PROCEEDINGS
SPIE—The International Society for Optical Engineering

Real Time Signal Processing XI

J. P. Letellier
Chair/Editor

18–19 August 1988
San Diego, California

Sponsored by
SPIE—The International Society for Optical Engineering

Cooperating Organizations
Applied Optics Laboratory/New Mexico State University
Center for Applied Optics Studies/Rose-Hulman Institute of Technology
Center for Applied Optics/University of Alabama in Huntsville
Center for Electro-Optics/University of Dayton
Center for Optical Data Processing at Carnegie Mellon University
Georgia Institute of Technology
Institute of Optics/University of Rochester
Optical Sciences Center/University of Arizona

Published by
SPIE—The International Society for Optical Engineering
P.O. Box 10, Bellingham, Washington 98227-0010 USA
Telephone 206/676-3290 (Pacific Time) • Telex 46-7053

Volume 977

SYMBOLIC AND NUMERIC REAL-TIME SIGNAL PROCESSING

John S. Baras
Electrical Engineering Department
and
Systems Research Center
University of Maryland
College Park, MD 20742

Abstract

We consider real-time sequential detection and estimation problems for non-gaussian signal and noise models. We develop optimal algorithms and several architectures for real-time implementation based on numerical algorithms, including asynchronous implementations of multigrid algorithms. These implementations are of high complexity, costly and cannot easily accomodate model variability. We then propose and analyze a different class of algorithms, which are symbolic, of the neural network type. The preliminary results presented here demonstrate that these algorithms have remarkably lower complexity and cost, work well under model variability and their performance is nearly optimal. We also discuss how these type of algorithms are incorporated in the DELPHI system for integrated design of signal processing systems.

1. Introduction

One of the basic activities of electrical engineering today is the processing of signals, be they in the nature of speech, radar, images, or of electromechanical or biological origin. By "processing" we generally mean conversion of the signals into some more acceptable format for analysis. Examples could be the reduction of noise content, parameter estimation, band-pass filtering, or the enhancement of contrast, as required for imaging systems. The signal theorist develops algorithms for performing these functions by constructing mathematical models of signals and the operations conducted on them. The result of this work over the last twenty years has been a rather sophisticated theory, which utilizes advanced concepts from stochastic processes, differential equations and algebraic system theory. For a survey of such work the reader is referred to [1], where it can be seen that researchers have gone far beyond the classic work of Doob [2] and Wong [3]. Much of this theory is inaccessible to "mainstream" signal processing engineers, who often deem such research impractical, due to its apparent analytical intractability, algorithmic complexity, and difficult numerical implementation.

One of the reasons for the lack of impact of the more theoretical work on the field has been the failure to meet economic as well as real-time processing constraints imposed by the problems engineers face. The electronic circuitry needed to perform the kind of advanced algorithms required by theory must still be cost-competitive with existing techniques before workers in the field will consider the trade-off between paying more for what could be only a few percentage points in improved accuracy. Even if a problem is shown to be more readily and accurately solved by methods, such as, say, Lie algebras or differential geometry, it usually turns out to be a highly specialized case. So limited demand often precludes the introduction of expensive computational techniques. In general, engineers will settle for cheap but suboptimal *ad hoc* methods of their own invention, rather than master totally new concepts.

The other issue to be resolved for at least a wide class of signal processing problems involves meeting the time constraints implicit in the design. The problem is that such techniques will have much greater demands for their numerical analysis. As this translates to mean a greater number of arithmetic operations per second, meeting real-time processing conditions will be all the more difficult. And in addition to this, the necessary electronic components must be kept small in size as well as durable and reliable.

This is indeed the trend throughout much of signal processing: a greater volume of signals must be processed in a lesser amount of time, in addition to requiring more sophisticated analysis and relatively inexpensive electronics packaged on a small scale. To better understand these issues, we should examine in more detail the nature of some of these advanced techniques of signal processing. In addition, advances in computer-aided design tools for application specific microelectronic chips, have created opportunities for an integrated design environment, from the system level (where one performs statistical test on time series data, model building and algorithm development) to technology selection and implementation. The DELPHI system [4] is such a system level design tool for real-time signal processing. Finally, we show

Research supported by NSF Engineering Research Centers Program NSFD CDR 8803012
and by a grant from Texas Instruments, Incorporated.

consider carefully trade-offs resulting from conventional numerical algorithms, and non conventional symbolic (associative type) algorithms. This type of analysis will become critical in the future as designers begin to assess model accuracy on device performance, cost of production and other factors.

2. Key problems

The typical problem considered in this paper is described below. There are two hypotheses H_1, H_2 each representing that the observed data $y(t)$ originate from two different models

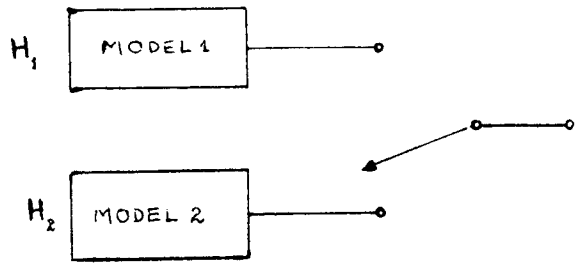


Figure 1. The typical sequential detection problem.

as shown in figure 1 below. The decision maker receives the data $y(t)$ and has to decide which of the two hypotheses is valid.

This problem is generic to a plethora of digital and analog signal processing problems, such as: pulse amplitude modulation, delta modulation, adaptive delta modulation, speech processing, direction finding receivers, digital phase lock loops, adaptive sonar and radar arrays, simultaneous detection and estimation.

To fix ideas consider the example of radar data, where one wishes to discriminate whether the received data are due to a ship target or to a decoy like a chaff cloud. Appropriate models have been developed by us [5, 6] for the pulse by pulse radar return as samples (sampled at the interpulse period) from the model

$$\begin{aligned} dx(t) &= \alpha x(t)dt + \beta dw(t) \\ dy(t) &= \exp(x(t))dt + dv(t) \end{aligned} \tag{1}$$

in the case of ship data, where $x(t)$ is scalar and $w(t), v(t)$ Wiener processes, while α, β are parameters of the model, controlling correlation time. A similar model can represent received pulse-by-pulse data from "chaff clouds" as samples from the model

$$\begin{aligned} dx(t) &= Ax(t) + Bdv(t) \\ dy(t) &= \|x(t)\|dt + dv(t) \end{aligned} \tag{2}$$

where $x(t)$ is two dimensional, A, B diagonal with identical elements, $\|\cdot\|$ is the Euclidean 2-norm. This discrimination problem can be thought of as the sequential discrimination between lognormal (1) and Rayleigh (2) data.

