APPLICATIONS OF THE MULTIGRID ALGORITHM TO SOLVING THE ZAKAI EQUATION OF NONLINEAR FILTERING WITH VLSI IMPLEMENTATION

by

Kevin Holley

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1986

ABSTRACT

Title of Dissertation: Applications of the Multigrid Algorithm
To Solving the Zakai Equation of Nonlinear Filtering
With VLSI Implementation

Kevin Craig Holley, Doctor of Philosophy, 1986.

Dissertation directed by: Dr. John Baras, Professor, Department of Electrical Engineering and Applied Mathematics

The feasibility of designing nonlinear filtering algorithms for implementation via special purpose VLSI systems is explored. The filtering equations are

$$dx_t = f(x_t)dt + g(x_t)dw_t$$

$$dy_t = h(x_t)dt + dv_t,$$

where $x_t \in R^n$ and dw_t and dv_t are white noise of appropriate dimension. Other conditions are required and explained in this paper.

The filtering algorithms are to be executed in real-time and in parallel, while the resulting filter can be used in various simultaneous estimation and detection systems.

We examine existence, uniqueness and asymptotic behavior of the stochastic partial differential equation governing the filter and show that it is amenable to numerical analysis, using a technique known as the Multigrid method.

The stochastic PDE is called the Zakai equation and its solution is, when normalized, the probability density of the state x_t conditioned on the observations $\{y_s : 0 \leq s \leq t\}$. Given this density, all statistical information regarding x_t is obtainable.

When defined in n dimensions the Zakai equation is of the form:

$$dU_t = [\Sigma_{ij}^n a_{ij}(x)U_{x_ix_j} + \Sigma_i b_i(x)U_{x_i} + c(x)U]dt + U < h(x), dy_t >$$

where the coefficients are defined in the text. This equation can be shown to have a stable, implicit finite difference scheme whose solution converges weakly to the solution of the PDE. We examine some properties of this equation that are relevant to its numerical analysis, such as how it is sometimes possible to approximate its solution, originally defined on R^n, on a compact set so as to meet with finite computational requirements.

The Multigrid method involves the use of nested grids in which an original, rather complicated, linear system can be solved by approximating it on coarser grids, and exploiting the resulting reduction in problem size. The error that is incurred can be smoothed out using relaxation techniques. This method can be shown to be very efficient as a parallel solver and can be extended without a dramatic increase in computing time to relatively high dimensions. We give an indepth analysis of this algorithm, demonstrating its performance and capabilities, together with an identification of its pertinent aspects regarding our problem.

We also describe some techniques in VLSI architectural analysis that will also prove useful in our work. These include design strategies for both systolic arrays, which are synchronous array machines, and more general asynchronous systems. These concepts are utilized in our own custom-made design for real-time processing with the Zakai equation.

An important question for us is: If the Zakai equation is defined in R^n, what is the maximum dimension n we can expect to allow, for real-time signal processing? We conduct an analysis of this question using the methods of R. W. Hockney for estimating performance of general computing systems. We find that dimensions no higher than about six or seven can be reasonably treated in conventional real-time signal processing environments, which is usually on the order of about one millisec, a time bound suggested by research on signal processing for the one-dimensional Zakai equation.

DEDICATION

To My Parents

without whose support and infinite patience

this manuscript would not have been possible

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1. Nature of Signal Processing

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, bandpass 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 the Nato Advanced Study series [12],* where it can be seen that researchers have gone far beyond the classic work of Doob [8] and Wong [22]. 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 levels 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

* For convenience, all references cited will be listed at the end of the chapter they appear. A complete list of all citations can be found at the end of the dissertation.

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. Consider a robot manipulator coupled with a vision system. If the robot is expected to grasp an object moving through some complicated visual field, it must be able to track the object in "real-time" as it commands the manipulator – all the while exploiting the available feedback. Techniques more advanced than those of today's will surely be needed, as both visual and tactile data will have to be "combined" in some way as well as incorporated into the control dynamics of the arm. 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.

2. Advanced Techniques in Estimation and Detection

This dissertation will only concentrate on a selected class of problems, but our work could be considered a case study of the issues mentioned above. Our interests will center around a stochastic partial differential equation (PDE) known as the Duncan-Mortensen-Zakai (DMZ) or Zakai equation.

There are many derivations of this equation, one of the most lucid can be found

in Davis and Marcus, [7]. A model of the signal process is first formulated *a priori* by use of Itô stochastic differential equations,

$$\begin{aligned} dx_t &= f(x_t)dt + g(x_t)dw_t \\ dy_t &= h(x_t)dt + dv_t, \end{aligned} \tag{1.2.1}$$

where, $t \in [0,T], x_t, \in R^n$, $y_t \in R^p$ and x_0 has probability density $p_0(x)$, while w_t, v_t are independent Brownian motions. The above model corresponds to a number of dynamical systems: the flight of aircraft, robot manipulators, electrical circuits, etc. In the celebrated case of the Kalman filter,

$$\begin{aligned} f(x_t) &= Ax_t \\ g(x_t) &= B \\ h(x_t) &= Cx_t, \end{aligned}$$

where A, B, C are given constant matrices. We also know that the Kalman filter gives us $E[x_t|\mathcal{Y}_t]$, which in words, is the expected value of the state x_t given all the information available to us from all the observations from time $s = 0$ to $s = t$. All of this observational information is contained in the sigma field of observations, denoted by $\mathcal{Y}_t$.

The approach offered by the Zakai equation is much more general and hence more powerful. In this case we can solve the bilinear stochastic PDE

$$dU = \mathcal{L}^* U dt + h(x_t) U dy_t \tag{1.2.2}$$

where,

$$\mathcal{L}^*(\cdot) = \frac{1}{2}\sum_{i,j=1}^{n} \frac{\partial^2}{\partial x^i x^j}(a^{ij}(x)(\cdot)) - \sum_{i,j=1}^{n} \frac{\partial}{\partial x^i}(f^i(x)(\cdot))$$

where $a^{ij}(x) = \sum_{l=1}^{n} g^{il}(x) g^{jl}(x)$ and $U(x,t)$ is the "unnormalized" probability density of x_t conditioned on the observations $\mathcal{Y}_t$. To normalize we simply find:

$$p(x_t|\mathcal{Y}_t) = \frac{U(x,t)}{\int_{-\infty}^{+\infty} U(z,t)\, dz}.$$

Now clearly if we had the function $U(x,t)$ in our possession we could solve for $E[\phi(x_t)|\mathcal{Y}_t]$ where $\phi(\cdot)$ is any smooth function; for example we could compute the higher moments of x_t. However, many applications are content with approximations to the first moment, which can be obtained from the extended Kalman filter for problems of type (1.2.1). The basic idea of this approach is to simply use piecewise linear approximations for f, g, h on subintervals $[t_k, t_{k+1})$, the estimate at t_{k+1} then being the initial condition for the next subinterval. This technique has enjoyed success in a number of cases, but practical problems still remain, for the extended Kalman filter is known to diverge under certain conditions. Such obstacles can be circumvented by the Zakai equation.

The question still remains as to how to numerically solve the Zakai equation in the context of signal processing. Consider first the attempt by Bucy and Senne, (1978), to solve a related, and perhaps equally difficult problem: phase demodulation in low dimensions, [5]. This reduces to a nonlinear filtering problem which, for Bucy, involved solving the Stratonovich-Kushner equation, a nonlinear version of the Zakai equation. The resulting mean-square error performance of their design was a considerable improvement over the classical phase-lock loop. However, in order to implement their optimum filter, they employed in succession, a CDC 6600, Illiac IV, CDC-star, AP-120B, and a Cray 1, and admitted that only the most important problems could be given to such expensive tools. To the criticism that such work was impractical because of the enormous number of megaflops needed, they responded by pointing out the enormous progress in computer speed and design in the last decade, thus underscoring the declining validity of such an argument.

Nonetheless, the problem with Bucy's work from our point of view is that it simply is not practical in the context of signal processing, no matter how much progress is made in the field of supercomputers. Consider our previous argument: the need for low-cost electronic components to implement the necessary algorithms. While there may still be a place for research such as Bucy's, it would have limited

utility in a field that requires systems to be installed on robots and spacecraft. We need to consider computing systems that are even more parallel than those mentioned above. In addition, they should also be custom-made for the problem.

We also saw from Bucy's research that the demands for accuracy have a direct impact on the time required for computation as well as the type of architecture needed. This raises the question as to what the demands will be in our case, and we can answer that by placing the numerical analysis of the Zakai equation into a subsystem whose performance requirements will be dictated by the overall system. For our intention is not just to numerically implement a nonlinear filter but to do so as part of a more general algorithm, such as the problem of simultaneous estimation and detection algorithm. Indeed, the estimation and filtering problems mentioned in these pages can be thought of as a case study of more general problems that would be equally suited for real-time processing.

A typical example of a larger system will be briefly described. We must choose between two hypotheses:

$$\begin{aligned} \text{Under } H_1: \quad & dx_t = f(x_t)dt + g(x_t)dw_t \\ & dy_t = h(x_t)dt + dv_t \qquad (1.2.3) \\ \text{Under } H_0: \quad & dy_t = dv_t, \end{aligned}$$

where the same assumptions hold as in (1.2.1). At each time $t > 0$, the system either accepts one of the above hypotheses as true or continues to collect data. A cost function, which includes the cost of the data collection and false alarms, must also be minimized.

An admissible decision policy is therefore of the type $u = (\tau, \delta)$ where τ is a stopping time, used to stop the data collection process, and δ is a decision, where $\delta = 0, 1$, corresponding to the hypothesis accepted as true.

Now if u is selected as a decision strategy the miss and false alarm probabilities associated with $u = (\tau, \delta)$ are,

$$\alpha(u) = P_1(\delta = 0), \quad \beta(u) = P_0(\delta = 1), \qquad (1.2.4)$$

which, in words are the probabilities that H_1 is correct but H_0 is chosen to be true, and H_0 is true but H_1 was chosen. Set,

$$\alpha = \max_{u \in \mathcal{U}} \alpha(u) \quad \beta = \max_{u \in \mathcal{U}} \beta(u), \tag{1.2.5}$$

where $\mathcal{U}(\alpha, \beta)$ is the class of admissible decision strategies.

Define,

$$\begin{aligned} \hat{h}_t &= E[h(x_t)|\mathcal{Y}_t] \\ \Lambda_t &= \exp\Big[\int_0^t \hat{h}_s^T \cdot dy_s - \frac{1}{2}\int_0^t \|\hat{h}_s\|^2 \, ds\Big]. \end{aligned} \tag{1.2.6}$$

Then the policy u^* is of the threshold type if it obeys, $u^* = (\tau^*, \delta^*)$ where,

$$\begin{aligned} \tau^* &= \inf(t \geq 0 | \Lambda_t \notin (\bar{A}, \bar{B})) \\ \delta^* &= \begin{cases} 1, & \Lambda_{\tau^*} \geq \bar{B} \\ 0, & \Lambda_{\tau^*} \leq \bar{A} \end{cases}. \end{aligned} \tag{1.2.7}$$

In the above,

$$\bar{A} = \frac{\beta}{1-\alpha}, \quad \bar{B} = \frac{1-\beta}{\alpha},$$

and it can be shown that $\alpha + \beta < 1$. Also, u^* is optimal in the sense that,

$$E_i\Big(\int_0^{\tau^*} \|\hat{h}_s\|^2 \, ds\Big) \leq E_i\Big(\int_0^{\tau} \|\hat{h}_s\|^2 \, ds\Big), \; i = 0, 1.$$

Assuming that α and β are given, we clearly need $\hat{h}_t$, and we see that if $U(x,t)$ is the unnormalized density that solves the Zakai equation, then,

$$\hat{h}_t = \frac{\int U(x,t) h(x) \, dx}{\int U(z,t) \, dz}, \tag{1.2.8}$$

where integration is over R^n. Thus a nonlinear filtering problem must be solved as part of the overall scheme of estimation and detection.

More details on the above together with derivations can be found in LaVigna's thesis, (1986, [19]). We emphasize that this is but one example of how a numerically efficient solver of the Zakai equation could be incorporated into an estimation-detection system. The focus in this dissertation will be on the numerical analysis

of the Zakai equation with respect to real-time processing, and not on how such "Zakai Solvers" could be placed into a larger framework.

The notion of "real-time" is admittedly rather vague and it is problem dependent. In our studies, we might assume that a solution to the Zakai equation is to be provided within the time frame of the sampling rate. As we will be deal with bandlimited processes, this rate is in turn determined by the Nyquist criterion. We believe a reasonable "benchmark" for such a rate is about 1 msec, and this number will reappear throughout this dissertation when we make our performance estimates of various algorithms and architectures.

Also, since our method of filtering is recursive, we need not worry about being unable to meet this constraint as t gets large.

Thus we have the following

Research Question: Can an approximate solution to the n-dimensional Zakai equation be found in real-time from on-line measurements from the observation process, with a cost-competitive computing scheme?

Our answer will be that, at least in principle, this can be done, although in practice we would have to restrict ourselves to low dimensions, of about six or seven dimensions. We will describe in detail how such a computing scheme can be performed. Together with a discussion of the various trade-offs with system architectures, performance, speed, processor efficiency, limits on dimensional growth, etc.

Since our primary consideration must be in meeting the necessary time constraints, we will focus on the design of high-speed algorithms that can be implemented on parallel processors. By "parallel" we mean those algorithms whose work can be so distributed throughout a computing network that many functions can be performed simultaneously. Obviously, this will be much faster than sequential computation, but the concept is actually more involved than this. The key to parallel processing is the avoidance of the von Neumann bottleneck. Roughly speaking, this

occurs as a result of programming in which intermediate steps require transfers to and from the main memory to the site of computation. The solution is to presumably design algorithms which can be partitioned within computing networks so that communication between the design modules is as small as possible. Any memory that is needed could be stored locally within the module.

The parallel processing scheme we will emphasize in this paper is the Multigrid method developed by Brandt, (1977, [3],see also, [4]), Hackbusch, (1978, [11]), and Bank and Dupont, (1981, [1]). To understand the central features of Multigrid, first imagine the error between a computed solution and the true solution of a linear system as having a Fourier expansion. Then the basic concept of this algorithm is that all relaxation methods, such as successive-over-relaxation, or the Jacobi method, are very effective at reducing the high frequency component of the error between the computed solution on a given grid of points and the true solution. But these same methods do not fare as well at reducing long wavelength components. To compensate for this difficulty, transferring solutions to grids of different sizes corresponding to different wavelengths can reduce the error considerably.

We summarize this idea by a very simple version of the Multigrid algorithm, adapted from Ortega and Voigt, [21].

1). Let $G^i, i = 1, ..., K$ be a sequence of nested grids covering the domain of interest such that the grid spacing of G^{i-1} is $2h_i$ where h_i is the grid spacing of G^i. Now G^K is the finest grid on which the solution is desired and G^1 is the coarsest grid.

2). An approximate solution, u^K, to the discretized PDE $L^h U^h = F^h$ is obtained by a few (about 4 or 5) iterations of a relaxation method on G^K. This is designed to reduce most of the high frequency error on this grid.

3). For $i = K, ...2$,

a). The *residual* $L^h u^i - F^h = F^i$ is *injected* into grid G^{i-1}.

b). Using relaxation, u^i is corrected on grid G^{i-1}.

4). The solution on the grid G^i is corrected using information interpolated from the grid G^{i-1} for $i = 2, ..., K$.

More details about the Multigrid method will be supplied in this paper. We give an indication here as to why it has attracted our attention. For one, any numerical algorithm we use must involve a convergence scheme that makes use of *global* information regarding the behavior of the solution process, and which is then used to decide when to stop the numerical analysis. No matter how cleverly this is implemented, it could well take up all the time we have allotted to ourselves for computation. Clearly what is needed is the ability to precompute the number of iterations of the method necessary for convergence to an approximate solution of a given tolerance. We propose to show that the Multigrid method lends itself very well to such a requirement.

Another property of the method that must hold is that the computation time needed for, say, an estimator-detector, to perform at a given efficiency must be within the real-time constraints imposed by the problem. We believe that the computing speed provided by Multigrid algorithm is ideal for this purpose.

But this is not all. Since we have imposed a real-time processing constraint upon our problem, a result of computing theory, if not of common sense, points out that any physical system designed to compute such an algorithm must grow in size as the problem dimension increases. Thus the affirmative answer to solving in real-time the n-dimensional Zakai equation must be qualified by noting that the arbitrary growth of n precludes the realistic implementation on a small-scale electronic system. We will later provide an analysis of just how fast this system growth is relative to dimension.

