Cascaded Quadratic Detectors for Analysis of Manufacturing Sensor Data," *IEEE International Symposium on Time-Frequency and Time-Scale Analysis*, pp. 345–348, October 1992.

[23] B. D. Forrester, "Time-Frequency Analysis in Machine Fault Detection," in Time-Frequency Signal Analysis: Methods and Applications, ed. B. Boashash, John Wiley & Sons, New York, 1992.

[24] G. H. Golub and C. F. Van Loan, Matrix Computations, 2nd ed., Johns-Hopkins University Press, Baltimore, Maryland, 1991.

[25] J. Granata, M. Conner and R. Tolimieri, "Recursive Fast Algorithms and the Role of the Tensor Product," *IEEE Transactions on Signal Processing*, Vol. 40, No. 12, pp.2921–2930, December 1992.

[26] Hlawatsch, F., "Regularity and Unitarity of Bilinear Time-Frequency Signal Representations," *IEEE Trans. on Information Theory*, Vol. 38, No. 1, pp. 82-94, January 1992.

[27] K. Hwang and F. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984.

[28] A. J. E. M. Jansen, "The Zak Transform: A Signal Transform for Sampled Time-Continuous Signals," *Philips J. Res.*, Vol. 43, pp. 23–69, 1988.

[29] J. Jeong and W. J. Williams, "Alias-Free Generalized Discrete-Time Time-Frequency Distributions," *IEEE Transactions on Signal Processing*, Vol. 40, No. 11, pp. 2757-2765, 1992.

[30] ———, "A New Formulation of Generalized Discrete-Time Time-Frequency Distributions," *Proc. International Conference on Acoustics, Speech and Signal Processing*, pp. 3189-3192, March 1991.

[31] R. Koenig, H. Dunn and L. Lacy, "The Sound Spectrograph," *J. Acoust. Soc. Amer.*, Vol. 18, pp 19-49, 1946.

[32] Neng-Chung Hu, Hong-I Chang and O. K. Ersoy, "Generalized Discrete Hartley Transforms," *IEEE Trans. on Signal Processing*, Vol. 40, No. 12, December 1992.

[33] J. R. O'Hair and B. W. Suter, "Multirate: A New Computational Paradigm," Submitted to IEEE Transactions on Signal Processing, May 1994.

[34] ———, "Multirate: A New Computational Paradigm," Submitted to IEEE Transactions on Signal Processing, May 1994.

[35] ———, "The Zak Transform and Decimated Time-Frequency Distributions," Submitted to IEEE Transactions on Signal Processing, December 1993.

[36] ———, "Kernel Design Techniques for Alias-Free Time-Frequency Distributions," Submitted to IEEE Transactions on Signal Processing, July 1993.

[37] ———, "Kernel Design Techniques for Alias-Free Time-Frequency Distributions," IEEE International Conference on Acoustic, Speech and Signal Processing, Adelaide, Australia, Vol. III, pp 333–336. 19-22 April 1994.

[38] ———, "The Zak Transform and Decimated Spectrograms," IEEE International Symposium on Circuits and Systems, London, U.K., 30 May–2 June 1994.

[39] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[40] A. Papandreou and G. F. Boudreaux-Bartels, "Generalization of the Choi-Williams

Distribution and the Butterworth Distribution for Time-Frequency Analysis," *IEEE Transactions on Signal Processing*, Vol. 41, No. 1, pp. 463-472, 1993.

[41] S. Qian and D. Chen, "Discrete Gabor Transform," *IEEE Trans. on Sig. Proc.*, Vol. 41, No. 7, pp. 2429–2438, July 1993.

[42] K. R. Rao and P. Yip, Discrete Cosine Transform: Algorithms, Advantages, Applications, Academic Press, Boston, 1990.

[43] P. A. Regalla and S. K. Mitra, "Kronecker Products, Unitary Matrices and Signal Processing Applications," *SIAM Review*, Vol. 31, No. 4, pp. 586–613, December 1989.

[44] C. Runge, Zeit. fur Math. und Physik, 48 (1903) 943.

[45] V. P. Sathe and P. P. Vaidyanathan, "Effects of Multirate Systems on Statistical Properties of Random Signals," *IEEE Transactions on Signal Processing*, Vol. 41, No. 1, pp. 131–146, January 1993.

[46] H. V. Sorensen, M. T. Heideman and C. S. Burrus, "On Computing the Split-Radix FFT," *IEEE Trans. Acoust., Speech and Sig. Proc.*, Vol. 34, No. 1, pp. 152-156, February 1986.

[47] H. V. Sorensen, D. L. Jones, C. S. Burrus and M. T. Heideman, "On Computing the Discrete Hartley Transform," *IEEE Trans. on ASSP*, Vol. 33, pp. 1231–1238, October, 1985.

[48] G. W. Stewart, Introduction to Matrix Computations, Academic Press, Orlando, FL, 1973.

[49] P. L. Tyack, W. J. Williams and G. Cunningham, "Time-Frequency Fine Structure of Dolphin Whistles," *IEEE International Symposium on Time-Frequency and Time-Scale Analysis*, pp. 17–20, October 1992.

[50] P. P. Vaidyanathan, Multirate Systems and Filter Banks, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[51] C. Van Loan, Computational Frameworks for the Fast Fourier Transform, SIAM, Philadelphia, 1992.

[52] E. F. Vilez and H. Garudadri, "Speech Analysis Based on Smoothed Wigner-Ville Distribution," in Time-Frequency Signal Analysis: Methods and Applications, ed. B. Boashash, John Wiley & Sons, New York, 1992.

[53] V. Ville, "Sur la Notion de Signal Analytique," Cables et Transmissions, tome 2, No. 1, pp. 61-74, 1948.

[54] L. B. White, "Transition Kernels for Bilinear Time-Frequency Distributions," *IEEE Trans. on Sig. Proc.*, Vol. 39, No. 2, pp. 542-544, February 1991

[55] E. Wigner, "On the Quantum Correction for Thermodynamic Equilibrium," Phys. Rev., 40 (1932) 749-759.

[56] W. J. Williams and J. Jeong, "Reduced Interference Time-Frequency Distributions," in Time-Frequency Signal Analysis: Methods and Applications, B. Boashash (ed.), Longman Cheshire, Melbourne, Australia, 1992.

[57] Zak, J., "Finite Translations in Solid State Physics," *Phys. Rev. Lett. 19*, pp. 1385–1397, 1967.

Vita

Captain John R. O'Hair was born at West Point, New York, on 16 November 1961. He graduated from Butte Central High School in Butte, Montana in May of 1980, and entered the United States Air Force Academy the following June. He graduated with a Bachelor of Science in Electrical Engineering in 1984 and was assigned to Headquarters Armament Division as an Intelligence Analyst. During the next two years, he performed technical analysis of radar and Surface-to-Air Missile systems and attended night school at the University of West Florida (UWF). He earned a Masters of Business Administration from UWF in 1986. That same year, he entered Texas Tech University in Lubbock, Texas, under the auspices of the Air Force Institute of Technology's Civilian Institute Program. He earned a Master of Science in Electrical Engineering from Texas Tech University in 1987 in the area of pulse power solid state devices. Upon graduation, he was assigned to the Foreign Technology Division at Wright-Patterson, AFB, Ohio, where he was responsible for the procurement of signal processing and analysis systems. He entered the Air Force Institute of Technology School of Engineering in July 1991.