There are of course many problems of this type. As a matter of fact almost any sequential detection problem can be formulated in a similar manner. The underlying mathematical models can be diverse: diffusion processes, point processes, mixed processes, Markov chains etc. In this paper we shall concentrate on diffusion process models. That is to say, under each hypothesis the model for the observed data is

$$\begin{aligned} dx^i(t) &= f^i(x^i(t))dt + g^i(x^i(t))dw^i(t) \\ dy(t) &= h^i(x^i(t))dt + dv(t) \end{aligned} \tag{3}$$

where $i = 1, 2$ correspond to hypotheses H_1 or H_2. If we let

$$\hat{h}_i(t) = E_i\left\{h^i(x^i(t))|\mathcal{F}_t^y\right\} \tag{4}$$

the likelihood ratio for the problem is

$$\begin{aligned} \Lambda_t = \exp&\left(\int_0^t (\hat{h}_1(s) - \hat{h}_2(s)\right)^T dy(s) \\ &- \frac{1}{2}\int_0^t \left(\|\hat{h}_1(s)\|^2 - \|\hat{h}_2(s)\|^2\right) ds \end{aligned} \tag{5}$$

In [7] we showed that the optimal sequential detector utilizes threshold policies under both Neyman-Pearson and Bayes formulations and the likehood ratio Λ_t.

First it is clear that the detector has to select two things. A time τ, to stop collecting data, and a decision δ which declares one of the two hypotheses. Given the miss and false alarm probabilities $\bar{\alpha}, \bar{\beta}$ one computes thresholds A, B [7] and then the optimal detection strategy is given by

$$\tau^* = \inf\{t \geq 0 | \Lambda_t \notin (A, B)\} \tag{6}$$

$$\delta^* = \begin{cases} 1, & \Lambda_{\tau^*} \geq B \\ 2, & \Lambda_{\tau^*} \leq A. \end{cases} \tag{7}$$

This is shown graphically in figure 2.

It is therefore clear that real time implementation of this rule is based on our ability to compute $\hat{h}(t)$ in real time. This is related to the so called Zakai equation of nonlinear filtering [1]. This is so because

$$\hat{h}^i(t) = \frac{\int u^i(x,t)h^i(x)dx}{\int u^i(x,t)dx} \tag{8}$$

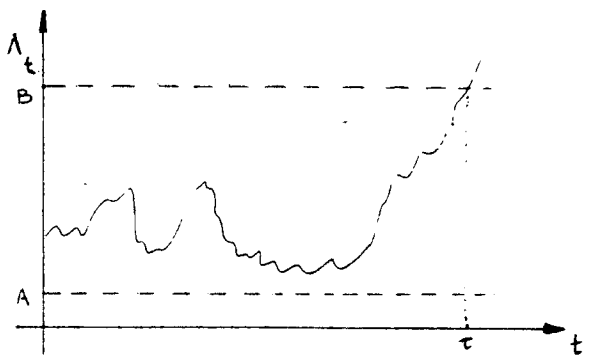


Figure 2. Optimal sequential detection rule.

where $u^i(x,t)$ is the unnormalized conditional probability density of $x(t)$ given $y(s), s \leq t$, under each model 1, or 2. This density satisfies the stochastic partial differential equation

$$
\begin{aligned}
du^i(x,t) &= L_i^* u^i(x,t)dt + u^i(x,t)h^{i^T}(x)dy(t) \\
u^i(x,0) &= p_0^i(x) \\
L_i^* u^i(x,t) &= \sum_{k,l} \frac{\partial^2}{\partial x_k \partial x_l}(\sigma^i_{k\ l} u^i(x,t)) - \\
&\quad - \sum_k \frac{\partial}{\partial x_k}(f_k^i(x)u^i(x,t)) \\
\sigma^i(x) &= \frac{1}{2} g^i(x) g^{i^T}(x).
\end{aligned} \tag{9}
$$

It can be shown that the likelihood ratio (5) can be represented as

$$
\Lambda_t = \frac{\int u^1(x,t)dx}{\int u^2(x,t)dx}. \tag{10}
$$

As a consequence the real-time implementation issue, is reduced to the real time implementation of (9), (10), by a digital circuit. In [7] we described a special architecture, which can achieve real-time operation for many applications, utilizing systolic arrays. We emphasized in [7] that this architecture solves the problem for dimensions of x^i, less than or equal to 2. The higher dimensional problems are not addressed in [7]. In the next section we provide a solution to the higher dimensional problem based on the so called multigrid method applied to (9).

3. Multigrid Algorithms for Zakai Equations.

In this section we show how multigrid algorithms [10, 11] can be developed for the Zakai equation (9) in a systematic manner. In particular we employ an implicit full discretization scheme that provides consistent time and space discretizations of the Zakai equation, in the sense that for each choice of discretization mesh, the problem can be interpreted as a nonlinear filtering problem for a discrete time, discrete state, hidden Markov chain.

To discretize (9), we choose a time step Δ and a space discretization mesh of size ϵ which determines a "grid" in $\mathbb{R}^d$, the space where we wish to solve the Zakai equation. In other words $x \in \mathbb{R}^d$. Using results on estimates of the tail behavior of $u(x,t)$ as $\|x\| \to \infty$. we can actually select a rectangular domain in $\mathbb{R}^d, D$, where we are primarily interested in solving (9). Let us suppose that there are $n(\epsilon)$ points on each dimension of the grid of size ϵ. Let $G_d(\epsilon)$ denote the hypercube generated in $\mathbb{R}^d$ by the spatial mesh of size ϵ. We assume for simplicity here uniform grid spacing.

Given this set-up we can construct following the methods of Kushner [12] a matrix $A(\epsilon)$ which approximates the operator L^* in (9), in the sense that $A(\epsilon)$ defines an approximating Markov chain to the diffusion (9). Let

$$
\begin{aligned}
\Delta y(k) &= y((k+1)\Delta) - y(k\ \Delta) \\
H_i(\epsilon) &= h(x_i(\epsilon))
\end{aligned} \tag{11}
$$

where $x_i(\epsilon)$ is a generic point on the grid $G_d(\epsilon)$. Finally let $\mathbf{V}^{k+1}(\epsilon)$ be the vector of samples $u(x_i(\epsilon),(k+1)\Delta)$ of the unnormalized conditional density over the grid $G_d(\epsilon)$. In [7] we proved the following, using semigroup techniques.