It is because we wish not just to solve the Zakai equation but to do so in real-time that we must exploit the computational resources available from parallel processing. To this end, we must also consider how such processing is to be electronically implemented, so as to guarantee its economic and technical feasibility.

3. Impact of VLSI on Signal Processing

The most fundamental observation we can make about the electronics of the 20^{th} century would be its pace of miniaturization. We have still not yet arrived at the physical limits imposed by quantum mechanics on the size and design of elementary switching units, such as the transistor, but we can place millions of such devices on a single chip. Now we might naively think that, as this technical progress continues, only a reduction in size of all existing computer systems will result. But this is completely false, for Very Large Scale Integrated, or VLSI, systems offer the opportunity of designing networks of processors capable of simultaneous operation and with only local data transfers. While such a design may have been possible years ago, the size differential between what is possible today and what was available before accounts, at least in part, for the computational speed-up, as information travels over that much less a distance. But perhaps most importantly, VLSI systems have already been shown to be affordable and cost-competitive with existing designs. The processor arrays are simple in structure and usually repetitive in design, making fabrication that much easier.

Furthermore, the last few years have seen extensive growth in the design of algorithms, along with the architectures to implement them, that have fulfilled much of the promise of high speed, reliable and inexpensive computing power, all in a single chip. One of the basic reasons for this in the case of signal processing, is because the fundamental operations performed in this field are often reducible to matrix and linear algebra. For this reason, bandpass filtering, matrix-matrix and matrix-vector calculation, convolution, (as in the Fast Fourier Transform, or FFT), can be shown to be readily computable on VLSI chips and have already made their way from academic and corporate research laboratories to the commercial marketplace. The algorithms can be theoretically shown to be faster than their sequential counterparts, and the corresponding chip performance bears this out.

A good example of this technology, and one that we will devote a great deal

of attention, is the systolic array concept of Kung [14]-[19], first proposed in 1979. Kung imagined an array of processors each capable of an identical set of elementary functions. By partitioning the work of an algorithm to be performed by the array into concurrent operations, each processor would compute its assigned arithmetic functions and pass the result onto a "nearest neighbor." No access to global memory would be required and so the von Neumann bottleneck would be avoided. As information exchange is purely local within the array, there is automatically a dramatic speed-up. Attractive though it sounds, decomposing a given algorithm so as to be executed in this way is far from easy. Kung and other researchers have concentrated their efforts on certain classes of algorithms, mostly those which can be represented as recurrent sequences. They have met with great success with matrix algebra, convolution, sorting and pattern matching.

And what has the VLSI community received in return from such investigations? Their interaction with signal processing experts has delineated certain theoretical issues, such as the complexity of linear algebraic operations, stimulated the design of novel architectures, and established some of the first successes of VLSI electronics.

We therefore see that we have a firmly established empirical as well as economic base of support in the engineering community for this new technology. It is because of this that we feel comfortable in pushing these techniques into the field of nonlinear filtering. We now wish to design a VLSI system to implement the Zakai equation in higher dimensions, in real-time with on-line measurements of the observation process.

4. Plan of this Dissertation and Preview of Results

As already stated, we will be preoccupied with the question of whether real-time VLSI signal processing is possible for a class of nonlinear filtering problems. Our answer is in the affirmative, at least for dimensions no higher than about six or seven, and we will describe an algorithm and architecture to carry out the necessary operations. But more importantly, we wish that our work in nonlinear filtering be viewed as a case study of a more general strategy that involves the application of VLSI theory to general estimation and detection problems. To this end, we will provide a brief survey of the theory of VLSI algorithms in the hope of demonstrating its versatility and potential applicability to this class of problems.

There will be three main themes throughout this work. One will be the mathematical theory of nonlinear filtering, the second is the requisite numerical analysis to perform the algorithm, and the third will be the relevant theory of VLSI architectures. The three seemingly disparate approaches will be placed on a common footing by reducing the process of finding a solution of the stochastic PDE of filtering to an algorithm that can be embedded on a custom-made VLSI architecture.

The following steps is a list of Chapter outlines:

In chapter 2 we provide a brief survey of parallel computing concepts, VLSI architectures and algorithmic theory will be given along with an emphasis on systolic arrays and other related designs and ideas. This will lay the foundations of our work. These design concepts include a study of the physical basis of computation, the shuffle-exchange network, the wavefront array processor, and more, all of which will be taken up in later chapters. We will also examine a proposed design of a scalar estimator and detector, developed here by our research team at Maryland, ready for VLSI implementation. We show that this work also has a precursor, namely the Systolic Kalman Filter, which we briefly discuss. We are indeed fortunate to already have well established prototypes to pave the way for our more advanced investigations.

In chapters 3 and 4 we include A discussion of the Zakai equation from a purely mathematical viewpoint, namely, its existence, uniqueness, and asymptotic behavior. Our basic result will be a procedure to reduce the infinite domain upon which the Zakai equation is defined to a compact set, the solution of our PDE being less than a given tolerance on its complement. This is clearly the necessary prelude to any numerical analysis. Our work involves refining and extending some earlier results by Baras, *et al*, [2], which are worked out in detail for the scalar bilinear filtering problem and some higher dimensional problems. Thus a wide range of Zakai equations is available for numerical analysis, a point we were required to make as it would be pointless to discuss the use of numerical methods on a highly restricted and unrealistic set of problems.

Chapter 5 provides an implicit numerical scheme for the Zakai equation, whose solutions converge weakly to the true solution as Δt and $\Delta x \to 0$, independently of each other. This is important from our point of view since our sampling rate Δt will probably be set by the Nyquist criterion, while Δx will be determined by our demands for accuracy. Also, for explicit schemes, we would have to obey the rather restrictive stability condition $\Delta t/d(\Delta x)^2$, where d is the dimension of the problem. Implicit schemes avoid this at the expense of having to solve a linear system, (which is why we are led to the fast solvers of such systems.) We derive a number of properties of the finite dimensional matrix of the system that will be useful when our problem is incorporated into the Multigrid algorithm. These properties include identification of its block tri-diagonal structure at higher dimensions, it sparsity patterns and positive definiteness. These facts will play a role in determining the appropriate relaxation scheme for our needs.

In chapter 6 we introduce a general algorithm capable of solving our PDE as well as being an excellent candidate for VLSI implementation. This is the Multigrid (or MG) method developed largely by Brandt [4]. As mentioned before, MG techniques reduce a problem on a given grid to a simpler grid, and interpolate back

to yield highly accurate solutions. The techniques can be shown to be very fast and eminently practical for parallel processing. Our own results involve the introduction of the Fokker-Planck equation, (which is the non-stochastic portion of the Zakai equation,) as our model problem for the Multigrid algorithm. The standard model problem, from which analytical results are usually obtained, is Poisson's equation on a rectangle. Our approach introduces some technical difficulties due to the loss of symmetry in the finite difference equation, but we are still able to provide estimates on the spectral radius of the Multigrid operator, which is primarily iterative in nature, and so convergence rates are functions of this radius. Our main result is in showing the similarity in behavior of our model problem with that which is usually found in mainstream Multigrid research. This leads us to expect similar performance for our problem with that of Poisson's equation on a rectangle. We also argue that all the various parameters of the algorithm are precomputable, such as the number of iterations needed to reach a given tolerance. We do not need a convergence test as part of the routine, which would probably take up all of the allotted time given to us anyway. We will close with a survey of available Multigrid software packages.

In chapter 7 we continue with a complexity analysis of the Multigrid algorithm based on work by Chan and Schrieber, [6]. We apply their results to our own real-time processing requirements and find that their estimates are somewhat lacking in detail, as they do not properly reflect the effect of problem dimension on computing time, or architectural constraints on data communications. A discussion of the issues of efficiency and speed in the light of our own requirements is included. As the MG algorithm requires an initial approximation in order to begin its computations, we compare trade-offs between using the automatically generated initial approximations of the full Multigrid algorithm, or in using the solution at the previous time-step. We finish with a critique of Concurrent Iteration, a concept proposed by Gannon and Van Rosendale, [9].

In chapter 8, we examine the issue of relaxation schemes, which play a fundamental role in the MG algorithm, as they smooth out the highly oscillatory components of the residual before transferring it to a coarser grid. By exploiting the properties of the matrix of our implicit scheme, we argue that the ideal relaxation scheme is the successive-over-relaxation or SOR–method. We prove this with a combination of theoretical and empirical evidence which shows that, due to the similarity of our problem with that of mainstream Multigrid researches, we can expect excellent smoothing capabilities from the SOR–method. We also show that methods already exist that measure the smoothing rate of the scheme automatically, thus we can optimize the choice of any parameters available to us.

In chapter 9 we include a discussion of some empirical results of Gannon and Van Rosendale, [9], while they were at ICASE. These were problems like our own, and executed on architectures similar to ones we propose. We comment on their results in the light of our own requirements. In addition, we show that some of these architectures can be implemented in an optimal layout scheme on an electronic VLSI chip, and even discuss a design methodology we feel is suitable for this implementation, known as the Configurable Highly Parallel Design Project. It will be both efficient and reliable.

We will introduce our own design for the direct solver to be incorporated into the Multigrid Framework.

A discussion of our results on the stability aspects of the algorithm including wordlength and round-off noise considerations.

We present our results on the effect of the dimension of the Zakai equation (posed in R^d) on the real-time computing speed of the Multigrid algorithm. We believe that real-time performance can only be achieved for dimensions no higher than about six or seven.

We then ask the question whether the Multigrid algorithm could have better performance on alternative architectures, and to this end we introduce a theory of

Hockney, [13], that characterizes all computing systems with two-parameters, and from which performance estimates can be derived. We apply this theory for the first time to the Multigrid algorithm and thus estimate how well we can achieve our real-time computing constraints with other systems. Our conclusion is that the custom-made system we propose is superior to that of more general-purpose computers.

In brief, we lay down the relevant foundations of VLSI theory,
demonstrate how compact sets can be formed as a prelude to numerical analysis
introduce an implicit finite difference scheme for such analysis,
describe an algorithm that solves the linear system,
undertake a complexity analysis in the light of real-time processing needs,
present empirical evidence demonstrating the feasibility of the method, and an architecture that implements it.

5. General Remarks

Information and control theory has frequently been criticized for being too theoretical and hence too inaccessible to the mainstream engineering community. While a counter-argument might be proposed, namely that it is the nature of fundamental research to progress from the seemingly impractical to the pragmatic, it must be admitted that theoreticians have often not made an honest effort to bring their own techniques to bear on real problems. One of the reasons for this is that the language of theoreticians is rooted in some fairly difficult mathematics, and similarly, the problems of ordinary engineering work often contain subtleties that elude academics.

The answer is to presumably identify among the academic investigations the more tractable problems and find some way to "package" the theoretical techniques for solving them along with some relatively easy to use set of instructions. Consider the analogy offered by numerical analysis over the past forty years. Sophisticated

algorithms are now available for the solution of, say, differential equations or combinatorial analysis, in the computer libraries of many commercial firms. The user must know a few basics of the theory, and then a manual directs him in the details. And while expert consultation may sometimes be required, the engineer can obtain the benefit of hands-on experience with what could be a deep and otherwise inaccessible mathematical theory. Because of this accessibility, very advanced techniques, such as the Finite Element Method, have become part of the "furniture" of conventional engineering.

Our intention is to follow along the same lines. Unless the theory of nonlinear filtering is made more accessible and competitive with existing methods, (which are often no more than *ad hoc* modifications of the Kalman Filter), it may never escape the largely unread pages of the IEEE transactions and even more abstruse mathematical journals. By considering questions such as processing speed and technically feasible electronic implementation, we hope to put our theoretical results on a common footing with real-world problems. This will yield algorithms demonstrably better than conventional estimation-detection techniques, as well as electronic chips that are cost-competitive and readily integrated into present day systems. In our opinion, this is how the gulf between theory and practice within the information and control community can best be bridged.

References for Chapter 1

[1] Bank, R. and Dupont, T., "An Optimal Order Process for Solving Finite Element Equations," *Mathematics of Computation*, vol. 36, No. 153, Jan. 1981.

[2] Baras, J., Blankenship, G. and Hopkins, A., "Existence, Uniqueness, and Asymptotic Behavior of Solutions to a Class of Zakai Equations with Unbounded Coefficients", *IEEE Trans. Automatic Control*, vol. AC-28, No. 2, Feb. 1983.

[3] Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation*, vol. 31, No. 138, 1977.

[4] ——, "Guide to Multigrid Development," in Hackbusch and Trottenberg.

[5] Bucy, R. and Senne, K., "New Frontiers in Nonlinear Filtering," Lincoln Lab. Technical Note 1978-16, 1978.

[6] Chan, T. and Schreiber, R., "Parallel Networks for Multi-grid Algorithms: Architecture and Complexity," *SIAM J. Sci. Stat. Comput.*, vol. 6, No. 3, July, 1985.

[7] Davis, M. H. A. and Marcus, Steven, "An Introduction to Nonlinear Filtering," in *Stochastic Systems: The Mathematics of Filtering and Identification*, (NATO Advanced Study Institute Series). Dordrecht, The Netherlands: Reidel, 1981, pp. 53-75.

[8] Doob, J., *Stochastic Processes*, Wiley, 1953.

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

[10] Hackbusch, W. and Trottenberg, U., eds., *Multigrid Methods*, Lecture Notes in Mathematics, Springer-Verlag, No. 960, 1982.

[11] Hackbusch, W., "Multigrid Convergence Theory," in Hackbusch and Trottenberg.

[12] Hazelwinkle, M. and Willems, J. C., eds. "Stochastic Systems: The Mathematics of Filtering and Identification," (NATO Advanced Study Series), Dordrecht, The Netherlands: Reidel, 1981.

[13] Hockney, R. W. "Performance of Parallel Computers," in Paddon, D., *Supercomputers and Parallel Computation,* Clarendon Pr., Oxford, 1984.

[14] Kung, H. T. and Lam, M., "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *Journal of Parallel and Distributed Computing,* vol. 1, 1984.

[15] Kung, H. T., "Two-Level Pipelined Systolic Arrays for Matrix Multiplication, Polynomial Evaluation and Discrete Fourier Transform," Workshop on Dynamic Behavior of Automata, Luminy, France, 1983.

[16] ——,"Systolic Arrays," Dept. of Computer Sci., Carnegie-Mellon Univ. Pittsburgh, Penn. 1984.

[17] ——, "Why Systolic Architectures," *Computer,* Jan. 1982.

[18] ——, "Systolic Algorithms," in *Large Scale Scientific Computation,* Academic Pr. 1984.

[19] ——, "Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI," from the Proc. of the Soc. of Photo-Optical Instr. Engr., 1980.

[20] LaVigna, A., *Real Time Sequential Detection for Diffusion Signals," Master's Thesis, Dept. of Electrical Engineering, Univ. of Maryland, 1986.*

[21] Ortega, J. and Voigt, R. "Solution of Partial Differential Equations on Vector and Parallel Computers," SIAM Review, vol. 27, No. 2, June, 1985.

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

2. Selected Topics in VLSI Systems

1. Introduction

This chapter is intended as a quick review to the concepts that will be reiterated throughout this dissertation. The reader is encouraged to scan through this chapter to orient himself for the kinds of arguments we will be using.

To believe that modern electronic circuit fabrication offers nothing more than a reduction in scale of previously existing integrated systems is to miss the point of the revolutionary advance to computation offered by VLSI technology. For two seemingly divergent mainstreams of thought are finding common ground; one involves a theoretical approach to the structure of algorithms, the other represents dramatic improvements in the manufacture of highly dense and reliable circuitry on a chip. There are four concepts that bridge these two fields of inquiry.

One is the notion of *concurrency*, which simply means that a program is so designed as to allow for a large number of calculations occurring at the same time. A synonym might be *parallelism*, but some authors would disagree, Seitz, [15], for one, as the second term has connotations of simultaneous operations being performed in lockstep with other computing streams. This conflicts with the suggestion of subsystem independence implicit in our notion of concurrency. In either case, clearly a reduction in overall computation time over the conventional sequential machine is possible. Unfortunately, as attractive as the idea may seem, our minds consciously work out problems sequentially, so we will have to rethink our approach to computing.

Pipelining is the second concept. One can visualize this as being analogous to an assembly line in a factory. Data enters a system and is successively transformed in stages until the desired result is obtained. While this may sound like a sequential program, each stage can perform its own work concurrently and the data can be fed into the system in parallel.

The third notion is *hierarchy*. The control of operations within the system must be distributed. This implies the use of modular designs with subsystems having as much control over their own operations as possible. Otherwise, requests and commands will flow up and down the hierarchical structure, with global control at the apex. This will be costly in computation time.

The fourth concept is related to the third. Designs based on the *Principle of Locality* involve keeping control, memory, and other requisite data as close to the computation site as possible. The general rule is: computation is cheap, but communication is expensive, and the expense is in the time needed to transmit the data. It was only until a great many reliable processing components, the kind and number needed for concurrency, could be placed very close to each other on a chip, dramatically reducing the communication costs, that parallel computation and VLSI technology could be brought together.

Even among VLSI designs there has been a proliferation of concepts such as arrays of all types, cube connected cycles, perfect shuffles, trees, and sychronous vs. asynchronous systems. It is not our intention to give a survey of all that is available. We will instead concentrate on the systolic array paradigm by Kung *et al*, [6]-[11]. While this design does not impart the full range of VLSI versatility, it certainly serves as an excellent case study of the basic ideas of the field. And of course, this is the design, along with its variations, which we have chosen to explore in our own research.

We begin with a brief overview of the systolic array. Then a remark on the physical basis of computation time, namely, the relevance of the $O(l^2)$ signal speed model on our future design considerations. An introduction to shuffle-exchange networks is given. These are precisely the class of networks that would be the most adversely affected by the $O(l^2)$ signal speed model, due to their high wire density. We cite arguments that suggest that these networks are still practical for high-speed computations, as we intend to use them for just this purpose.

The problem of synchronization of large arrays is introduced, and as we intend to use large arrays for our PDE computations, together with nearly nanosecond clock timing, we conclude that asynchronous control will be needed.

On that last note we introduce the Wavefront Array Processor (or WAP) which we feel is suitable for use in the relaxation processes that are part of the Multigrid algorithm.

Finally, to demonstrate that there exist precursors to our research, we cite the work performed here at Maryland on one-dimensional nonlinear filters and the systolic Kalman Filter.

2. Systolic Arrays

The design proposed by Kung appears to follow quite naturally from some basic principles. First, we must divide the sequential processor up into a network of smaller, simpler processors, each capable of carrying out their own independent tasks such as short addition/multiplication operations. This network could be be composed of several to hundreds of such processors, which are then arranged in some regular geometrical format. In this way we can accomplish the distribution of labor required by parallel processing.

Once a task is performed by a single processor, the resulting product, which is but an intermediate step in a much larger computation, must be transferred on to another processor, and it is desirable that this processor be as close by as possible; a "nearest neighbor" would be the best choice. This solves the problem of time lost to distance in the communication of data, and is an example of the Principle of Locality at work. It also plays a role in determining the geometrical configuration of the array. This will not be the last time that we will witness a subtle connection between interprocessor communication and network topology.

Our final design feature is that the flow of data be regular, and hence the movement of processor output to it nearest neighbor be spurred by a global clock

pulse. Thus at each regular interval, each processor transmits and receives data. This last characteristic may not necessarily be a requirement for parallel processing, but it is the most conspicuous feature of systolic arrays, next to their geometry. In fact, the word "systolic" comes from the analogous flow of blood in the cardiovascular system, for this flow is not uniform, but moves instead in rhythm with what physiologists call the systolic beat of the heart.

Despite the attractiveness of this array at first sight, a number of controversies have surrounded it, some only now being settled. Some of the first disagreements centered around circuit fabrication, for it was not clear that the processor networks could be reliably built to submicron dimensions. Actually this argument could be lodged against all VLSI technology, for there are a multitude of problems faced by micro-miniaturization. Most of the current work involves MOS technology, which has the highest circuit density capability. But parasitic capacitance dominates the circuit speeds and transistor driving performance. As an example, one minimum size transistor can drive the gate of an adjacent identical transistor in about 0.1 nanoseconds. But add a few hundred microns of wiring, and the delay is increased to several nanoseconds. The nonzero resistance of the wires, with their parasitic capacitance, impose a delay that is becoming increasingly troublesome. Of course, there are many other problems as well, such as the macro to micro interface of interchip wiring, package pins, etc.

From the above remark about signal speeds one can see why the Principle of Locality has become so important, and it is clearly this principle, among others, that systolic arrays exploit.

The other objection has centered around the design of parallel algorithms. These must be custom-made for the array and until recently no general theory was available to design them. Originally, *ad hoc* procedures were used, and so it was not clear how powerful this approach really was. However, to this day, it appears that algorithms that are reducible to recurrence relations are best, matrix analysis

being the best example. As much of signal processing is itself reducible to linear algebraic operations displayed as recurrence relations, systolic arrays have perhaps had their most celebrated success in this field.

As stated before, systolic arrays are a combination of individual processors in some geometrical arrangement, such as linear, orthogonal and hexogonal designs. In the original Kung model, the unit element of the array is the *inner product step processor*. It is designed in the following way. Let A, B, C be input data fed into lines I_A, I_B and I_C respectively. The following operations which are fed into the output lines are done in a single time step:

$$\begin{aligned} O_C &= I_C + I_A \times I_B \\ O_B &= I_B \\ O_C &= I_A \end{aligned} \tag{2.2.1}$$

Note that this simple processor has no memory and performs only a single arithmetic operation and an in-and-out data transfer. Semi-systolic arrays would be, among other things, variations on the above theme. However, the reader should not be misled by the apparent simplicity of this fundamental "workhorse" of the systolic concept, for many useful numerical results can be obtained from them.

More examples of systolic array applications and designs can be found in Mead and Conway, [13], as well as the papers of Kung, [6]-[11]. Matrix operations are discussed there as well. It is shown that a linear system with no need of pivoting can be solved in $O(n)$ time units.

3. Physical Basis of Computation Time

Throughout this investigation, we will be concerned with the speed with which computation can be performed, for we want to conduct our computations in a real-time environment. Certain issues involving the physics of computation are worth mentioning.

Mead and Conway, [13], give some excellent examples of how elementary physics predetermined basic system parameters such as the duration of time-steps. For example, taking fanout and stray capacitance into account, they calculate that the time needed for one clock cycle, during which register to register data transfers are made, is about 100τ, where $\tau = 0.3$ nanoseconds.

Computation is clearly a function of signal speed, and one would hope that as we turned to VLSI designs, that signal speed would be the least of our worries. For if signals were to travel at the speed of light, surely the time required to travel down a wire would simply be proportional to the length of that wire. However, this reasoning is simply not correct, as intuitive as it may appear.

A wire has a resistance and a capacitance, and (according to the fabrication technique), both grow linearly with wire length. Thus the time constant of a transistor load grows proportionately to l^2, where l is the length of the wire. However, if this is the case, networks of substantially long wire lengths will be ruled out as impractical. This would include the shuffle exchange, the cube-connected cycle or the tree connected machine.

Unfortunately, it is networks analogous to shuffle-exchange design that we will investigate in later chapters. Thus some attention to the problem of signal speed is in order.

Bilardi, *et al* [1] conducted an investigation of the problem of signal speed and found that signals governed by $O(l)$ speeds were indeed possible even with anticipated changes in circuit design, such as dramatic reductions in size.

Acknowledging the parasitic capacitance of the wire, which takes precedence

over its resistivity, Seitz, [15] derives a worst case value of $O(\log l)$ as the signal speed which differs from the assertion of Bilardi.

Adding the resistivity of the wire yields a signal speed $= O(l^2)$, and this is already apparent in the silicon and polycrystalline silicon wires of today - which disagrees with Bilardi's optimistic account that such phenomenon is avoided altogether. But Seitz also points out that adding active repeater amplifiers periodically along a long communication path reduces this delay to $O(l)$.

While it may be surprising that such a controversy still exists in the electronics community, we can still conclude that not only the current but all projected MOSFET VLSI technologies are well within the $O(l)$ signal speed. This says that as we scale the electronics down in size we need not pay for it with poor signal speed. While homogeneous systolic arrays will have the most attractive signal speed, (due to close processor proximity), even networks like the perfect shuffle will still have an attractive signal speed.

Finally, we might remark that we will exploit the results in this section in a later chapter.

4. Shuffle-Exchange Processor Arrays

The effectiveness of parallel processing is heavily dependent on the topology of interprocessor connections, as it impinges directly on the efficiency of data-transfer and perhaps indirectly on the duration of the clock cycle–due to the impact of wire length on signal transmission. Area layout of the circuit is also clearly affected. All of these considerations require us to carefully choose the architecture needed for the algorithm. The *shuffle-exchange networks,* of which there are several variations, are often among those choices, and their many advantages and disadvantages must be carefully weighed. One immediate remark one can make is that these networks are different in structure from those already examined. The previous section made a distinction between pure and semi-systolic arrays. An example of the latter is the Perfect Shuffle processor array, and as it will play a role in later developments in our research, we discuss it here.

A shuffle of components of a vector is equivalent to viewing that vector as a card deck, and shuffling those components so that after one shuffle the components from the two halves of the vector alternate. While such a capability may not seem especially useful, the perfect shuffle has been used successfully in a number of applications, such as the FFT, polynomial evaluation, sorting, and matrix transposition, as described by Stone, [16], one of the network's early expositors.

Consider fig. (2.4.1). On the left in the figure is a vector of operands with indices running from 0 to $N-1$, where $N = 2^m$ for some integer m. The components of the vector on the left are transferred to cells as they are numbered on the right. Examination of the indicial pattern yields the following relationship: indices on the left are mapped onto indices on the right according to the permutation $P(\cdot)$ such that

$$\begin{aligned} P(i) &= 2i \qquad 0 \le i \le N/2 - 1 \\ &= 2i - 1 - N \qquad N/2 \le i \le N - 1. \end{aligned} \tag{2.4.1}$$

Now return to the card deck analogy and divide the vector on the left in two pieces. Now shuffle the two pieces by transferring components on the left to components

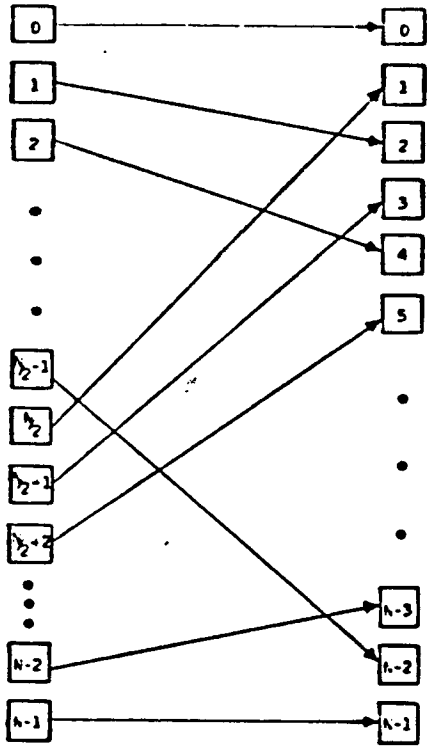


Figure 2.4.1

The Perfect Shuffle Connection

on the right. The result is clearly similar to the shuffling of cards.

An equivalent formulation is related to the binary representation of the indicies of a vector. More precisely, let the i^{th} element be shuffled to position i' whee i' is obtained by cyclically rotating the bits in the binary representation of i one bit position to the left. For example, let $i_k = \{0, 1\}$ and let,

$$i = i_m 2^m + i_{m-1} 2^{m-1} + ... + i_2 2 + i_1.$$

Then i' will be,

$$i' = i_{m-1} 2^m + i_{m-2} 2^{m-1} + ... + i_1 2 + i_m.$$

This is equivalent to pairing adjacent nodes in a Boolean m-cube.

The elementary idea expressed here has several variations. Let the above model be called the Ω network. Then the *inverse Omega network*, denoted Ω^{-1}, is simply

the reverse of the original, i.e., data flows from right to left in fig. (2.4.2). Then we would have $\log_2(N)$ unshuffles.

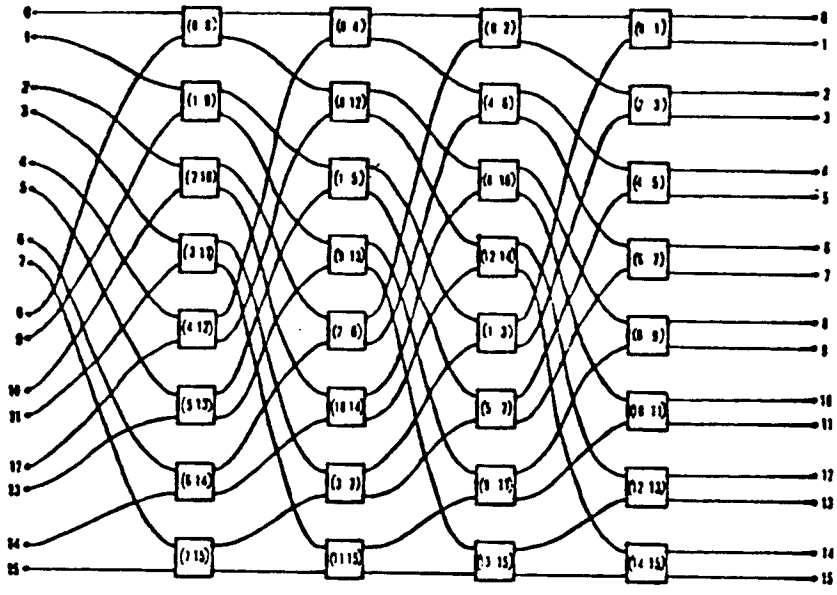


Figure 2.4.2

The Ω-Network

There are many other networks that are based on similar principles of the $\Omega-$network; indeed, many are isomorphic to it, both in the functional and topologic sense, (see Parker, [14], for details.)

Any discussion of permutations invokes the theory of the Symmetric Groups on N elements, denoted S_N. We could ask the following questions: What is the smallest value of k such that

$$S_N \subseteq (\Omega_N)^k, \tag{2.4.2}$$

i.e., how many passes through the Omega network are necessary to insure that any given permutation in S_N can be generated? We would want to know this in order to

determine this so as to know how to realize a given processor-memory interaction, or the transfer of a specific pattern of information from one network of processors to another. More general forms of data transfers (i.e., those which may not be bijection) can also be determined by extension of results on S_N.

Unfortunately, $k \neq 1$ in (2.4.2) since the cardinality of S_N is

$$N! \sim (\sqrt{2\pi})2^{(N+1/2)\log_2 N - N\log_2 e}, \tag{2.4.3}$$

and the cardinality of $\Omega_N = 2^{N/2 \log_2 N}$. By a sequence of technical lemmas, Parker was able to show that,

$$S_N \subseteq (\Omega_N)^{\min(3,\log_2 N)}. \tag{2.4.4}$$

Such interprocessor networks become troublesome, however, in their area layout requirements. These are high wire area designs–indeed, the lion's share of the chip area is taken by the interconnection rather than by the processors, and this area grows rather fast with the number of processors. This is perhaps why Ullman, (in whose book area layout analysis can be found, [18]), argues that such systems are best suited for "supercomputers," at least for when the number of processors becomes large. He also discusses the result that the shuffle-exchange graph has an optimal area layout capacity of $A = \Omega(n^2/\log^2 n)$.

Thus we see that unless N is kept fairly small, these networks quickly become impractical for small VLSI chips. This has relevance for our own design considerations, since we will investigate the use of the Ω–network as a means of communication between interprocessor grids.

5. Synchronization of Large VLSI Processor Arrays

Returning now to our original systolic array model, we recall that one of its central tenets is the global systolic clock pulse by which all processor operations are synchronized. This is performed by signals emitted from one or more sources that are designed in such a way as to propagate throughout the network, initiating processor operations simultaneously throughout the array. Unfortunately, as the system becomes large, clocking schemes can be difficult to implement due to the inevitable problem of clock skews and delays, due to non-uniform wire construction, chip defects, etc, which tend to become large for VLSI systems as feature sizes shrink. The reason is not hard to see. As we move from micro to nano-second timing, networks become more sensitive to the slightest move out of synchrony. But as our computational requirements are so intensive, synchronization of processor arrays is an issue we cannot ignore.