Theorem 1: Let

$$
\mathbf{D}(\epsilon,k) = \text{ diag } \{\exp(H_i^T(\epsilon)\Delta y(k) - \frac{1}{2}\|H_i(\epsilon)\|^2\Delta)\}
$$

and consider the implicit iteration

$$
\begin{aligned}
\frac{(I - \Delta A(\epsilon))\mathbf{V}^{k+1}}{(\epsilon)} &= D(\epsilon,k)V^k(\epsilon) \\
V^o(\epsilon) &= \{p_0(x_i(\epsilon))\}.
\end{aligned} \tag{12}
$$

Then as $k\Delta \to t$, with $\epsilon \to 0$ (along some sequence)

$$
\lim_{\epsilon \to 0} \sup_{i \in G_d(\epsilon)} |V_i^k(\epsilon) - u(x_i(\epsilon), k\Delta)| = 0. \tag{13}
$$

In other words Theorem 1, provides a uniformly convergent uniform scheme. Once we have this the likelihood ratio (10) can be easily approximated since

$$
\int u(x,t)dx \sim \sum_{i \in G_d} V_i^k(\epsilon)\Delta x(\epsilon); \text{ for } k\Delta \le t < (k+1)\Delta \tag{14}
$$

where $\Delta x(\epsilon)$ is the approximation to the volume element in $G_d(\epsilon)$.

Therefore the real-time solution of (9) has been reduced to the analysis of the real-time computation of (12). We note that the matrix $A(\epsilon)$ does not depended on time, and that the only part of (12) which depend on the real time data is the diagonal matrix $D(\epsilon,k)$.

We shall use in the sequel certain important properties of the matrix $A(\epsilon)$ which we now wish to touch upon briefly. For further details we refer to [8], [9]. Let $G_d(\epsilon)$ be a sequence of hypercubes (where we have varying dimension d) with $n^d(\epsilon)$ points ($n(\epsilon)$ being fixed). We consider the matrix $I - \Delta A(\epsilon)$ on $G_d(\epsilon)$ and $G_{d+1}(\epsilon)$. There is a convenient way to label the states of the resulting Markov chain so as to have some recursion between these two representations. Indeed let the two matrices be denoted as Γ_d and Γ_{d+1} respectively. Then [8]

$$
\Gamma_{d+1} = \begin{bmatrix} \Gamma_d & T & 0 & \cdots & 0 \\ T & \Gamma_d & T & \ddots & 0 \\ 0 & T & \ddots & \Gamma_d & T \\ 0 & \cdots & 0 & T & \Gamma_d. \end{bmatrix} \tag{15}
$$

where T is a tridiagonal matrix with positive entries. Γ_{d+1} is an $n \times n$ block matrix. Furthermore it is straightforward to establish [8, 9] that for any d Γ_d is strongly diagonally dominant. Furthermore $I - A(\epsilon)\Delta$ has finite bandwidth [8]. The strong diagonal dominance of $I - \Delta A(\epsilon)$ implies that we will need no pivoting. As we shall see this property will help also in the selection of the relaxation scheme in the multigrid iteration.

The fundamental idea of multigrid (MG) algorithms is relatively easy to understand. Let us suppose that we want to solve a partial differential equation (p.d.e) in two dimensions (i.e. on the plane). We now introduce two grids (as opposed to one). The finer grid $G_1(\epsilon)$, with uniform mesh spacing ϵ where an approximate solution u_i to the discretized p.d.e.

$$
L^1 u_1 = f_1 \tag{16}
$$

is given. Suppose U_1 was the true solution of (16). If we consider the difference $u_1 - U_1$, in the Fourier domain, it will have substantial high frequency error. To reduce this we conduct a few *relaxation* sweeps on the fine grid. Let the smoothed approximate solution on the fine grid, resulting after several relaxation sweeps be denoted by $\tilde{u}_1$. The value of the solu-

tion after smoothing can be measured by examining the *defect equation*

$$d_1 = f_1 - L^1\tilde{u}_1. \tag{17}$$

The latter is derived by considering

$$v_1 = -u_1 + U_1$$

and observing that $L^1U_1 = f_1$, so that

$$L^1v_1 = -L^1u_1 + L^1U_1 = f_1 - L^1u_1. \tag{18}$$

So we can consider the computation of $\tilde{u}_1$ as solving

$$d^1 = L^1v_1 \tag{19}$$

in such a manner as to make d^1 as small in norm as possible. Gauss-Seidel, Jacobi, etc. are common relaxation methods for affecting this smoothing. Because of this smoothness the defect equation can be transferred to a *coarser grid* without loss of too much information, since the highly oscillatory error components have died down. Let G_2 denote this coarser grid, where we consider the defect equation

$$d_2 = L^2v_2. \tag{20}$$

The correction v_2 can be found by solving (20) which is less computationally expensive. Clearly one can consider multiple grids by iterating the above process. Now once the correction is computed on G_2, it is transferred back to the finer grid by interpolation, which yields v_1 and the sum $v_1 + \tilde{u}_1$ is used as a starting point for more smoothing. This is the basic idea of *nested iteration*: the use of coarser grids to obtain good initial approximations for relaxations on finer grids.

The primary reasons for using MG methods are as follows. Direct solvers of discretized p.d.e's have computation time that grows linearly with n, the width of the finest grid, while MG methods can actually do much better than this as we shall see. What is perhaps more important, direct solvers take longer to solve problems in higher dimensions than do the methods described here.

As for relaxation schemes, slow convergence is a typical problem, although they are perfectly suited for parallel implementation, as they rely only on "local" information when a sweep is performed. Thus, relaxation schemes have a computation time that is independent of the size of the grid. In Multigrid algorithms, such schemes are used only for smoothing the high-frequency error as a prelude to intergrid transfers, which we note, can also be done in parallel. The Multigrid algorithm is therefore an attempt to preserve the highly parallel structure of relaxation algorithms, while overtaking their slow convergence rates by reducing the original linear equations to systems of lower dimensions. The only direct solving to be performed in the Multigrid algorithm is on the coarsest grid which can be made as small as we like, at the cost of increased grid levels. Because of its naturally parallel properties, it turns out that the Multigrid method has a computation time that is essentially independent of the dimension of the problem. Because we wish to compute in real-time, such a numerical method is an ideal candidate for investigation.

We now described a one-cycle full Multigrid algorithm program. Let there be K point-grids which we will denote by $G_1, G_2, \ldots, G_K$ with the finest being G_K and the coarsest being G_1.

The finest grid contains the problem:

$$L^KU^K = f^K. \tag{21}$$

Smoothing Part I:

Given an initial approximation to the problem in (21), smooth j_1 times to obtain u^K.

Coarse-grid correction:

Compute the residual $d^K = f^K - L^Ku^K$.