However, pessimism expressed over synchronization has apparently not always been warranted (see and S. Y. Kung, [5]), as the circuit designer can often compensate for some difficulties by adjusting RC constants of clock signal wires or exploiting better fabrication materials. The electrical problem of passing a clean signal within a chip can be difficult, but sometimes long wires can be replaced by strings of buffers, which can restore signal levels and prevent backward noise propagation.

Clock skew is due to three factors. The RC of the global clock distribution line is one important factor. A second is the sometimes unavoidably unequal clock paths to processors in the array. These two factors are functions of the layout design. Finally, a third factor is the variance of values of the gate threshold voltage of the processors, which receives the global clock signal and transmits it throughout its interior. It therefore serves as a buffer between the array's global clocking scheme and the processors own local synchronization. This last factor, as well as the first two, are dependent on the fabrication process.

As pointed out in the previous section on the "Physical Basis of Computation Time," the behavior of signals along the "wires" in the chip are similar to what one encounters in transmission lines, except that inductance is negligible, while capacitance plays a dominant role. Over long signal paths, the delay is sure to cause synchronization problems. The line resistance will also plays a role.

The equation that governs clocking signals on the line is

$$V_t = V_{DD}[1 - \exp(-t/RC)]. \tag{2.5.1}$$

Now clock skew is due to variations in V_t, R and C. As an example, V_t may be $\pm 20\%$ of its nominal value. Thus clock skew obeys

$$\begin{aligned} \Delta t &= \text{clock skew} \\ &= RC(max)\log(V_t(max)] - RC(min)\log[V_t(min)] \end{aligned}, \tag{2.5.2}$$

where Δt is the pulse width.

One of the ways of transmitting the clock signal throughout the array is by distributing the clock through an *H-tree*. This design is built as a recurrence process. Begin with the letter "H." In the center of the wire connecting the two vertical is the root, and the four "leaves" are the two top and two bottom points of those verticals. From those four points, make four smaller H patterns, stretching the original design to accommodate. This can be continued indefinitely. At each leaf will be a processor, all of which will be equidistant from the root. For details on H-trees, see Ullman, [18].

Fisher, [3], investigating clock skew phenomenon proved

Theorem 2.5.1: (Fisher, [3]) For the linear array, a clocking scheme exists with clock skew bounded from above and below and a fixed clock period independent of the size of the array.

Unfortunately, the theorem cannot be extended to the two-dimensional processor array even with H-tree wiring. For if n^2 is the number of processors in the

$n \times n$ array, then the clock skew σ cannot be bounded from above by a constant independent of n. Fisher uses a graph argument to show that

$$\sigma = \Omega(n). \tag{2.5.3}$$

An even more pessimistic analysis was performed by H. T. Kung and Gal-Ezer who found that clock skews grow as $O(N^3)$. They pointed out that the H-tree has arms of equal length, and so the values of R and C representing the levels of the H-tree structure are equal. But as the lengths of the H-tree arms double from one H-level to the next, the values of both R and C at each H-level will be twice that of their successor H-levels. Using this reasoning, the resistance of an H-tree signal path, used in the global clock scheme, will be of order $O(N^3)$, while its capacitance will be of order $O(1)$. Thus the time constant of the distribution network is, $O(N^3)$.

Conducting a SPICE analysis on the equivalent circuit of the distributed clock, where each segment of the H-tree is a lumped RC branch, and using various appropriate values of resistivity and capacitance per unit length, the researchers found clock skew on the order of 10^2 nanoseconds for $N > 60$ and even $N > 40$ for one model. We are forced to conclude that as the mesh-array becomes larger, no synchronization scheme can save us from worsening clock skew. We are therefore led to consider strategies similar to that proposed by Seitz, [15], which combines local synchronization and a global asynchronous design. The basic idea is to let processors synchronize their communication locally with some variety of "handshaking" protocol. Now any synchronized parallel system where processors operate in lockstep can be converted into a corresponding asynchronous system–just by letting each processor start computing as soon as its inputs become available from other processors. But this form of system timing can be costly in terms of extra hardware and delay in each processor. However, this timing scheme can be implemented without regard to the size of the array.

We conclude that as we move to large mesh-arrays, (and we will later in our study,) asynchronous systems may have to be considered.

6. The Wavefront Array Processor

As indicated in the last section, unless technical advances intervene, synchronous control of large systolic arrays may prove to be so difficult as to reduce the cost-effectiveness of this architecture. A naive solution to the problem is to turn off the global clock and allow each processor to send output on to the processors dependent on it simply when it is ready, and not to wait for the clock signal. This is the basic idea of the *data-flow architecture.* Being a language-based architecture, algorithms can be mapped directly into the multiprocessor array in a way that achieves high performance. It is designed as a general-purpose system, and therefore the data-flow machine often involves a great amount of data and resource management, and needs a powerful supervisory system. Now one of the recurrent themes we have seen in the theory of VLSI algorithms, is that special-purpose architectures, whenever possible, are usually easier to program and have higher performance than those architectures designed for a wider class of problems. Systolic arrays, despite their possible synchronization problems, are good examples of special-purpose systems that exploit the recurrence structure of the algorithms they implement, matrix algebra being an example. As the the problems we wish to examine in our research can be thought of as being in this special class of algorithms, it behooves us to consider a variation on the systolic array that bypasses the synchronization issue. An architecture that offers a compromise the systolic array and the general-purpose data-flow machine is the *wavefront array processor* or WAP promoted by S. Y. Kung, * *et al*, (1982), [5].

We are still dealing with a processor laid out in a repetitive geometrical pattern, so as to reduce fabrication costs. And we still have localized communication between processors, so as to avoid the cost of global data exchanges; thus the Principle of Locality still holds.

What is different is that instead of systolic clock, we have asynchronous dis-

* Not to be confused with H. T. Kung of the systolic array.

tributed control with localized data flow.

If the central paradigm of the systolic array is the rhythmically pulsed flow of data, then the distinguishing feature of the WAP is the notion of a *computational wavefront.* This portrays the movement of data as the back and forth roll of a wave throughout the array. This results directly from an exploitation of the recurrency of the algorithm and the built-in protocols controlling data dependency and transfer. A computational sequence starts with one element and propagates through the arrays, in a manner resembling a physical wavefront. All algorithms that admit locality and recursivity will exhibit this phenomenon. Thus it can be safely said that any algorithm suitable for a systolic array can be implemented on a WAP, but, as we will see, without synchronization problems.

If we were to imagine ourselves looking down on the array and witnessing the motion of the wavefronts, we would see that Huygen's principle was in force. Thus the wavefront never intersects. Problems with timing uncertainties, such as local clocking, random delay in communications, and fluctuations of computing times, are altogether avoided by what amounts to asynchronous waiting times between the processors. According the S. Y. Kung, the WAP is the optimal trade-off between the globally synchronized and dedicated systolic array, (that works on a similar set of algorithms), and the general-purpose data-flow multiprocessors.

The concept of the computational wavefront can be made more clear in an example. Consider the multiplication of two $N \times N$ matrices: $C = A \times B$. Let A_i be the column vectors of A and B_j be the row vectors of B. Then using "$\cdot$" to denote matrix multiplication, we have,

$$C = A_1 \cdot B_1 + A_2 \cdot B_2 + ... + A_N \cdot B_N. \tag{2.6.1}$$

Using N recurrences we have,

$$C^{(k)} = C^{(k-1)} + A_k \cdot B_k, \qquad k = 1, 2, ..., N. \tag{2.6.2}$$

We can map this problem in a natural way into a square $N \times N$ array, where we label the processors by (i,j) and whose final output will correspond to C_{ij}. The registers are all initialized to zero, so $C_{ij}^{(0)} = 0$. Entries of A and B are stored in memory outside the array, but ready for pipelining, with A on the left of the array and B on the top. Starting with processor $(1,1)$ we obtain

$$C_{11}^{(1)} = C_{11}^{(0)} + a_{11} \times b_{11}. \tag{2.6.3}$$

The computation then "propagates" to (1,2) and (2,1) to yield,

$$\begin{aligned} C_{12}^{(1)} &= C_{12}^{(0)} + a_{11} \times b_{12}, \\ C_{21}^{(1)} &= C_{21}^{(0)} + a_{21} \times b_{11}. \end{aligned} \tag{2.6.4}$$

Moving on to processors (3,1), (2,2) and (1,3) we also obtain similar results. The computational wavefront is analogous to the optical wave phenomenon that obeys Huygen's principle, in that each processor acts as a secondary source and is responsible for propagation of the wavefront. This localized data-flow is implied by this notion of wave propagation.

The first recursion is over when the first wavefront hits processors (N,N). But we have already started the second recursion by propagating a second wave beginning at (1,1) immediately after the first one. Thus (1,1) will execute,

$$C_{11}^{(2)} = C_{11}^{(1)} + a_{12} \times b_{21},$$

and so on. The (i,j) processor will execute the k^{the} recursion as

$$C_{ij}^{(k)} = C_{ij}^{(k-1)} + a_{ik} \times b_{kj}.$$

Note that the wavefronts never intersect, which again is analogous to Huygen's principle. Computational wavefront phenomenon such as indicated here stems from locality, regularity, recursivity and concurrency.

S. Y. Kung has constructed a language specially adapted to the needs of matrix algebra, called the *matrix data-flow language* (MDFL). This language allows relatively easy programmability, simulation and verification.

A detailed description of the language with examples can be found in [], but we state here that the structure of the language rests on:

1). *Space Invariance:* The tasks performed by a wavefront in a particular kind of processor must be identical in all wavefronts (as in the case of $N \times N$ matrix multiplication when the first wave is at (1,1), the second at (2,1) and (1,2), continuing to $(N, N), 2N - 1$ fronts in all.)

2). *Time Invariance:* Recursions are identical.

MDFL makes it possible for the array programmer to program all processors in a wavefront at the same time, thus the wavefront concept is a simple organizing principle. When written, the programs convey all of the intuitive notions inherent in the problem statement, such as locality, regularity, recursivity and concurrency. Centralized control and globally shared memory is avoided by asynchrony and maximal parallelism. S. Y. Kung offers MDFL as a prototype of future parallel processing language.

The architecture of the WAP is similar to all data-flow machines except that all processors must wait for a primary wavefront before performing their computations and only then act as a secondary source of new wavefronts by transmitting their outputs. To avoid overrunning of data wavefronts (in conformity with Huygen's principle), the processor hardware ensures that a processor cannot send new data to the buffer unless the old data has been used by the neighbor. All of the above is orchestrated by buffers and flags such as DATA READY/DATA USED signals. Thus when data is ready the transmitting processor informs the recipient of this fact. Once the data is used, the sender is similarly informed. This is referred to as a simple handshaking protocol. (See S. Y. Kung *et al*, [5], for more details).

Hardware characteristics of the WAP are similar to the systolic array: it is modular in structure, and laid out in a regular geometrical pattern for easier fabrication.

Also, if we are dealing with matrices, we need not have an array size to match

our matrix dimension. If the WAP is an $N \times N$ array and A is an $N' \times N'$ matrix, with $N' = mN$ (m an integer), we can partition A into submatrices of size $m \times m$ and allow each processor work sequentially with its submatrix. Therefore, we have system extendibility.

S. Y. Kung gives an number of applications of the WAP in his paper, [5]. Letting t_a, t_m, t_d be executions times of addition, multiplication and division respectively for a single processor, and letting β be the interprocessor data transfer time, (or their expected values, if these operations are inherently random), we find that $N \times N$ matrix multiplication takes $(3N-2)(t_a + t_m + \beta)$, an improvement over the $O(N^2)$ time for systolic arrays.

Of special interest to us is the use of a *relaxation scheme* to solve a partial differential equation in time $3m(3t_a + t_d)$, where m is the number of iterations. In this case, the scheme involves taking central differences. Consider Laplace's equation in two dimensions,

$$\Delta u = 0, \tag{2.6.5}$$

with $u(i,j)$ corresponding to the value stored in processor (i,j). We set,

$$u(x,y) = \frac{1}{4}\Big[u(x-1,y) + u(x+1,y) + u(x,y-1) + u(x,y+1)\Big]. \tag{2.6.6}$$

The boundary conditions are preloaded into the processor on the array's perimeter. Each recursion consists of only one wavefront, and we have m recursions the number of which having been predetermined by numerical analysis, (so we are not using some comparative scheme to decide when to halt the computations). Since each processor requires four data transfers (two of which are done in parallel), three additions and one division we have a processing time of

$$3m(3t_a + t_d). \tag{2.6.7}$$

Note that the computing time is completely independent of n. Also, no convergence criterion is included. This design will reappear in our discussion of Multigrid algorithms, which make extensive use of relaxation methods.

Comparisons with other architectures are possible. The creators of the WAP argue that the systolic array lacks the programming flexibility offered by their MFDL. More importantly, global synchronization is a major obstacle, especially for large arrays, with S. Y. Kung pointing out that clock skew can grow as fast as $O(N^3)$ for an $N \times N$ array (see [3], or previous section on synchronization). Admittedly, hardware complexity, extendibility, testability, etc, must also be considered, and current technology has yet to reach a verdict on these issues.

As mentioned before, the WAP exhibits most of the advantages of data-flow machines while still remaining a special-purpose devices, with all of the desirable features that this implies, such as programming ease, functional modularity and reduction in design costs.

For another comparison, and one often cited as a typical example of an array computer, the Illiac IV shares few of the features usually deemed compatible for VLSI implementation, although a wider range of problems can be programmed on it than either the systolic array or the WAP. Like NASA's Massively Parallel Processor and ICL's Distributed Array Processor, this SIMD computer has both a global memory and control unit with oversees all processor operations, which also occur synchronously. While currently suitable for LSI, (clock skew is less of a problem here), global communication costs would still make this architecture inappropriate for high-speed processing even if fabricated in a high-density circuit.

Contrasting these features with the WAP we find that local communication, local instruction storage and data-flow based control makes it much more attractive for VLSI.

We conclude, the Wavefront array processor, or variations of its design, should not leave our attention as we continue our researches into array processing.

7. A VLSI Architecture for the Scalar Nonlinear Filter

A procedure for calculating the solution of the Zakai equation when the state is a scalar has been presented elsewhere, (see LaVigna April '86, [12]). The basic idea is to use a certain implicit finite difference scheme represented as

$$(I + \Delta t A)V^{k+1} = D_k V^k, \tag{2.7.1}$$

which is equivalent to the one we will employ for higher dimensions. The derivations and properties of this scheme will be presented in chap. 5, but suffice it to say for the present that the matrix $(I + \Delta t A)$ is tridiagonal for the scalar case and strictly diagonally dominant, which implies that no pivoting is required as a prelude to finding the solution of (2.7.1).

The matrix D_k is diagonal, so the multiplication of $D_k V^k$ is a purely parallel operation, as V^k is a vector in R^n corresponding to the n grid points on a compact set of R.

We might remark that an explicit finite difference scheme is also available for the Zakai equation, but we deemed it as less satisfying due to its constraints on Δt and Δx, (see chap. 5). However, this would only require a matrix-vector multiplication and we already know that this can be efficiently done in $O(n)$ steps for band matrices (here the bandwidth is three.) This approach is especially unattractive for higher dimensions though, since the constraints on Δx and Δt are even more confining.

If we want to solve the linear system (2.7.1) we can use LU decomposition, i.e.,

$$LU = (I + \Delta t A),$$

where L is a unit lower triangular matrix and U is a upper triangular matrix. In fact, both L and U are bidiagonal.

Thus we solve

$$Lx = D_k V^k, \tag{2.7.2}$$

and then solve for

$$UV^{k+1} = x. \tag{2.7.3}$$

Both systems (2.7.2) and (2.7.3) can be solved by systolic arrays. For example, in the case of $Lx = b$, where x is a vector, we have the following recursion,

$$\begin{aligned} y_i^{(1)} &= 0, \\ y_i^{(k+1)} &= y_i^{(k)} + L_{ik}x_k, \\ x_i &= (b_i - y_i^{(i)})/l_{ii} \end{aligned} \tag{2.7.4}$$

A systolic array that can solve for x, given the data L and b, needs a special processor to compute $(b-y)/a$, but this slight modification leaves the system essentially a linear processor array. Note that no synchronization problems arise because of this.

Solving the upper triangular system $Ux = b$ is basically the same.

Note that since $(I + \Delta tA)$ is a constant matrix for each t, the factorization is precomputable and is done off-line.

A block diagram of the sequential detector alluded to in the introduction of this dissertation that incorporates this Zakai solver is shown in fig. (2.7.1).

A future technical report of the Systems Research Center at the University of Maryland (to appear probably in the summer of 1986, and to be authored by D. Simmons, a student research assistant of the SRC,) will describe the actually design and layout of the sequential detector.

A major question relating to the electronic implementation of the detector was whether fixed or floating point arithmetic should be used. The trade-off here is dynamic range vs. computing speed. As we would expect to have numerical values ranging from 10^{-6} to 10^{9}, floating point computing power would appear to be in order, so speed was sacrificed in exchange for using the IEEE standard floating point. Optimal design considerations are awaiting the actual construction of this processor.

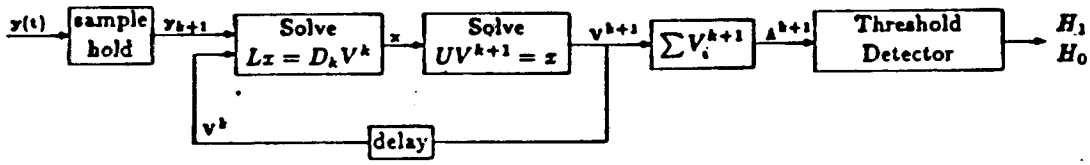


Figure 2.7.1

The Simultaneous Estimator-Detector with Zakai Solver

At this point we might mention the work of Travassos at Systolic Systems, Inc., ('83, [17]) involving the design of a real-time systolic Kalman filter. The predictor-corrector equations are decoupled and reduced to matrix-vector multiplications, using the design described in an earlier section of this chapter. Real-time implementation was shown to be possible, with stability, wordlength considerations and round-off noise posing no problem.

Another design of the Kalman filter on a chip was reported in *Electronic Design*, 1984, [2].

8. Conclusion

After giving a brief overview of parallel computing concepts, we examined systolic arrays: their basic unit, the inner-product step processor, array configuration, and some examples that can be implemented on them, all involving matrix analysis. Regularity of design, recursivity of the algorithm, concurrency of operations and synchronization of the processing were all identified as trademarks of this architecture.

Moving more deeply beneath the architecture we found that various models of signal speeds in VLSI are as yet still in competition–with the prevention of $O(l^2)$ data transfer times, with l being the length of the wire, all but completely precluded. This will become an important issue with some of the designs we will be examining later in our research.

A discussion of the shuffle-exchange networks as examples of semi-systolic arrays followed. These are often used as part of a data-transfer mechanism and are very flexible as such. We will refer to them later when we wish to efficiently convey data from one processor grid to another, as will be needed in the Multigrid architecture. One drawback, however, is that they consume large quantities of the chip's area. Also, if the $O(l^2)$ hypothesis is assumed to hold, then these networks will quickly become impractical for the kinds of high-speed computing we have in mind.

Returning to systolic arrays, we pointed out that global synchronization of a large array is problematic at best, with clock skew becoming difficult to control as the size of the array gets large. S. Y. Kung argues that for an $N \times N$ array, clock skew can grow as large as $O(N^3)$. Whether future circuit designers will find this problem tractable remains to be seen, but we are forced to consider the alternative of asynchronization.

The Wavefront Array Processor or WAP was offered as a compromise between the systolic array and the data-flow machine, which is an asynchronous general-

purpose machine not totally suitable for VLSI implementation. The WAP will be referred to later when the synchronization problem resurfaces in our designs.

We ended this chapter with a discussion of current research here at the SRC involving the real-time implementation of a scalar nonlinear filter. An implicit finite difference scheme for the Zakai equation was cited, and a way of solving the linear system through *LU* decomposition via a linear systolic array was described. Such research is a natural continuation of earlier work done on the real-time systolic implementation of the Kalman filter, which was also briefly described.

References for Chapter 2

[1] Bilardi, G., Pracchi, M. and Preparata, F., "A Critique of Network Speed in VLSI Models of Computation," *IEEE Jour. of Solid State Circuits*, vol. SC-17, No. 4, Aug. 1982.

[2] Davis, R. and Thomas, D., "Systolic Array Chip Matches the Pace of High-Speed Processing," *Electronic Design* vol. 32, no. 22, Oct. 1984.

[3] Fisher, A. and Kung, H. T., "Synchronizing Large VLSI Processing Arrays," *IEEE Trans. on Computers*, vol. C-34, No. 8, Aug. 1985.

[4] Gray, J., ed., *VLSI '81*, Academic Press, 1981.

[5] Kung, S. Y., Arun, K. S., Gal-Ezer, R. and Bhaskar, R., "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computers*, vol. C-31, No. 11, Nov. 1982.

[6] Kung, H. T. and Lam, M., "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *Journal of Parallel and Distributed Computing*, vol. 1, 1984.

[7] Kung, H. T., "Two-Level Pipelined Systolic Arrays for Matrix Multiplication, Polynomial Evaluation and Discrete Fourier Transform," Workshop on Dynamic Behavior of Automata, Luminy, France, 1983.

[8] ——,"Systolic Arrays," Dept. of Computer Sci., Carnegie-Mellon Univ. Pittsburgh, Penn. 1984.

[9] ——, "Why Systolic Architectures," *Computer*, Jan. 1982.

[10] ——, "Systolic Algorithms," in *Large Scale Scientific Computation*, Academic Pr. 1984.

[11] ——, "Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI," from the Proc. of the Soc. of Photo-Optical Instr. Engr., 1980.

[12] LaVigna, A., "Real Time Sequential Detection for Diffusion Signals," Dept. of Electrical Eng., Univ. of Maryland, 1986.

References for Chapter 2

[1] Bilardi, G., Pracchi, M. and Preparata, F., "A Critique of Network Speed in VLSI Models of Computation," *IEEE Jour. of Solid State Circuits*, vol. SC-17, No. 4, Aug. 1982.

[2] Davis, R. and Thomas, D., "Systolic Array Chip Matches the Pace of High-Speed Processing," *Electronic Design* vol. 32, no. 22, Oct. 1984.

[3] Fisher, A. and Kung, H. T., "Synchronizing Large VLSI Processing Arrays," *IEEE Trans. on Computers*, vol. C-34, No. 8, Aug. 1985.

[4] Gray, J., ed., *VLSI '81*, Academic Press, 1981.

[5] Kung, S. Y., Arun, K. S., Gal-Ezer, R. and Bhaskar, R., "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computers*, vol. C-31, No. 11, Nov. 1982.

[6] Kung, H. T. and Lam, M., "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *Journal of Parallel and Distributed Computing*, vol. 1, 1984.

[7] Kung, H. T., "Two-Level Pipelined Systolic Arrays for Matrix Multiplication, Polynomial Evaluation and Discrete Fourier Transform," Workshop on Dynamic Behavior of Automata, Luminy, France, 1983.

[8] — —,"Systolic Arrays," Dept. of Computer Sci., Carnegie-Mellon Univ. Pittsburgh, Penn. 1984.

[9] ——, "Why Systolic Architectures," *Computer*, Jan. 1982.

[10] ——, "Systolic Algorithms," in *Large Scale Scientific Computation*, Academic Pr. 1984.

[11] ——, "Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI," from the Proc. of the Soc. of Photo-Optical Instr. Engr., 1980.

[12] LaVigna, A., "Real Time Sequential Detection for Diffusion Signals," Dept. of Electrical Eng., Univ. of Maryland, 1986.

[13] Mead, C. and Conway, L., *Introduction to VLSI Systems,* Addison-Wesley, 1980.

[14] Parker, D., "Notes on Shuffle Exchange-Type Switching Networks," *IEEE Trans. on Computers,* vol. C-29, No. 3, March, 1980.

[15] Seitz, Charles, "Concurrent VLSI Architectures," *IEEE Trans. on Computers,* vol. C-33, No. 12, Dec. 1984.

[16] Stone, H. "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers,* vol. C-20, No. 2, Feb. 1971.

[17] Travassos, R. H., *Application of Systolic Array Technology ot Recursive Filtering,* Prentice Hall, 1983.

[18] Ullman, J., "Computational Aspects of VLSI," Computer Science, Pr. 1984.

3. Some Qualitative Results on the Zakai Equation (Part I)

1. Introduction

Equations defined on infinite domains clearly must be approximated to bounded domains in order that numerical analysis can be done. This chapter describes some methods of how this can be done for a class of fairly difficult problems. However, we point out that in practice, one usually can obtain a finite domain by numerical experimentation, and when it comes time to do our own numerical work, this is probably what will have to be done, as theoretical estimates are often too conservative.

In this chapter we apply and extend some results of Baras, *et al*, [1], to some nonlinear filtering problems, and we will show that a refinement of their procedure is in order. In particular, we show that some of their constraints on certain growth functions, (which are standard tools in theorems relating to PDEs), and on the functional coefficients, of the Zakai equation, are inherently inconsistent. As a result, they implicitly required that all sample observational paths, (which can viewed Brownian motions) are *all* bounded by the same constant.

Our model problem will be the bilinear filtering problem in one dimension. Certain technical difficulties of the methods of Baras, *et al* will become especially apparent. We also feel that this problem is both tractable enough and sufficiently rich in mathematical insights that it is an excellent vehicle for the ideas we wish to discuss. Extensions to higher dimensions will also be shown to be possible.

Our goal is to demonstrate that, for a sufficiently wide class of problems defined on R^n, a compact set can be constructed such that the solution of the Zakai equation virtually vanishes on its complement. This is clearly a necessary prelude to any kind of numerical analysis.

This chapter is part of a general strategic plan that reads:

1). Begin with an n-dimensional Zakai equation defined on $[0,T] \times R^n$.

2). Define compact set Ω where the solution U in 1), obeys $0 \leq V < \epsilon$ on Ω^c.

3). Construct Finite Difference scheme that converges weakly to U in Ω.

4). Describe Multigrid algorithm that rapidly (in real-time) the resulting linear system in 3).

5). Provide a VLSI implementation of the algorithm in 4).

6). Discuss limitations of the design of 5).

This chapter, together with chapter 4, deals with steps 1) and 2). We begin with a one-dimensional equation as our model problem and then move on to some problems in higher dimensions in the next chapter.

Main Results: refinement of techniques to provide existence, uniqueness and asymptotic estimates of the solution of the Zakai equation. The model problem of the bilinear filtering equation is completely worked out.

2. Transformations on the Zakai Equation of Nonlinear Filtering

Although we will concentrate on the bilinear filtering problem, we begin this section with a quick review of the basic concepts. The general filtering problem in R can be expressed as follows. Consider the state and observation equations,

$$\begin{aligned} dx(t) &= f(x(t))dt + g(x(t))dw(t) \\ dy(t) &= h(x(t))dt + dv(t) \\ x(0) &= x_0, \quad y(0) = 0, \quad 0 \leq t \leq T < \infty \end{aligned} \tag{3.2.1}$$

Here w, v are Wiener processes in R, mutually independent, and independent of x_0 which is a random variable with density $p_0(x)$. The functions f, g, h are smooth ($f \in C(R), g, h \in C^2(R)$) and may grow rapidly as $|x| \to \infty$. They are all given as part of the modeling assumptions. The filtering problem is to estimate the state $x(t)$ given the σ algebra $\mathcal{Y}_t = \sigma\{y(s), 0 \leq s \leq t\}$ created by all the observations taken from 0 to t. The σ algebra can be thought of as having all the available observational information up to time t. It can be shown that the best nonlinear

estimate is $E[x(t)|y_t]$, in the mean-square sense, which is also the first moment of the conditional probability density. Hence, if we had this density in our possession we could not only find the state estimate but other important statistical information as well, such as higher moments. In fact, anything of the form $E[\phi(x(s))|y_t]$, where ϕ is smooth would be permissible.

It turns out that the conditional density is related to a stochastic partial differential equation. More accurately, the conditional density of $x(t)$ given y_t is the normalization of $U(t,x) \geq 0$ which satisfies the Duncan-Mortensen-Zakai (DMZ) equation,

$$\begin{aligned} dU(t,x) =& \left[a(x)U_{xx}(t,x) + b(x)U_x(t,x) + c(x)U(t,x)\right]dt \\ &+ h(x)U(t,x)dy(t) \\ U(0,x) =& p_0(x), \quad 0 \leq t \leq T < \infty \end{aligned} \tag{3.2.2}$$

where

$$\begin{aligned} a(x) &= \frac{1}{2}g^2(x) \\ b(x) &= 2g(x)g_x(x) - f(x) \\ c(x) &= g_x^2(x) + g(x)g_{xx}(x) - f_x(x) - \frac{1}{2}h^2(x) \end{aligned} \tag{3.2.3}$$

The above has been written in the Fisk-Stratonovich form of the stochastic calculus. Derivations of the above can be found in [3].

The key observation at this point is that the problem is posed on an infinite domain R, and that the coefficients are unbounded and in fact may have greater than polynomial growth. Existence and uniqueness of solutions of the DMZ for state processes evolving on a bounded domain in R^n or when the domain is unbounded but the coefficients are bounded and possibly degenerate, have been available form some time. See, for example, Pardoux, (1979, [6]). The case when both domain and coefficients are unbounded has also been studied and theorems were available only in the presence of less than polynomial growth when Baras, *et al* [1] undertook their investigations. (See, for example, the 1979 paper of Pardoux, [6]). The existence and uniqueness theorems stated and proved here deal with the case of f, g, h being

equal to az, bz, cz respectively, the only restriction being that $c \neq 0$. We also give results on the tail behavior of the density which is essential for numerical analysis. By working out in detail this model problem, certain technical shortcomings we found in the methods of Baras, *et al* , will be brought to light, and corrected. Specifically, our proofs require boundedness conditions on the observations, but this is accomplished through a transformation, and does not reflect actual constraints that must be imposed on the observations prior to, or in the course of, filtering.

The DMZ can be transformed to a parabolic differential equation which, for each path $y(t)$ is a conventional, i.e., non-stochastic PDE. This leads us to consider classical methods of proving existence, uniqueness, and asymptotic behavior and so we use the results of Besala [2], whose approach involves the use of a maximum principle and weight functions.

The "robust" transformation can now be introduced. Define,

$$V(t,x) = \exp[-h(x)y(t)]U(t,x) \tag{3.2.4}$$

Now V satisfies an "ordinary" parabolic PDE, for any given path $y(t), 0 \leq t \leq T$,

$$\begin{aligned} V_t(t,x) &= A(x)V_{xx}(t,x) + B(t,x)V_x(t,x) + C(t,x)V(t,x) \\ V(0,x) &= p_0(x), \quad 0 \leq t \leq T \end{aligned} \tag{3.2.5}$$

where

$$\begin{aligned} A(x) &= a(x), \\ B(t,x) &= b(x) + 2a(x)h_x(x)y(t) \\ C(t,x) &= c(x) + b(x)h_x(x)y(t) + a(x)[h_{xx}(x)y(t) + h_x^2(x)y^2(t)] \end{aligned} \tag{3.2.6}$$

This will be the parabolic PDE upon which we will concentrate. It can be noted at this point that the asymptotic results we seek will allow us to construct a bounded domain D, such that we will know prior to solving the PDE, that $V < \epsilon$ on the complement of D, for some given positive ϵ. This will prove essential in our numerical analysis. In particular, we could assume ϵ to be at or less than the machine

tolerance, (for we will ultimately use VLSI systems with limited bit size). We could also further constrain the PDE with boundary conditions set equal to zero.

We apply the above ideas to the specialized case of the general bilinear filtering problem as represented by the following system

$$\begin{aligned} dz(t) &= az(t)dt + bz(t)dw(t) \\ dy(t) &= cz(t)dt + dv(t) \\ z(0) &= z_0, \qquad y(0) = 0, \qquad 0 \le t \le T < \infty \end{aligned} \tag{3.2.7}$$

with $p_0(z)$ being the density of z_0, and $z_0, w(t), v(t)$ are mutually independent. We make no restrictions on a, b, c except that $c \neq 0$. We will also assume that $b \neq 0$ at first and work out the special case when it is separately.