Inject the residual into the coarser grid G_{K-1},

$$d^{K-1} = I_K^{K-1}d^K.$$

Compute the approximate solution $\tilde{v}^{K-1}$ to the residual equation on G_{K-1}:

$$L^{K-1}v^{K-1} = d^{K-1}, \tag{22}$$

by performing $c \geq 1$ iterations of the Multigrid method, but this time we will be using the grids $G_{K-1}, G_{K-2}, \ldots, G_1$ applied to equation (22).

Interpolate the correction $\tilde{v}^K = I_{K-1}^K\,\tilde{v}^{K-1}$.

Compute the corrected approximation on G^K,

$$u^K + \tilde{v}^K. \tag{23}$$

Smoothing Part II:

Compute a new approximation to U^K by applying relaxation sweeps to $u^K + \tilde{v}^K$.

The recursive structure of the algorithm entering just after eq. (22) is apparent. Here the algorithm simply repeats itself, so in the case of $c = 1$ we have initial smoothing, computation of residual equation, injection to coarser grid, all until the coarsest grid is reached, where the equation is directly solved. Then we have interpolation upward through the grids, offering each finer grid an approximation for relaxation. This would be a "V-shape" structure as opposed to a "W-shape struture [8, 9]. Only if $c > 1$, would we have a "W shape" structure. Of course it is clear that there can be many variants of the algorithm, depending on the number of interpolation and injection operations.

The convergence of the multigrid method is based on the following representation of the algorithm. For the general grid G_k we will have the equation

$$L^kU_k = f_k \quad , k = 1, \ldots, K. \tag{24}$$

The formula for obtaining a new approximation to the solution U_k from the old one u_k can be written as

$$\begin{aligned}\tilde{u}_k &= (I - ML^k)u_k + Mf_k \\ &= S_ku_k.\end{aligned} \tag{25}$$

Here S_k is the smoothing operation on the grid G_k, and we assume that M is invertible and the smoother is consistent. With this notation S_k^j denotes the smoother that uses j relaxation sweeps, or is applied j times.

As examples of smoothers, define D to be the matrix whose diagonal entries are equal to those of L^k, and which is zero everywhere else. Then $m = \omega D^{-1}$ is the modified Jacobi method. If T is the "upper triangular part" of L^k, and zero elsewhere, then we have the Gauss-Seidel method by setting

$M = T^{-1}$. In fact, M is usually some approximation to the inverse of L^k, which forces $\rho(I - ML^k)$ to be close to zero.

We already have the interpolation (coarse to fine) and injection (fine to coarse) operators: I_{k-1}^k and I_k^{k-1} respectively. We also define I_k to be the identity operator on grid G_k.

By constructing the "Multigrid operator" we can show that, like any other iterative process, convergence is guaranteed under certain conditions.

Given u^k as the old approximation, the new approximation $\tilde{u}^k$ will be

$$\tilde{u}_k = M_k\, u_k + I_{k-1}^k (L^{k-1})^{-1}\, I_k^{k-1} f_k. \tag{26}$$

M_k will be the Multigrid operator on grid G_k we will concentrate on, for it is its spectral radius that determines whether the iteration converges or not. By M_k^c we will mean c multiples of the MG operator applied on the k grids. The following recursion will define this operator, which begins at grid level 2 and proceed up to $k = K - 1$ where we have K grid levels in all:

$$\begin{aligned} M_2 &= S_2^{j_2}(I_2 - I_1^2 (L^1)^{-1}\, I_2^1 L^2)\, S_2^{j_1} \\ M_{k+1} &= S_{k+1}^{j_2}(I_{k+1} - I_k^{k+1}(I_k - M_k^c)(L^k)^{-1}\, I_{k+1}^k L^{k+1})\, S_{k+1}^{j_1}. \end{aligned} \tag{27}$$

These equations can be easily shown by induction.

Note that the coarsest grid G_1 uses a direct solver $(L^1)^{-1}$, (although it need not actually require this inverse; this is just operator notation.) Once again, note that we are only interested in that part of the formula in (27) that determines convergence; in particular, we must show $\rho(M_k) < 1$.

There is another way of writing the above that is useful in studying convergence properties. For $k = 2, 3, \ldots, K-1$ and K grid levels, let

$$\begin{aligned} M_k^{k-1} &= S_k^{j_2}(I_k - I_{k-1}^k (L^{k-1})^{-1}\, I_k^{k-1} L^k) S_k^{j_1} \\ A_k^{k+1} &= S_{k+1}^{j_2}\, I_k^{k+1} : G_k \to G_{k+1} \\ A_{k+1}^k &= (L^k)^{-1}\, I_{k+1}^k (L^{k+1})^{-1}\, S_{k+1}^{j_1} : G_{k+1} \to G_k \end{aligned} \tag{28}$$

Thus we can write,

$$M_{k+1} = M_{k+1}^k + A_k^{k+1}\, M_k^c\, A_{k+1}^k. \tag{29}$$

Now if $\|M_{k+1}^k\|, \|A_k^{k+1}\|$ and $\|A_{k+1}^k\|$ for $k \le K-1$ are known, then one can obtain an estimate of $\|M_K\|$, where $\|\cdot\|$ represents any reasonable operator norm.

Theorem 2 [13]: Let the following estimate be assumed known for $k \le K - 1$,

$$\|M_{k+1}^k\| \le \sigma, \qquad \|A_k^{k+1}\| \cdot \|A_{k+1}^k\| \le C.$$

Then,

$$\|M_K\| \le \nu_K, \tag{30}$$

where ν_K is recursively defined by

$$\nu_2 = \sigma, \quad \nu_{k+1} = \sigma + C(\nu_k)^c, \quad k = 2, 3, \ldots, K-1. \tag{31}$$

We have not described how to obtain a value for c. We could simply set c to be a function of k, and this is often done. The first case is simply a constant

$$c = 2, \qquad \text{for } k = 2, 3, \ldots, K-1 \tag{32}$$

which yields the W cycle. The second case makes c dependent on k,

$$c_k = \begin{cases} 1, & k \text{ odd} \\ 2, & k \text{ even} \end{cases} \tag{33}$$

Theorem 3: [13]: For the case of eq. (32) if $4C\sigma \le 1$

$$\|M_K\| \le \nu = \left(1 - \sqrt{1 - 4C\sigma}\right)/2C \le 2\sigma, \quad K \ge 2, \tag{34}$$

and for eq. (33), if $4C^2(1+C)\sigma \le 1$,

$$\|M_K\| \le \begin{cases} \left(1 - \sqrt{1 - 4C^2(1+C)\sigma}\right)/2C^2 \\ \qquad \le 2\sigma(1+C), \quad (K \text{ even}) \\ \left(1 - 2C^2\sigma - \sqrt{1 - 4C^2(1+C)\sigma}\right)/2C^3 \\ \qquad \le \sigma(1+2C)/C, \quad (K \text{ odd}) \end{cases} \tag{35}$$

If $c = 2$ for all k, which implies that W-cycles are used, and if σ is small enough, then $\nu \approx \sigma$ for the bound in eq (34). For example, if $C = 1$, then (34) yields,

$$\nu \le 0.113 \quad \text{if} \quad \sigma \le 0.1$$

Typically, $C \ge 1$, but not very large. Further insights along this line show that if a problem on a given 2-grid method converges sufficiently well for small enough σ, then the corresponding multigrid method with $c = 2$ will have similar convergence properties. Thus, MG practitioners have found that for reasonable problems, one need only analyze the 2-grid method and assume the results hold for the general multigrid case. Also, there appears no need to work with $c > 2$.

For the case of interest here, i.e. the discretization of the Zakai equation (12), following well known methodology for MG application we first identify a simpler, albeit characteristic problem. This is the problem with no input, i.e. when the right hand side of (12) becomes $V^k(\epsilon)$. In other words if one understands how MG is applied to the discretized Fokker-Planck equation

$$(I - \Delta A(\epsilon))\, V^{l+1}(\epsilon) = V^l(\epsilon), \tag{36}$$

then complete understanding of the application of MG to the Zakai equation is straightforward.

It follows [8, 9] that the spectral radius of the associated MG generator (see (27, 28)) is determined entirely by the matrix $I - \Delta A(\epsilon)$, along with the choice of relaxation scheme. Also note that because $I - \Delta A(\epsilon)$ is not time dependent all *program parameters are precomputable.* In particular for the Zakai equation, they do not depend on the sample path $y(\cdot)$.

A highly recommended relaxation method in MG applications is the so called *successive overrelaxation method* (SOR). To define it suppose one wants to solve

$$Ax = b$$

with $a_{ii} \neq 0$. Then define B to be the $n \times n$ matrix

$$b_{ij} = \begin{cases} -a_{ij}/a_{ii}, & i \neq j \\ 0, & i = j \end{cases}$$

and define the vector c in $\mathbb{R}^n$ to have components,

$$c_i = b_i/a_{ii}.$$

Then let us consider the $L - U$ decomposition of B

$$B = L + U$$

where L is strictly lower and U is upper triangular matrix. Choose a real number w, and define the iteration

$$x_{n+1} = \omega(Lx_{n+1} + Ux_n + c) + (1 - w)x_n. \qquad (37)$$

This is the SOR method. If $\omega = 1$, the SOR method reduces to the Gauss-Seidel method, with $\omega > 1$ implying overcorrecting, and $\omega < 1$ implying undercorrecting.

Recall that the matrix $I - \Delta A(\epsilon)$, for the discretized Zakai equation, is strongly diagonally dominant. Furthermore this matrix is also an L-matrix, i.e. it has positive diagonal elements and non positive off diagonal elements. Finally this matrix is *consistently ordered* [8, 9]. This is a consequence of the natural ordering on a rectangular grid. One can measure the properties of smoothing operators with a variety of measures [8]. So one can describe "optimal" smoothing operation. We thus have [8, 9].

Theorem 4: Because of the properties of $I - \Delta A(\epsilon)$, the MG operator converges. The optimal relaxation scheme for the Zakai equation is the SOR method. There is an optimal choice for ω in (37) with respect to convergence as well.

4. Architectures for Implementing MG in Real-Time.

In this section we analyze the complexity of the MG schemes described in section 3, in particular with respect to real time implementation. We shall see that the result is a multilayer processor network. Here the processors and the interconnections are more complicated than the ones used in the systolic architecture of [7]. So fabrication may be a problem.

The computing network will be a system of grids of identical processing elements. Therefore, we have two kinds of grids, one of points and one of processors, and these will be layered one on top of another. For each $1 \leq k \leq K$, processor grid P_k has $(n_k)^\gamma$ elements, where γ is a positive integer not greater than the problem dimension d. Also, we have $n_K = n$, and $n_i < n_j$, if $i < j$.

Similarly, in keeping with the above notation, there are, for each $1 \leq k \leq K$ a corresponding point grid G_k with $(n_k)^d$ points. (Note that the number of processors per grid is never greater than the number of points). Again we have $n_K = n$ while $n_i < n_j$ if $i < j$. A key assumption, which is quite realistic, is that for each step of the multigrid algorithm on point grid G_k, the processing grid P_k requires $O((n_k)^{d-\gamma})$ time to perform its computations.

As in the last section, we also have injection and interpolation operators, I_k^{k-1} and I_{k-1}^k respectively. We will also be needing the operator M_k later in this chapter.

An important consideration is the notion of ***speedup*** and *efficiency*. With any given design, one would hope that the addition of processors, which might be drafted for the exploitation of parallelism, would lead to a decrease of computation time, or speedup, of the algorithm. We therefore define efficiency to be the ratio of speedup achieved to the number of processors employed. If this ratio remains bounded from below as the width of point grids, n, tends to infinity, then the design is said to be efficient. Recall that the number of processors is a function of n, since processor grid P_k has n^γ elements. Chen and Schreiber (14) showed that when $\gamma < d$, some algorithms can be implemented efficiently. But when $\gamma = d$, which is the most parallelism one can reasonably expect to use, no algorithm can be implemented efficiently. To design a parallel machine capable of performing the MG algorith, we assume our problem is in d dimensions over a rectangular domain using a regular point grid of n^d points. We further have

$$\begin{aligned} n_K &= n \\ n_{k+1} &= a(n_k + 1) - 1, \quad k = 1, 2, \ldots K - 1 \end{aligned} \qquad (38)$$

for some integer $a \geq 2$. We map grid points in such a way that neighboring grid points reside in the same or neighboring processors.

Smoothing sweeps of at least some type can be accomplished in $O(n^{d-\gamma})$ time with this given connectivity; (we will give more details on this in the next chapter.) Let t be the time taken by a single processor to perform the operations at a single gridpoint that, done over the whole grid, constitute a smoothing sweep. Then, setting S as the time needed to perform the smoothing sweep over the whole of grid G_k on processor grid P_k, we have,

$$S = t\, n_k^{d-\gamma}. \qquad (39)$$

Obviously, it is to our advantage to conduct as few smoothing sweeps as necessary and still assure sufficient accuracy.