The Duncan-Mortensen-Zakai equation (DMZ) for (1.2.1) is

$$\begin{aligned} dU(t,z) &= [(1/2)(b^2z^2U)_{zz} - (azU)_z - (1/2)c^2z^2U]dt + czUdy(t) \\ U(0,z) &= p_0(z), \qquad (t,z) \in [0,T] \times [0,\infty) \end{aligned} \tag{3.2.8}$$

We can assume z is nonnegative by requiring that z_0 be so, that is, by taking the absolute value of the initial condition. (Since the solution of (3.2.7) can be represented by an exponential, namely $z(t) = z_0 \exp[(a - b^2/2)t + bw_t]$, it will remain positive, zero or negative depending on the value and sign of z_0). The reason for this will become clear. We therefore take $p_0(z)$ to be defined only on $[0,\infty)$ and to be continuous and integrable there.

Our objective in this paper is to determine the existence and uniqueness of solutions of the DMZ for this bilinear case. Two prominent difficulties are the degenerate operator in (3.2.8), and the unboundedness of the coefficients in (3.2.7). Our strategy as discussed before is to obtain, via a series of transformations, a new equation known as the "robust" version of the DMZ. For every given path $y(t)$, we will be dealing with a conventional parabolic PDE. Thus we can apply standard theorems, which in turn involve further transformations, and in particular some results of Besala [2], to determine not only existence and uniqueness, but also

the asymptotic growth of the solutions. Of course, as the transformations will be invertible, there will be a one-to-one correspondence between our results and the original Zakai equation.

To deal with the problem of the operator in (3.2.8) not being uniformly elliptic, we can do one of two things. First we can introduce a logarithmic change of coordinants, $x = \log z$, this being the reason why we wanted $z > 0$. Then the system, (3.2.7) becomes, by the Itô calculus,

$$\begin{aligned} dx &= (a - \frac{b^2}{2})dt + bdw \\ dy &= ce^x dt + dv \end{aligned} \tag{3.2.9}$$

Using this system we can construct the DMZ and then apply the robust transformation which is defined as $V(t,x) = e^{-ce^x y(t)}U(t,e^x)$. Using the formulas for the coefficients for the DMZ we get,

$$\begin{aligned} a(x) &= \frac{b^2}{2} \\ b(x) &= -(a - \frac{b^2}{2}) \\ c(x) &= -\frac{1}{2}c^2e^{2x} \end{aligned} \tag{3.2.10}$$

The coefficients for the robust transformation now become, by (1.6),

$$\begin{aligned} A(x) &= \frac{b^2}{2} \\ B(x,t) &= -(a - \frac{b^2}{2}) + b^2ce^x y(t) \\ C(x,t) &= -\frac{1}{2}c^2e^{2x} + ce^x y(t)\big(-(a-b^2) + \frac{b^2}{2}ce^x y(t)\big) \end{aligned} \tag{3.2.11}$$

The theorems we have in mind can then be applied to the result. Another way is by simply changing coordinants in the original DMZ, using the rule, $\frac{\partial}{\partial x} = e^{-x}\frac{\partial}{\partial x}$. Letting $W(t,x) = U(t,e^x)$, we have,

$$\begin{aligned} dW(t,x) = &\Big[(\frac{b^2}{2})W_{xx} + [\frac{3b^2}{2} - a]W_x + [b^2 - a - \frac{c^2}{2}e^{2x}]W\Big]dt \\ &+ ce^x W dy(t) \\ W(0,x) = &\, p_0(e^x) \end{aligned} \tag{3.2.12}$$

Using the appropriate coefficients as before, we can convert this equation into the robust DMZ. The coefficients of this new equation are:

$$\begin{aligned} A(x) &= \frac{b^2}{2} \\ B(x,t) &= [\frac{3}{2}b^2 - a] + b^2 c e^x y(t) \\ C(x,t) &= [b^2 - a - \frac{c^2}{2}e^{2x}] + [\frac{3}{2}b^2 - a]ce^x y(t) \\ &\quad \frac{b^2}{2}[ce^x y(t) + c^2 e^{2x} y^2(t)] \end{aligned} \tag{3.2.13}$$

The reader will note that the two transformations do not lead to the same result, and we can express this difference by the following diagram:

Original system (3.2.7) $\Rightarrow$ Change coordinants in system

$\Rightarrow$ Derive DMZ $\Rightarrow$ Robust Eqn.

Original system (3.2.7) $\Rightarrow$ Derive DMZ

$\Rightarrow$ Change Coordinants in DMZ $\Rightarrow$ Robust Eqn.

We have different results because the two operations can be thought of as group transformations that do not commute. In fact, (3.2.12) does not actually "solve" the filtering problem posed by (3.2.9). In fact, (3.2.12) does not correspond to any filtering problem of the form (3.2.1), but that does not matter as far as proving existence, uniqueness and asymptotic behavior. The main thrust of our approach is in providing an invertible transformation from the original problem to one that is tractable by the methods of Besala [2].

Thus, although our theorems can be applied to both systems, we will concentrate on the second transformation.

We will now introduce the following transformations, which include a weight function, $\psi(x,t)$ soon to be determined.

$$\begin{aligned} V(t,x) &= W(t,x)\exp[-ce^x y(t)] \\ u(t,x) &= V(t,x)\exp[\psi(x,t) - \gamma t] \end{aligned} \tag{3.2.14}$$

correspond to the transformed robust equation

$$\begin{aligned} u_t &= a(x)u_{xx}(t,x) + b(t,x)u_x(t,x) + c(t,x)u(t,x) \\ &(t,x) \in (0,T) \times R \\ u(0,x) &= p_0(e^x)\exp[\psi(x,0)] \end{aligned} \tag{3.2.15}$$

This follows by using the robust DMZ in (3.2.5) and applying it to (3.2.14). We obtain,

$$\begin{aligned} u_t - u(\psi_t - \gamma)e^{-(\cdot)} &= V_t \\ -C(x,t)ue^{-(\cdot)} &= -C(x,t)V \\ -B(x,t)[u_x - u\psi_x]e^{-(\cdot)} &= -B(x,t)V_x \\ -A(x)[u_{xx} - 2u_x\psi_x + u(\psi_x)^2 - u\psi_{xx}]e^{-(\cdot)} &= -A(x)V_{xx} \end{aligned} \tag{3.2.16}$$

Obviously, $\exp -(.) = \exp -(\psi(x,t) - \gamma t)$. Adding these terms up gives equation (3.2.15).

At this point, we require only the following:

$$\begin{aligned} &1).\ \psi(x,t) \to +\infty \text{ as } |x| \to +\infty \text{ and is } C^2(R) \text{ for all } t. \\ &2).\ \psi(x,t) \text{ is } C^1([0,T]) \text{ for each } x. \end{aligned} \tag{3.2.17}$$

We also have, from the foregoing calculation,

$$\begin{aligned} a(x) &= \frac{b^2}{2} \\ b(t,x) &= -b^2\psi_x - (a - \frac{3b^2}{2}) + b^2ce^xy(t) \\ c(t,x) &= \frac{b^2}{2}[(\psi_x)^2 - \psi_{xx}] - \psi_x[b^2ce^xy - (a - \frac{3b^2}{2})] \\ &\quad + \frac{b^2}{2}[(ce^xy)^2 + ce^xy] \\ &\quad - ce^xy[a - \frac{3b^2}{2}] \\ &\quad (b^2 - a - c^2e^{2x}/2) - \gamma + \psi_t \end{aligned} \tag{1.18}$$

The reader should note that (3.2.8) is a nondegenerate parabolic equation. We will exploit the growth conditions so as to identify the dominant terms in the potential $c(t,x)$ in (3.2.18) and in the potential of the adjoint of (3.2.15). Proper selection of

the weight functions in accordance with given constraints will permit us to guarantee that these potentials are non-positive. This will in turn set the stage for using a maximum principle. It should perhaps be noted that the use of weight functions and maximum principles are standard techniques in the existence and uniqueness theorems of parabolic equations.

Our goal is to show that the robust equation (3.2.5) has a fundamental solution under the conditions discussed. To quickly review, a fundamental solution of (3.2.5) is a real valued function $\Gamma(t,x;s,z)$ defined for $0 \leq, t \leq T$, $x, z \in R$, which satisfies the following conditions:

1). As a function of (t,x), Γ has continuous derivatives $\Gamma_t, \Gamma_x, \Gamma_{xx}$ and satisfies (1.5) in $(s,T) \times R$;

2). If $\rho(x)$ is continuous and has compact support, then

$$\lim_{\substack{t \downarrow s \\ x \to z}} \int_{-\infty}^{\infty} \Gamma(t,x;s,v)\rho(v)\,dv = \rho(z)$$

Roughly speaking, a fundamental solution, $\Gamma(t,x;s,z)$ when used as a kernel within an integral, (which then forms a linear operator), takes initial data at time s, and transforms it to a solution of equation (3.2.5), evaluated at time t and point x, which obeys the given initial conditions. Furthermore, properties of the fundamental solution determine the character of any other solution. Therefore, questions of boundedness and asymptotic behavior can be readily settled. We will use this fact in the sequel.

3. Existence and Uniqueness Theorems for the Robust Zakai Eqn.

Our results will be stated and proved in a series of theorems based upon the fundamental solution of a parabolic partial differential equation. These follow Besala, [2]. The first forges a connection between the solutions of the transformed and the original version of the robust equation. A central idea is that we must prove the existence of the Zakai solution for each path of the observation and to this end it is necessary that each path be normalized so as to remain within a certain bound. Thus we can say that for any given path $y(t)$, there exists an invertible transformation from it to a path which is contained within a certain bound. This bound, to be calculated below, will be the same for all paths, and we will say that the paths have been normalized when transformed in this way. The reader should not be confused with the fact that while each $y(t)$ can be viewed as a Brownian motion under an appropriate measure, and is therefore continuous on a closed interval and hence bounded, the family of possible sample paths is not uniformly bounded unless normalized.

We can now state our

Theorem 1: For each Hölder continuous path $\{y(t), 0 \leq t < \infty\}$ of the observation process there exists a constant $\alpha > 0$, dependent on the path, such that for each $t, |y(t)|/\alpha \leq 1/3|b|$, where b is the diffusion coefficient in (3.2.7), and an associated fundamental solution $\tilde{\Gamma}(t, x; s, z)$ of (3.2.15). Moreover,

$$\Gamma(t, x; s, u) = \tilde{\Gamma}(t, x; s, u) \exp[\psi(u, t) - \psi(x, s) + \gamma(t - s)] \qquad (3.3.1)$$

is a fundamental solution of the robust DMZ equation (3.2.5) on $[0, T]$. In addition, $\tilde{\Gamma}(t, x; s, u)$ satisfies the inequalities

$$0 \leq \tilde{\Gamma}(t, x; s, u) \leq K/(t - s)^{\frac{1}{2}} \qquad (3.3.2)$$

for some constant K and $x, z \in R$, $t \in [0,T]$ and

$$\begin{aligned} \int_{-\infty}^{\infty} \tilde{\Gamma}(t,x;s,u)\,du &\leq 1 \\ \int_{-\infty}^{\infty} \tilde{\Gamma}(t,x;s,u)\,dx &\leq 1 \end{aligned} \tag{3.3.3}$$

Before proving this theorem, we first state a technical lemma found in Besala [2].

Lemma : (Besala, [2]) Let $a(t,x), b(t,x)$, and $c(t,x)$ be real-valued together with a_x, a_{xx}, b_x be locally Hölder continuous in $D = (t_0, t_1) \times R$. Assume that,

a). $a(t,x) \geq \lambda > 0, \qquad \forall (t,x) \in D$ for some constant λ .

b). $c(t,x) \leq 0 \qquad \forall (t,x) \in D$

c). $(c - b_x + a_{xx})(t,x) \leq 0 \qquad \forall (t,x) \in D$

Then for the Cauchy problem,

$$\begin{aligned} u_t(t,x) &= a(t,x)u_{xx} + b(t,x)u_x + c(t,x)u \\ u(0,x) &= u_0(x) \qquad (t,x) \in D \end{aligned} \tag{3.3.4}$$

has a fundamental solution $\Gamma(t,x;s,u)$ which satisfies

$$0 \leq \Gamma(t,x;s,u) \leq K/(t-s)^{\frac{1}{2}} \tag{3.3.5}$$

for some positive constant K and

$$\begin{aligned} \int_{-\infty}^{\infty} \Gamma(t,x;s,u)\,du &\leq 1 \\ \int_{-\infty}^{\infty} \Gamma(t,x;s,u)\,dx &\leq 1 \end{aligned} \tag{3.3.6}$$

Moreover, if $u_0(x)$ is continuous and bounded, then

$$u(t,x) = \int_{-\infty}^{\infty} \Gamma(t,x;t_0,w)u_0(w)\,dw \tag{3.3.7}$$

is a bounded solution of (3.2.15). For a proof see Besala, [2]. It is worth noting at this point that is $u_0(x)$ is non-negative, so is $u(x,t)$, as Γ is also non-negative. Boundedness of $u(t,x)$ follows by similar reasoning.

Proof of Theorem 1:

According to the theorem by Besala we need to show conditions $a), b), c)$ are satisfied by the coefficients of the transformed robust PDE. Clearly, $a)$ is satisfied whenever $b \neq 0$. (As stated before, the special case of $b = 0$ will be investigated later).

Now to condition $b)$. We see that e^{2x} appears to be a dominant term in $c(t,x)$ so that whatever $\psi(x,t)$ is it must offset this growth to keep $c(t,x)$ negative. Thus it seems, upon inspection, worth trying $\psi(x,t) = A(t)e^x + Bx$. We see at once that for this function to qualify under our restrictions in (3.2.17) we must have $A(t)$ in $C^1([0,T])$ and $B < 0$. We also need $A(t) > 0$.

Plugging this term into $c(t,x)$ we get

$$\begin{aligned} & e^{2x}\left[\frac{b^2}{2}A^2(t) - A(t)b^2cy + \frac{b^2}{2}c^2y^2 - \frac{c^2}{2}\right] \\ & + e^x\left[(a-2b^2)A(t) + (2b^2-a)cy + b^2B(A(t)-cy) + A'(t)\right] \\ & + \frac{b^2B^2}{2} + B(a - \frac{3b^2}{2}) + b^2 - a - \gamma = c(t,x) \end{aligned} \tag{3.3.8}$$

Since $A(t)$ and $y(t)$ are assumed to be bounded, as $x \to -\infty$ we have no problem if the constant terms are non-positive. The key problem is that the coefficient of e^{2x} must be negative, so we have upon using the quadratic formula with $A(t)$ as the unknown, the requirement that $A(t)$ lie between the roots of a concave up parabola, these roots are,

$$\begin{aligned} A(\pm) &= \frac{b^2cy \pm \sqrt{(b^2cy)^2 - 4\left[\frac{b^2}{2}\left(\frac{b^2c^2y^2}{2} - \frac{c^2}{2}\right)\right]}}{b^2} \\ &= \frac{b^2cy \pm |bc|}{b^2} \end{aligned} \tag{3.3.9}$$

and thus we have the condition,

$$A(t) \in (cy(t) - |\frac{c}{b}|, cy + |\frac{c}{b}|) \tag{3.3.10}$$

In fact we will also require for reasons that will become clear, the more stringent conditions that

$$cy(t) < A(t) < cy(t) + |\frac{b}{c}| \text{ and } A(t) > 0 \tag{3.3.11}$$

Now suppose $y(i)$ to have been normalized and for now we will express this by saying $|y(t)| < k/|b|$ where $k > 0$, and to be chosen below. Now we ask, can $A(t)$ be a constant and still satisfy the above inequality? If so, we must have, for $A(t) \equiv A$, a constant,

$$k|\frac{c}{b}| < A < (1-k)|\frac{c}{b}| \tag{3.3.12}$$

which can be expressed in words by saying, A must be greater than the greatest lower bound, and less that the least upper bound formed by the inequality in (3.3.11). We see that $k = 1, 1/2$ are unacceptable, while $k = 1/3$ will work. Thus we set

$$\alpha = \max_{0 \le t \le T} |y(t)|3|b| \tag{3.3.13}$$

and we will assume henceforth that each $y(t)$ has been normalized by its appropriate α.