Now processor grid P_k is connected to processor P_{k+1}. Processor $i \in P_k$ is connected to processor $a(i+1) - \mathbf{1} \in P_{k+1}$ where $\mathbf{1} = (1, 1, \ldots, 1)$. These connections allow any intergrid operations, such as interpolation, to be performed in $O(S)$ time. Now define the system of processor grids $\{P_1, P_2, \ldots, P_J\}$ as the machne M_J for $J = 1, 2, \ldots, K$. Then the execution of performed by M_k proceeds as follows:

1. First, j smoothng sweeps on grid G_k are done by P_k; all other processor grids idle.
2. The coarse grid equation is formed by P_k and transferred to P_{k-1}.
3. MG is iterated c times on grid G_{k-1} by M_{k-1}. P_k is idle.
4. The solution v^{k-1} is transferred to P_k by iterpolation: $I_{k-1}^k\, v^{k-1}$.
5. The remaining m smoothing sweeps are done by P_k.

Now we let $W(n)$ be the time needed for steps $1, 2, 4, 5$ and find

$$W(n) = (j + m + s)\, t\, n^{d-\gamma}, \qquad (40)$$

where s is the ratio of the time needed to perform steps 2 and 4 to the time needed for one smoothing sweep. Note that s is independent of n, d and γ.

We discuss now the time complexity of MG. We will denote by $T(n)$ the time complexity of MG algorithm on a grid of n^d points. It turns out that $T(n)$ solves the recurrence:

$$T(an) = cT(n) + W(an), \tag{41}$$

where $W(an)$ denotes the work needed to pre-process and post-process the (an)-grid iterate before and after transfer to the coarser n-grid. In effect the term $W(an)$ includes the smoothing sweeps, the computation of the coarse grid correction equation (i.e., the right-hand side d^{k-1}) and the interpolation back to the fine grid ($I^k_{k-1}\, v^{k-1}$). Then we have

Theorem 5, [14]: Let $T_p(\cdot)$ be a particular solution of (41), i.e.,

$$T_p(an) = c\, T_p(n) + W(an).$$

Then the general solution of (41) is:

$$T(n) = \alpha\, n^{\log_a c} + T_p(n), \tag{42}$$

where α is an arbitrary constant. Using this result we have the general solution to (42),

$$T(n) = \begin{cases} \beta(a^p/(a^p - c))n^p & \text{if } c < a^p, \\ \beta n^p \log_a n + O(n^p) & \text{if } c = a^p, \\ O(n^{\log_a c}) & \text{if } c > a^p. \end{cases} \tag{43}$$

We see that it would take a single processor $O(n)$ steps to complete the above mentioned tasks on one dimension, while n processors could do the same for a two-dimensional problem in $O(n)$ time.

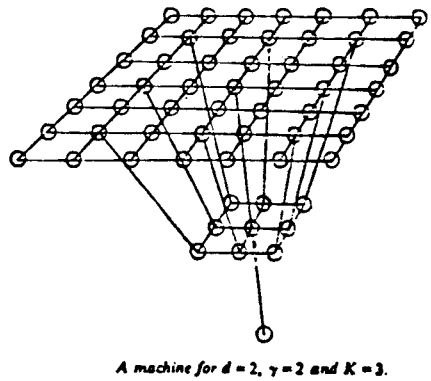
We say that the MG algorithm is of *optimal order* if $T(n) = O(n^{d-\gamma})$, a possibility that is sometimes precluded by some choices of c, a, γ and d, which in turn influence $T(n)$. Examination of (43) demonstrates the relations between the various parameters. As an example, in the one-processor case, with $\gamma = 0, d = 2$, we have $g(n) = n^2$. We then have an optimal scheme if $a = 2, c < 4$, for only then is $T(n) = O(n^2)$. But $c \geq 4$ is non-optimal, with $T(n) = O(n^2 \log n)$ for $c = 4$.

In general, we have an optimal scheme if and only if $c < a^d$.

There also exists a natural way to build a VLSI system to implement our algorithms. The $\gamma = 1$ machine can be embedded in two dimensions as a system of communicating rows of processors. The $\gamma = 2$ machine can be embedded in three dimensions as a system of communicating planes, and so on. Realizations in three-space will be possible in a natural way for any value of γ. Consider the case of $d = 2, \gamma = 2$. In this case, we have a set of homogeneous planar systolic arrays layered one on top of the other. If we let $a = 2, K = 3$, and $n_1 = 1, n_2 = 2(1+1) - 1 = 3, n_3 = 2(3+1) - 1 = 7$, we would have a 7×7 array on top of a 3×3 array which is then on top of a single processor corresponding to n_1, see figure 3 below.

Unfortunately, this design differs from the classical systolic array concept of Kung [15] in that there exists no layout in which wire lengths are all equal. Also, each layer of the system is homogeneous while the entire machine is clearly not. We might also remark that it is not necessary that the layers converge down to a single processor as in fig. 3. Instead, 3 or 4 levels of grids could be used and the multigrid method would still be highly efficient.

Now the four parameters c, a, γ, d are to be chosen with any implementation MG, and, of course, they are not unrelated to each other. Extending the earlier notation, we call any one choice of the four a design and denote its corresponding computing time by $T(c, a, \gamma, d)$. We will now begin with an examination of the trade-offs incurred by one choice over another. Following [14], an important issue is *efficiency*, E vs. *speedup* S in a particular design. We define,



A machine for $d = 2$, $\gamma = 2$ and $K = 3$.

Figure 3. Architecture for MG.

$$S(c, a, \gamma, d) = T(c, a, 0, d)/T(c, a, \gamma, d) \tag{44}$$
$$E(c, a, \gamma, d) = T(c, a, 0, d)/(P(\gamma)T(c, a, \gamma, d))$$

Note that the speedup S corresponds to the gain in speed going from the one-processor system to that of the multiprocessor. Whereas the efficiency E reflects the trade-off between using more processors vs. time. It thus is a measure of how efficiently a given architecture exploits any additional increase in the number of processors in the hope of improving speedup.

We say that a design $T(c, a, \gamma, d)$ is *asymptotically efficient* if E tends to a constant as $n \to +\infty$, and it will be *asymptotically inefficient* if $E \to 0$ asn $n \to +\infty$.

Theorem 6, [14]: Let $\gamma > 0$.

1) If $c < a^{d-\gamma}$ then $E(c, a, \gamma, d) = (a^\gamma - 1)(a^{d-\gamma} - c)/(a^d - c)$.

2) If $c = a^{d-\gamma}$ then $E(c, a, \gamma, d) = (a^\gamma - 1)a^{d-\gamma}/((a^d - c)\log_a n)$.