We see at once that if $y(t)$ has been normalized by α so that the magnitude of the observations does not exceed $1/3|b|$ then we can, for example, set $A = |\frac{c}{2b}|$ and equation (3.3.12) will be valid for all t and we will also have $A > 0$. From now on we will assume that A takes on only constant values in the interval formed by (3.3.11), and we reflect this fact by setting $\psi(x,t) = \psi(x) = Ae^x + Bx$. The coefficient of e^x may be positive or negative, for large values of positive x will always be superceded by the e^{2x} term. (Note that since $A(t)$ is now a constant, $A'(t) \equiv 0$ which is fortunate as we would have had no control over this term). However, it will be desirable to make a careful choice of B. To see this, maximize $c(t,x)$ for each fixed t by differentiating with respect to x. We obtain an expression of the form:

$$2e^{2x}(\text{coefficient }) + e^x(\text{coefficient}) = 0$$

Recalling that the coefficient of e^{2x} was made strictly negative, if the coefficient of e^x were also strictly negative or zero, the maximum of $c(x,t)$ would occur at $x = -\infty$, which can be readily calculated. Now the coefficient of e^x in $c(t,x)$ is, when made negative,

$$b^2B[A - cy] - (2b^2 - a)[A - cy] < 0 \tag{3.3.14}$$

and since $A - cy > 0$ we must have,

$$B \leq \min\left[0, \frac{(2b^2 - a)}{b^2}\right] \tag{3.3.15}$$

where we have also required $B < 0$ as part of condition 1) in (3.3.17). Of course, when $x \to -\infty$ only the constant terms will remain so they must be negative as well. Thus $c(t,x)$ is a smooth function which asymptotically goes to 0 as $x \to +\infty$. Thus we can choose γ to insure that $c(t,x) \leq 0$. These constant terms are, when made non-positive,

$$\frac{b^2}{2}B^2 - B\left[\frac{3b^2}{2} - a\right] + (b^2 - a) - \gamma \leq 0 \tag{3.3.16}$$

From this, γ can be chosen. Therefore, condition b) is satisfied.

Condition c) is automatically satisfied (at least asymptotically) since the subtraction of b_x from c changes only the coefficient of the e^x term. (Note that $a_{xx} = 0$). However, we must choose a new value of B if we wish for the maximum of the term in condition c) to occur at $x = -\infty$. The new coefficient of e^x, when made negative, is

$$[b^2 B - (b^2 - a)](A - cy) \leq 0 \tag{3.3.17}$$

To maintain both this requirement and the one in (3.3.15), we demand that

$$B < \min\left[0, (1 - \frac{a}{b^2})\right] \tag{3.3.18}$$

Since we now know where the maximum of $c(t,x)$ and $c - b_x$ occur we can easily calculate it, and using γ meet both conditions. We see that

$$\gamma \geq \frac{b^2}{2}B^2 - B(\frac{3}{2}b^2 - a) + (b^2 - a) \tag{3.3.19}$$

where B obeys (3.3.18).

Thus all conditions for Besala's theorem are satisfied. **Q.E.D.**

Our next theorem will show that the solution of the DMZ equation is unique within a certain class of functions.

Theorem 2: Assume $y(t)$ has been normalized so that $|y(t)| \leq 1/3|b|$. And suppose further that

$$\begin{aligned} & p_0(e^x)\exp(\theta_1 e^x - \theta_2 x) \leq M \quad \forall x \in R \quad M < \infty \\ & \text{for } \theta_i > 0,\ i = 1,2 \quad \theta_1 \leq \frac{2}{3}|\frac{c}{b}|, \quad -\theta_2 \leq 1 - \frac{a}{b^2} \end{aligned} \tag{3.3.20}$$

Then for all positive $\tilde{\theta}_i < \theta_i, i = 1,2$ there exists a unique solution to the DMZ equation (3.2.8) within the class of functions

$$\lim_{|x|\to+\infty} \sup U(t,e^x)\exp[\tilde{\theta}_1 e^x - \tilde{\theta}_2 x] \to 0 \tag{3.3.21}$$

Proof: Let ϵ be a constant with

$$0 < \epsilon < \frac{1}{12}|\frac{c}{b}| \tag{3.3.22}$$

and define,

$$\begin{aligned} \tilde{\psi}(x) &= \psi(x) + \epsilon[e^x - x] \\ \tilde{u}(t,x) &= u(t,x)\exp\left[\epsilon(e^x - x)\right] \end{aligned} \tag{3.3.23}$$

Then $\tilde{u}$ also satisfies an equation of the form (3.2.15), whose coefficients are formed by replacing $\psi(x)$ with $\tilde{\psi}(x)$ in the definitions of the coefficients in (3.2.18). Choose $A = |\frac{c}{2b}|$ so that it satisfies the inequality

$$cy(t) < A < cy(t) + |\frac{c}{b}| \tag{3.3.24}$$

Thus $\tilde{u}(x,t)$ will have a solution (by the Besala lemma) provided that

$$A + \epsilon \in (cy(t) - |\frac{c}{b}|, cy(t) + |\frac{c}{b}|)$$

but this is guaranteed by our restrictions on $A, y(t)$, and ϵ since,

$$\begin{aligned} cy(t) \leq A + \epsilon &= \frac{7}{12}|\frac{c}{b}| \\ &\leq \frac{2}{3}|\frac{c}{b}| \leq cy(t) + |\frac{c}{b}| \end{aligned} \tag{3.3.25}$$

Therefore, $u, \tilde{u}$ exist and are bounded, while U is given by (3.2.8). Since $\epsilon > 0$, $u(t,x) \to 0$ as $|x| \to \infty$, applying the maximum principle as in [5], [6], shows that u is unique in the class of functions that tend to zero a $|x| \to \infty$. (actually, the method of Besala [2] only applies to this class). In the coordinants of (3.2.12) this original class consists of those functions satisfying:

$$U(t, e^x) \exp[Ae^x - \gamma t - ce^x y(t)] \to 0 \text{ as } |x| \to \infty \tag{3.3.26}$$

Note that the requirement that $A > cy(t)$ implies $U \to 0$ as $|x| \to \infty$. The choice of parameters (3.3.22), (3.3.24) satisfies all the conditions (3.3.20) and the result follows. **Q.E.D.**

We are now in a position to state exactly where we corrected the methods of Baras, *et al*, [1].

Claim: The methods of Baras, *et al*, [1] *implicitly* required that the family of all possible sample paths of $y(t)$ each be bounded by the *same* constant.

To understand why, we adopt the same notation as in their 1983 paper. They defined a sequence of stopping times:

$$\begin{aligned} t_0 &= 0, \\ t_{k+1} &= \inf_{t > t_k} \{t : |y(t) - y(t_k)| = \epsilon\} \end{aligned} \tag{3.3.27}$$

for

$$0 < \epsilon < \frac{1}{4} \lim \sup_{|x| \to \infty} z|h_x(z)|/(h^2 + f^2/g^2)^{1/2} \tag{3.3.28}$$

The reader will see at once that in our case, since $f(z) = az$, $g(z) = bz$, and $h(z) = cz$, the limit on the right hand side is $\frac{1}{4} \cdot 1$.

Now on each subinterval (t_k, t_{k+1}), we define,

$$\psi^k = \alpha\phi_1(x) + \beta_1^k \phi_2(x) + \beta_2[1 + \phi_2^2(x)]^{1/2} \tag{3.3.29}$$

where α, β_2 are to be defined and the set $\{\beta_1^k\}$ is dependent on $y(t_k)$.

also,

$$\phi_1(x) = -\int_1^x f(s)/s^2\, ds|_{x=e^z} = -ax$$
$$\phi_2(x) = h(e^x) = ce^x \tag{3.3.30}$$

Just as we did, Baras, *et al* required that

$$\phi(x) \to +\infty \text{ as } |x| \to +\infty \tag{3.3.31}$$

This is clearly necessary as, once u^k is shown to be bounded,

$$V^t, x) \sim u^k(t,x)e^{-\phi^k(x)} \to 0 \text{ as } |x| \to \infty$$

which is what we desire. Thus, for this to happen, the definition of $\phi^k(x)$ implies that

$$\beta_2 > |\beta_1^k| \tag{3.3.32}$$

as it is the e^x that will become dominant for large x.

Using the same strategy as presented in these pages they found the $c(t,x)$ coefficient of the transformed robust equation, (which for them was denoted $c^k(t,x)$, as it was defined on each subinterval (t_k, t_{k+1}), and after Besala, required that

$$c(t,x) \leq 0 \qquad \forall x \in R$$

Working out the algebra, they found that they must have

$$|\beta_2 \pm (\beta_1^k - y(t))| < 1/2 \tag{3.3.33}$$

on each (t_k, t_{k+1}). With an eye on the above equation, they derived the natural choice of parameters as

$$0 < \beta_2 < 1/4$$
$$\beta_1^k = y(t_k) \tag{3.3.34}$$

The condition that $|y(t) - y(t_k)| < \epsilon < 1/4$ provides for the inequality (3.3.33). But the constraint in (3.3.32) that

$$\beta_2 > |\beta_1^k| = y(t_k)$$

implies that all samples $y(t)$ are bounded in this way.

The only way out of this problem is to argue that for each sample path $y(t)$, there exists a linear map

$$y(t) \rightarrow y(t)/\alpha$$

such that the resulting sample paths are bounded appropriately.

We still argue that the basic strategy derived by Baras, *et al* is sound, as we are obviously still following it despite some modifications. However, bounding each $y(t)$ by the same constant resolve the existence and uniqueness issue, but it dones not conveniently allow us to calculate the asymptotic bounds on solutions of the Zakai equation.

We conclude that while the concept of normalizing the observational paths is a useful theoretical device for purposes of proving existence and uniqueness, we are not advocating that all observational path $y(t)$ be bounded by the same constant prior to filtering. This fact is especially pertinent in any kind of "real time" signal processing, where we cannot see our observational paths in advance of computation. Indeed, although each path is bounded, the family of all possible sample paths is not uniformly bounded, so there is always the chance that some path will exceed any given bound.

We now introduce a more natural method that solves these difficulties. It will keep $|y(t)| \leq 1/3|b|$, and will do so without scaling the sample path. This technique shall be called *resetting*.

To understand the basic principle of resetting, consider first any path $y(t)$ that has been normalized, i.e. $|y(t)| \leq 1/3|b|$. Let $t_0 = 0$, and form $\{t_k\}$, $k = 1, n$, the sequence of hitting times where $|y(t_k)| = \epsilon > 0$. This creates a sequence of half-open subintervals $[t_k, t_{k+1})$, and on each such subinterval we reset $y(t_k) = 0$, although $dy(t)$ remains the same as it would without resetting. Indeed, the path has simply been translated by amount equal to ϵ in magnitude so that its initial value is zero on $[t_k, t_{k+1})$.The dynamics have remained unaltered.

Main Observation: The solution to the Zakai equation is driven by the differential $dy(t)$, and is not dependent on the magnitude of $y(t)$.

Under an appropriate change of measure $y(t)$ is a Brownian motion, and hence the number of resetting times must be finite for the interval $[0,T]$, as $y(t)$ will almost surely not cross a level curve infinitely often in a bounded time interval.

Now we know that $U(t,e^x)$ has a unique solution on $[0,t_1)$, with $U(0,e^x) = p_0(e^x)$.Now on $[t_1,t_2)$, we consider the Zakai equation (3.2.8), but with the resetting of $y(t_1) = 0$ and the initial condition,

$$U(t_1,e^x) = \lim_{\substack{t\to t_1 \\ t<t_1}} U(t,e^x) \equiv U(t_1^-,e^x) \tag{3.3.35}$$

(The limit will make sense since continuity will be guaranteed by the Besala lemma). This equation too will have a unique solution, due to our earlier results, and since $dy(t)$ is the same as what it was before, without resetting, the solution must be the same as it was before on this interval. It can be said that since it is the dynamics of $y(t)$ that drive the Zakai equation, it does not notice the translations of $y(t)$ on each subinterval. The technique is easily continued on the remaining subintervals.

What are the corresponding changes for the robust transformations? We have

$$V^k = \exp[-ce^x y(t)]U(t,e^x), \quad (x,t) \in R \times [t_k,t_{k+1})$$
$$V_t^k = A(x)V_{xx}^k + B(x,t)V_x^k + C(x,t)V^k$$

$$V^k(t_k,x) = \begin{cases} U(t_k^-,x), & k \geq 1 \\ p_0(e^x) & k = 0 \end{cases} \tag{3.3.36}$$

Thus on each subinterval $U(t,e^x)$ will be recoverable from $V^k(t,x)$. Using resetting we can keep $|y(t)| < 1/3|b|$, without the need for re-scaling $y(t)$.

We therefore have the

Claim: The solution $U(t,x)$ is invariant under resetting.

We already know that we have a unique solution $U(t,x)$ to the Zakai equation. Thus while $V(t,x)$ and $e^{-ce^x y(t)}$ undergo the effects of translating $y(t)$ on each

subinterval $[t_k, t_{k+1})$, the product,

$$V(t,x)e^{ce^x y(t)} = U(t,x)$$

is invariant.

We now have

Theorem 3: Suppose resetting to be used, with $|y(t)| \leq 1/3|b|$ for all $t \in [0,T]$. Also assume $p_0(e^x)$ to be as in Theorem 2, with the same inequality as before (3.3.20). Then for any $T < \infty$, there exist positive constants M, K, which may depend on the path $\{y(t), 0 \leq t \leq T\}$ such that the solution of the DMZ given by (3.2.8) satisfies:

$$U(t,e^x) \leq M \exp[-K(\exp(x) - x)] \quad \forall (t,x) \in [0,T] \times R \tag{3.3.37}$$

Proof: The bound in (3.3.37) is a simple application of the well-known comparison theorem for parabolic partial differential equations. Suppose $w_1(t,x), w_2(t,x)$ satisfy

$$\begin{gathered} w_1(r,x) \leq w_2(r,x) \\ Tw_1(t,x) \leq Tw_2(t,x) \\ (t,x) \in (r,s) \times R \end{gathered} \tag{3.3.38}$$

where $T = \frac{\partial}{\partial t} - \mathcal{L}$ is a parabolic operator and that $w_1, w_2, \to 0$ as $|x| \to \infty$ uniformly for $t \in (r,s)$. Then,

$$w_1(t,x) \leq w_2(t,x), \quad \forall (t,x) \in (r,s) \times R \tag{3.3.39}$$

Let $t_0 = 0$ and form a finite sequence of resetting times $\{t_k\}$ with $t_k \leq T$. Hence when $y(t)$ hits $1/3|b|$ or some number very close to it, we reset $y(t_k) \equiv 0$ and proceed as described before.

Now let the parameters A, γ be defined as in the proof of Theorem 1, and let (r,s) be any one of the intervals $(t_k, t_{k+1}), k \geq 0$. Let $\mathcal{L}$ be the operator in

the transformed robust equation (3.2.15) and define, for each subinterval, using a natural notation,

$$\begin{aligned} w_1(t,x) &= u^k(t,x) \\ w_2(t,x) &= \exp[\lambda t - \mu\phi_{(x)}] \end{aligned} \tag{3.3.40}$$

for some positive constants λ, μ to be chosen and

$$\phi(x) = \exp(x) - x \tag{3.3.41}$$

To put the problem in a familiar form, we set $-Tw_2 \leq 0$, which implies

$$\begin{aligned} &-\lambda + \frac{b^2}{2}[(\mu(e^x-1))^2 - \mu e^x] \\ &-\mu(e^x-1)[-(Ae^x+B)b^2 + (3b/2-a) + b^2 c e^x y] \\ &+\frac{b^2}{2}[(Ae^x+B)^2 - Ae^x] - (Ae^x+B)[3b^2/2 - a + b^2 cye^x] \\ &[b^2 - a - \frac{c^2}{2}e^{2x}] + (3b^2/2-a)ce^x y \\ &\frac{b^2}{2}[ce^x y + c^2 e^{2x} y^2] - \gamma \leq 0 \end{aligned} \tag{3.3.42}$$

Again the coefficient in front of e^{2x} is the deciding factor, for it must be negative. This coefficient, when made negative is,

$$\mu^2\frac{b^2}{2} + \mu b(A-cy) + \frac{b^2}{2}(A-cy)^2 - \frac{c^2}{2} < 0 \tag{3.3.43}$$

Since the last term is already designed to be negative, we might be tempted to make

$$b^2\mu[\mu/2 + (A-cy)] \leq 0$$

but since μ is positive, this is clearly impossible as $A - cy > 0$. Thus we wish for μ to be between the roots of a concave up quadratic in μ. These roots are,

$$\begin{aligned} \mu(\pm) &= \frac{-b^2(A-cy) \pm \sqrt{(A-cy)^2 b^4 - 4\frac{b^2}{2}[\frac{b^2}{2}(A-cy)^2 - \frac{c^2}{2}]}}{b^2} \\ &= -(A-cy) \pm |\frac{c}{b}| \end{aligned} \tag{3.3.44}$$

Given that $A = |\frac{c}{2b}|, |cy| \leq |\frac{c}{3b}|$, the smallest positive root that (3.3.44) can have is $|\frac{c}{6b}|$, thus we can set

$$\mu = \frac{1}{12}|\frac{c}{b}| \tag{3.3.45}$$

As we did before in Theorem 1, let us see if the coefficient of e^x in (3.3.42) is negative so as to insure that the maximum occurs at $x = -\infty$. From this we can calculate λ to make $-Tw_2 \leq 0$. This coefficient, when made negative, must satisfy,

$$b^2 B[\mu + (A - cy)] < b^2\mu^2 + [\mu + (A - cy)](2b^2 - a) + \mu(A - cy) \qquad (3.3.46)$$

or, equivalently,

$$B < [2 - \frac{a}{b^2}] + \frac{\mu^2}{[\mu + (A - cy)]} + \frac{\mu(A - cy)}{b^2[\mu + (A - cy)]} \qquad (3.3.47)$$

But we have already required that $B < \min[0, 1 - \frac{a}{b^2}]$ so (3.3.47) is automatically satisfied since its right hand side is $1 - a/b^2 +$ positive terms.