3) If $c > a^{d-\gamma}$ then

$$E(c, a, \gamma, d) = \begin{cases} O(1/n^{\log_a(c-d+\gamma)}) & \text{if } c < a^d \\ O((\log_a n)/n^\gamma) & \text{if } c = a^d \\ O(1/n^\gamma) & \text{if } c > a^d \end{cases}$$

We have at once that

1) *A design is asymptotically efficient if and only if* $c < a^{d-\gamma}$.

2) *The fully parallel design* $\gamma = d$, *is always asymptotically inefficient.*

3) "Halfway" between asymptotic efficiency and inefficiency is *logarithmic asymptotic efficiency*, with $E = O(\log n)$, as $n \rightarrow = \infty$. *A fully parallel design* ($\gamma = d$) *if logarithmically asymptotically efficient iff* $c = 1$.

4) If we start with a non-optimal design in the one processor case, then adding more processors will not make the design asymptotically efficient. This is because so many coarse grid corrections are being performed that if more processors are added so as to lower the set-up time when transferring to the coarser grids, we still would be losing too much time on the coarser grids.

To get $T(n) = O(n)$ we have to select $c = 1$.

We also have considered the concurrent iteration schemes of Gannon and Van Rosendale [16].

Thus the fully parallel architecture has a computation time of at most $O(\log n)$ and so it is very competitive with the systolic direct solver. More importantly, this time is largely independent of dimension d, at least for small values of d. Of course, increases in d will result in large increases in circuit layout area, due to an increase in interconnections between grids, and thus a subsequent loss in computing speed.

We can implement the SOR method in a parallel fashion, using the "red-block" or "checker board" method [8, 9]. In higher dimensions we need to utilize multicolor ordering. Employing the intrinsic locality of the SOR we can implement it asynchronously as well.

A detailed analysis of timing performed in [8, 9] demonstrates that if the real-time constraint for the Zakai equation is 1 msec, then we can realistically achieve real-time implementation with the multi-layered networks of this section, only for dimension $d \leq 8$. We note that this is quite an advance from the results of [7].

5. Neural Networks

The results presented in the previous sections can be utilized to provide solution to a variety of real-time signal processing problems of the detection and estimation type as shown in section 2. However the resulting implementations are very complex and therefore costly. Furthermore these designs are based on perfect knowledge of the model parameters f, g, h in (3). In reality, the models will be at best known with uncertainty. Although the methods of sections 2-4 can be used to handle parametric uncertainties, the complexity of the resulting designs will increase even further. In this section we describe an alternative methodology which appears more promising, particularly regarding complexity of resulting processors.

To make the exposition simpler, let us assume that both models discussed in section 1, 2 are Gaussian. Under either hypothesis the vector of observed data

$$\mathbf{y} = \{y(1), \ldots, y(N)\} \tag{45}$$

is Gaussian, with mean zero, and covariance matrix

$$\begin{aligned} \Sigma_{ij}^k &= E\{y_i\, y_j | H_k\} \\ &= c_k^T\, A_k^{|i-j|} R_k c_k + d_k^2 \end{aligned} \tag{46}$$

$$i, j = 1, \ldots, N \quad ; k = 1, 2$$

where A_k, R_k, d_k, c_k are the parameters of the underlying linear model

$$\begin{aligned} x_k(t+1) &= A_k x_k(t) + B_k v_k(t) \\ y(t) &= c_k^T x_k(t) + d_k w_k(t), \end{aligned} \tag{47}$$

and y, has been assumed scalar for simplicity. Hence the probability density of the data given hypothesis H_k is Gaussian

$$p(\mathbf{y}|H_k) = \frac{1}{(2\pi)^{N/2} (\det \sum_k)^{1/2}} \exp\left(-\frac{1}{2} \mathbf{y}^T \Sigma_k^{-1} \mathbf{y}\right). \tag{48}$$

Classical binary hypothesis testing theory, results to a decision region in $\mathbb{R}^N$ determined by the threshold rule:

$$\frac{p(\mathbf{y}|H_1)}{p(\mathbf{y}|H_2)} \underset{2}{\overset{1}{\gtrless}} A. \tag{49}$$

Using (48) we can rewrite (49) after taking logarithms as

$$ln\left(\frac{\det \Sigma_2}{\det \Sigma_1}\right)^{1/2} + \frac{1}{2} \mathbf{y}^T \left(\Sigma_2^{-1} - \Sigma_1^{-1}\right) \mathbf{y} \underset{2}{\overset{1}{\gtrless}} lnA \tag{50}$$

or

$$\mathbf{y}^T \left(\Sigma_2^{-1} - \Sigma_1^{-1}\right) \mathbf{y} + A_1 \underset{2}{\overset{1}{\gtrless}} 0. \tag{51}$$

The test (51) is a quadratic form positivity test. It can be thought of as a map from point $\mathbf{y}$ in $\mathbb{R}^N$ (our decision space) to the set $\{1, 0\}$ identifying the decisions for hypotheses H_1 or H_2 to be true respectively. We first want to point out that this map can be realized precisely as a three layer neural network.

To see this, observe that $\Sigma_2^{-1} - \Sigma_1^{-1}$ is symmetric so it can be diagonalized

$$\Delta = \Sigma_2^{-1} - \Sigma_1^{-1} = W^T D W. \tag{52}$$

Let

$$z = W\mathbf{y}, \quad z_i = \sum_{j=1}^{N} W_{ij}\, y_j. \tag{53}$$

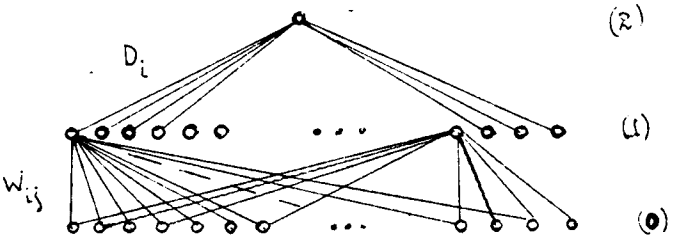


Figure 4. Neural network implementation

Then the test statistic in (51) becomes

$$\mathbf{y}^T W^T D W \mathbf{y} = (W\mathbf{y})^T D W \mathbf{y} = \sum_{i=1}^{N} D_i z_i^2 \tag{54}$$

So let

$$\xi_i = f(z_i) = z_i^2.$$

Consider now a three layer network as shown in figure 4. Here the data enter at the nodes of the (0) layer. Then

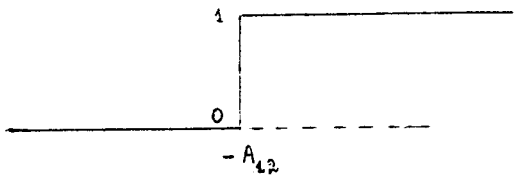
$$z(1) = W\mathbf{y} = \sum_{j=1}^{N} W_{ij} y(j) \tag{55}$$

$$\xi_i(1) = f(z_i(1)) = z_i^2 \tag{56}$$

$$z(2) = \sum_{i=1}^{N} D_i \xi_i(1) \tag{57}$$

$$\xi(2) = \tilde{f}(z(2)) \tag{58}$$

where $\tilde{f}$ is the function



Since we can implement the optimal decision rule by a neural network, it stands to reason to examine whether a more generic neural network can accomplish the task, and in particular provide adaptive behavior, when the model parameters Σ_1, Σ_2 are unknown. Note that in the above computations Σ_1, Σ_2 were assumed to be known.

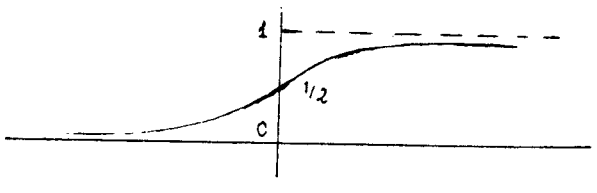
Let us then consider a generic neural network with three layers, N nodes in the (0) layer, M nodes in the intermediate layer. The diagram is similar to that of figure 4.

Let

$$\left.\begin{aligned} z_i(1) &= \sum_{j=1}^{N} W_{ij}(1) y_j \\ \xi_i(1) &= f(z_i(1)) \\ z_i(2) &= \sum_{j=1}^{M} W_{ij}(2) \xi_j(1) \\ \xi_i(2) &= f(z_i(2)) \end{aligned}\right\} \tag{59}$$

where f is the nonlinear function

$$f(z) = \frac{1}{1 + e^{-\beta z}} \tag{60}$$



We provide adaptation to this neural network, by a standard back-propagation algorithm. Let $t_i^{(n)}$ denote the correct output for the n^{th} training pattern, then the weight are adjusted by

$$\left.\begin{aligned} W_{ij}^{(n)}(l) &= W_{ij}^{(n-1)}(l) + \eta\, \delta_i^{(n)} \xi_j^{(n)}(l-1) \\ \delta_i^{(n)}(2) &= (t_i^{(n)} - \xi_i^{(n)}(2)) f'(z_i^{(n)}(2)) \\ \delta_i^{(n)}(l) &= f'(z_i^{(n)}(l)) \sum_j \delta_j^{(n)}(l+1) W_{ji}^{(n-1)}(l+1) \\ l &= 1, 2. \end{aligned}\right\} \tag{61}$$

In [17] we show that this generic network, performs nearly optimally without prior knowledge of the models. We also provide estimates of the time it takes to identity the models. The time performance of the network is quite satisfactory as well.

In [17] we have also extended these ideas to the non-Gaussian case described in (3). Here the data are $y(1), \ldots, y(N)$. Considering for simplicity scalar y and block detection, we know from our analysis of section (2) that the optimal algorithm is to compute V_1^N, V_2^N from (12), utilizing the two different models, corresponding to hypotheses H_1, H_2. Then the optimal test is provided by

$$\frac{\sum_{i=1}^{M} V_1^N(i)}{\sum_{i=1}^{M} V_2^N(i)} \underset{2}{\overset{1}{\gtrless}} A \tag{62}$$

or, in the case neither is satisfied increase N. Here M is the number of discretization points needed. We can with appropriate transformations visualize this algorithm as a neural network, with planar layers. This suggests again the consideration of a more generic neural network. At this time we have only preliminary results on this general case. The generic network suggested has three layers, each layer being a planar network. We shall describe these results elsewhere.

We are also incorporating these neural network algorithms in the DELPHI system [4].

References

[1] M. Hazewinkel and J.C. Willems, edts, "Stochastic Systems: The Mathematics of Filtering and Identification", Proc. of NATO Advanced Institute, Les Arcs, France, Dordrecht, The Netherlands: Reidel 1961.

[2] J. Doob, Stochastic Processes, Wiley, 1953.

[3] E. Wong, "Stochastic Processes in Engineering Systems", Springer-Verlag, 1985.

[4] J.S. Baras, F. Ebrahimi, B. Israel, A. LaVigna, and D.C. MacEnany, "The DELPHI System: A System Level Tool for Integrated Design of Real-Time Signal Processors", Proceedings of SPIE, Vol 827 Real Time Signal Processing X, (J.P. Letellier ed), Aug 1987, pp10-14.

[5] J.S. Baras, "Ship RCS Scintillation Simulation", Naval Research Laboratory Technical Report 8189, 1978.

[6] J.S. Baras, A. Ephremedies and G. Panayotopoulos, " Modeling of Scattering Returns and Discrimination of Distributed Targets", Technical Report, Electrical Engineering Department, University of Maryland, 1980.

[7] J.S. Baras and A. LaVigna,"Architectures for Real-Time Sequential Detection", Proceedings of SPIE, Vol 827 Real Time Signal Processing X, (J.P. Letellier ed), Aug 1987, pp100-105.

[8] K. Holley, "Applications of the Multigrid Algorithm to Solving the Zakai Equation of Nonlinear Filtering with VLSI Implementation", Ph.D. Thesis, University of Maryland, December 1986.

[9] J.S. Baras and K. Holley, "Parallel Architectures for Zakai Equations in Higher Dimensions", submitted for publication, 1988.

[10] S.F. McCormick, Edt, "Multigrid Methods", Frontiers in Applied Mathematics, SIAM 1987.

[11] W.L. Briggs, "A Multigrid Tutorial", SIAM, 1987.

[12] H.J. Kushner, "Probability Methods for Approximations in Stochastic Control and for Elliptic Equations", Academic Press, 1977.

[13] K. Stüben and V. Trottenberg, "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications", in *Multigrid Methods*, W. Hackbusch and W. Trottenberg (edts), Springer Verlag, 1982.

[14] T. Chen and R. Schreiber, "Parallel Networks for Multigrid Algorithms: Architecture and Complexity", SIAM J. Sci. Stat. Comput., Vol. 6, No. 3, July 1985.

[15] H.T. Kung, "Systolic Algorithms", in *Large Scale Scientific Computation*, Academic Press, 1984.

[16] D. Gannon and J. Van Rosendale, "Highly Parallel Multigrid Solvers for Elliptic PDE's: An Experimental Analysis", ICASE Report No. 82-36, Nov. 1982.

[17] J.S. Baras and A. LaVigna, "Neural Network Algorithms for Real-Time Detection", in preparation.