Since we have guaranteed that the maximum of $-Tw_2$ occurs at $x = -\infty$, we can now calculate the value of λ to insure that this term is non-positive. We need only the terms associated with μ as the other constant terms are accounted for by γ. We must have

$$\mu^2\frac{b^2}{2} - \mu B b^2 + \mu(\frac{3b^2}{2} - a) \leq \lambda \qquad (3.3.48)$$

We can now show that this value of λ will be valid for all the subintervals, which will in turn give us a uniform bound on $V(t, x)$. By the comparison theorem, and the definitions of u, U and V, we have,

$$u^0(t, x) \leq \exp[\lambda t - \mu\phi(x)] \text{ for } t \in [0, t_1) \qquad (3.3.49)$$

At $t = 0$, $u^0 = p_0(e^x)\exp[Ae^x + Bx] \leq \exp[-\mu\phi(x)]$ since, using the notation of Theorem 2,

$$\begin{aligned} \tilde{\theta}_1 = A + \mu = |\frac{7c}{12b}| < \theta_1 = |\frac{2c}{3b}| \\ -\tilde{\theta}_2 = B + \mu \leq -\theta_2 \leq 1 - \frac{a}{b^2} \end{aligned} \qquad (3.3.50)$$

Thus we see that conditions for uniqueness are satisfied, as long as $B < -\mu$. (This additional restriction on B will be discussed later). We therefore have, by virtue of the comparison theorem,

$$U(t, e^x) \leq e^{(\lambda+\gamma)t}e^{-[\mu+(A-cy)]e^x - [B+\mu]x} \qquad (3.3.51)$$

for $t \in [0, t_1)$. Because of continuity, as $t \to t_1$, the above inequality will still be valid, although $y(t)$ will be reset, with $y(t_1) \equiv 0$. But $A - cy \geq |\frac{c}{6b}|$, thus this term can be replaced in (3.3.51) and the bound will be valid for all the following subintervals.

To get a tighter bound for negative x, we see at once that we should have $B < -\mu$, and indeed, we saw this condition before in meeting the uniqueness rement. One might suspect that since B can be any large negative number, that we should increase the magnitude of B as much as possible. Unfortunately, there is a B^2 term in the definition of γ, (see (3.3.16)). But we can minimize our bound by differentiating the sum $\gamma + \lambda$ by B, to get

$$Bb^2 - (\frac{3b^2}{2} - a) - \mu b^2 = 0 \tag{3.3.52}$$

or,

$$B = (\frac{3}{2} - \frac{a}{b^2}) + \mu \tag{3.3.53}$$

But we already have,

$$B < \min\left[-\mu, 1 - \frac{a}{b^2}\right] \tag{3.3.54}$$

so we see that B should not be too large in magnitude. Thus we will set,

$$B = \min\left[-(\mu + 1), 1 - \frac{a}{b^2}\right] \tag{3.3.55}$$

and this is our final definition of B.

We summarize our result by stating, in the original coordinants,

$$U(t, z) \leq Mz^K \exp(-Kz) \quad \forall (t, z) \in [0, T] \times (0, \infty) \tag{3.3.56}$$

where,

$$\begin{aligned} M &= \exp\left[(\mu^2 b^2/2 + b^2 \mu B(B/2 - 1) + (\mu - B)(3b^2/2 - a) + b^2 - a)T\right] \\ K &= \min\left[-(B + \mu), |\frac{c}{4b}|\right] \end{aligned} \tag{3.3.57}$$

where B is (3.3.55) and μ is (3.3.45).

To get a bound for V we simply multiply (3.3.51) by $\exp[-ce^x y(t)]$, and get,

$$V(t,x) \leq M \exp\left[-K_1(e^x - x)\right] \tag{3.3.58}$$

where M is the same as in (3.3.57) but,

$$K_1 = \min\left[-(B+\mu), |\frac{7c}{12b}|\right] \tag{3.3.59}$$

and again this bound will be valid over all subintervals formed by resetting. **Q.E.D.**

In effect we have two ways to obtain the necessary asymptotic bounds. One is by scaling $y(t)$, the other is by resetting. We believe that resetting is a more natural approach as it emphasizes the fact that the Zakai equation is dependent on the dynamics of $y(t)$ (i.e., $dy(t)$) and not on the magnitude of $y(t)$, and resetting preserves $dy(t)$ while scaling does not, in that we would have the transformation

$$dy(t) \rightarrow dy(t)/\alpha$$

Finally, it should be clear that we a have an asymptotic bound for $U(t,z)$ such that

$$U(t,z) \rightarrow 0 \text{ as } z \rightarrow 0$$

$$U(t,z) \rightarrow 0 \text{ as } z \rightarrow +\infty$$

where we have actually been able to compute the parameters of the asymptotic bound. From this we could, given an $\epsilon > 0$, construct a domain $\Omega = (0, z_{max})$ where

$$0 \leq U(t,z) \leq \epsilon \text{ for } z \geq z_{max}$$

4. Comments and Further Results

Remark 1: One might wonder if the need for resetting can be avoided by the use of scaling of the form $x = k_1 \log z + k_2$. This will only amplify the width of the interval formed by the roots $A(\pm)$ in the coefficient of e^{2x} in (3.3.9), as below,

$$A(\pm) = \frac{e^{-k_2/k_1} c}{k_1}(y(t) \pm |\frac{1}{b}|) \tag{3.4.1}$$

We still need $|y(t)| \leq 1/3|b|$ if we want at least one of the roots to always be strictly positive to make $A(t)$ a positive constant.

Remark 2: It is perhaps worth mentioning that stability, or the lack of it, in the state equation, does not play an important role in the effectiveness of our method, as far as determining the parameters A, B, μ, γ is concerned. This is true for stability in the sense of Lyapunov, where $a - b^2/2 < 0$, or in the sense of requiring the variance of z_t to tend to zero as $t \to +\infty$. The latter condition demands that $a < -b^2/2$.

Remark 3: It might be asked: why did we not use $\psi(x) = \cosh(x)$ as our weight function? It certainly seems it would have yielded more symmetry in our bound for $\pm x$. But we see that ψ_x is proportional to $e^x - e^{-x}$, and when we calculate $(\psi_x)^2$ in $c(t,x)$ (see (3.2.18)), we get a term of e^{-2x} with a positive coefficient. There is no other term to compensate for this growth when $x \to -\infty$.

A related question is whether $\psi(x) = Ae^{kx} + Bx^r$ with positive values of $k, r \neq 1, A > 0, r = p/q, q$ odd and B chosen so that $\psi(x) \to +\infty$ as $|x| \to \infty$, would make a still better choice. We have $\psi_x = ake^{kx} + Brx^{r-1}$ and we again notice that $c(t,x) = \frac{b^2}{2}(\psi_x)^2$+other terms, where

$$(\psi_x)^2 = \begin{cases} A^2k^2e^{2kx} + o(e^{2kx}) \text{ for large } x > 0 \\ B^2r^2x^{2(r-1)} + o(x^{2(r-1)}) \text{ for large magnitudes of } x < 0 \end{cases} \tag{3.4.2}$$

We see that if $k > 1$, and for any value of r, $c(t,x) \to +\infty$ as $|x| \to +\infty$, while if $k < 1$, $\psi(x)$ loses control on the growth of other terms as $x \to +\infty$.

Finally, the most important complaint about the choice of this weight function is that it makes placing a bound on $y(t)$ necessary. Disregarding for the moment

that any computational system we use will do the same, it is worth asking if this could be avoided in principle. Looking again at the equation of $c(t,x)$ in (3.2.18), we see that the only control we can exert on this quantity in the effort to keep it non-positive, and to do so independently of $y(t)$, is by exploiting the term $(\psi_x)^2 - \psi_{xx}$. Suppose we could set

$$(\psi_x)^2 - \psi_{xx} = -q(x) \tag{3.4.3}$$

where $q(x) > 0$ and $e^{2x}, \psi_x e^x = o(q(x))$ as $x \to +\infty$. (Negative x is really no problem). If this were so, placing bounds on $y(t)$ would be unnecessary. Now (3.4.3) is a Ricatti equation in ψ_x, so we can make use of a standard technique and introduce a new function $w(x)$, with,

$$\psi_x(x) = -\frac{1}{w}\frac{dw}{dx} \tag{3.4.4}$$

to obtain,

$$\ddot{w} + q(x)w = 0 \tag{3.5.5}$$

We have at once that $\psi(x) = -\log(w(x))$, and, of course, these equations need be valid only for large values of positive x. This means that, since $\psi \in C^2(R)$, there exists an r such that, for all $x > r, w(x) > 0$ and since $\psi(x) \to +\infty$ as $x \to +\infty$, we need $w(x) \to 0$ as $x \to +\infty$. But does such a function exist? Since $q(x), w(x) > 0$, we need $\ddot{w} < 0$ in (3.4.5). Or, by the same token, if $\ddot{w}$ changes sign, so does w. This implies that we have, at least asymptotically, a positive function that tends to zero monotonically but whose second derivative is negative. But no function has such properties as the reader can easily verify by graphical experimentation.

These considerations suggest that $\psi(x) = Ae^x + Bx$ is an optimum or nearly optimum choice of a weight function.

Remark 4: The case of $b = 0$ in (3.2.7) is not readily incorporated into our general scheme, but we include this case for the sake of completeness.

We simply use a variant of the Kalman filter to obtain $\hat{z}(t) = E[z(t)|y_t]$ which

obeys,

$$d\hat{z}(t) = a\hat{z}(t)dt + K(t)[dy(t) - c\hat{z}(t)dt] \tag{3.4.6}$$

To derive the Kalman gain $K(t)$ we define $P(t) = E[(\hat{z} - z)^2 | y_t]$ and note that,

$$d(\hat{z} - z) = a(\hat{z} - z)dt + K(t)[-c(\hat{z} - z)dt + dv_t] \tag{3.4.7}$$

where we have used the definition of $y(t)$ in (3.2.7). We also have,

$$d(\hat{z} - z)^2 = (\hat{z} - z)d(\hat{z} - z) + (d(\hat{z} - z))^2 \tag{3.4.8}$$

which gives, upon taking expectation, and using $E[(dv_t)^2] = r^2 dt$,

$$\dot{P}(t) = (a - cK(t))P(t) + r^2 K^2(t) \tag{3.4.9}$$

To minimize $\dot{P}(t)$ we differentiate (3.4.9) by K to get,

$$-cP(t) + 2r^2 K(t) = 0 \tag{3.4.10}$$

or,

$$K(t) = \frac{cP(t)}{2r^2} \tag{3.4.11}$$

which yields,

$$\begin{aligned} \dot{P}(t) &= (a - \frac{c^2}{4r^2}P(t))P(t) \\ P(0) &= Var(z(0)) \end{aligned} \tag{3.4.12}$$

We see at once that if $a > 0, P(t) \to \frac{2r^2}{c^2}$ as $t \to +\infty$ while $a < 0$ makes $P(t) \to 0$.

5. Conclusion

We have shown that certain transformations can put the DMZ equation in the form of a parabolic PDE and that therefore classical techniques can be applied to it in order to obtain existence and uniqueness theorems, and we based our work on that of Baras, *et al* [1] in which certain technical problems were encountered and resolved in this chapter.

Our model problem was the bilinear filtering problem in one dimension, for which our methods were completely worked out in detail here for the first time. Extensions to higher dimensions are also possible, as will be shown in the next chapter.

The modifications of the methods of Baras, *et al* described here can be reapplied to their original work. These results allow for unbounded coefficients (of polynomial growth) on infinite domains. Asymptotic estimates were also obtained and will prove a useful prerequisite to numerical analysis, in the form of a tight uniform bound of our DMZ solution that tends to zero as $|x| \to \infty$. Combining this with an implicit finite difference scheme to be described in chapter 5, we will be able to numerically implement a linear systems solver (known as the Multigrid algorithm) on a compact set.

In effect, we have a general class of problems amenable to the numerical analysis to be described in this dissertation.

References for Chapter 3

[1] P. Besala, "Fundamental Solution and Cauchy Problem for a Parabolic System with Unbounded Coefficients," *J. Diff. Eqs.* vol. 33, pp. 26-38, 1979.

[2] Baras, Blankenship, and Hopkins, "Existence, Uniqueness, and Asymptotic Behavior of Solutions to a Class of Zakai Equations with Unbounded Coefficients", *IEEE Trans. Automatic Control*, vol. AC-28, No. 2, Feb. 1983.

[3] Davis and Marcus, "An Introduction to Nonlinear Filtering" in *Stochastic Systems: The Mathematics of Filtering and Identification* (NATO Advanced Study Institute Series). Dordrecht, The Netherlands: Reidel, 1981, pp. 53-75.

[4] E. Pardoux, "Stochastic Partial Differential Equations and Filtering of Diffusion Processes," *Stochastics*, vol. 3 pp. 127-167, 1979.

[5] W. H. Fleming and S. K. Mitter, "Optimal Control and Pathwise Nonlinear Filtering of Nondegenerate Diffusions," presented at the 20th IEEE Conf. Decision Contr., San Diego, CA, Dec. 1981.

[6] Davis, M. H. A., "A Pathwise Solution of the Equations of Zakai Filtering," Unpublished manuscript.

4. Some Qualitative Results on the Zakai Equation (Part II)

1. Introduction

This chapter is a continuation of chapter 3. Our goals is to extend the methods used on the model problem of bilinear filtering to some more general cases, especially those in higher dimensions.

We remind the reader that our goal is to demonstrate the wide class of Zakai equations we believe are available for numerical analysis. Our results, combined with filtering problems that are bounded or have less than polynomial growth, provide a broad range of realistic problems. Surely there would be no point in developing numerical methods for what could be a small (and probably unrealistic) class of problems. We believe we can show that this is not the case.

The theorems of the preceding chapter can be readily adapted to the equations described here but some minor modifications are needed. The asymptotic results can therefore yield bounded domains to be used in numerical analysis.

The first example is the scalar filtering problem with polynomial coefficients. This problem was also investigated by Hopkins in his dissertation, [3], and some of his results are cited in the paper by Baras, *et al*, [1]. However, our approach is similar to his, although he appears to be unaware of the subtle mistake found at the heart of the Baras *et al* paper, (for he includes the uncorrected bilinear problem in his dissertation.) Also, he was understandably content with "big $O(\cdot)$" arguments, while our numerical interests lead us to consider the computational requirements involved in finding the various constants that appear in our bounds.

The second example is a two-dimensional problem with linear state but nonlinear observations. This problem arose out of investigations conducted at the Naval Research Laboratory on the radar scattering behavior off of ships.

The third example is the general n-dimensional bilinear filtering problem. There exists a natural extension from the scalar case to the general formulas.

The reader should glean from this chapter the straightforward (albeit laborious) means by which asymptotic bounds can sometimes be derived for a reasonable class of problems.

Main Results: demonstration that methods of previous chapter can be extended to realistic problems of higher dimensions.

2. Scalar Filtering Problem with Polynomial Coefficients

The filtering problem with polynomial coefficients in R can be stated as follows. We have,

$$\begin{aligned} dx_t &= P(x_t)dt + Q(x_t)dw_t \\ dy_t &= R(x_t)dt + dv_t \end{aligned} \tag{4.2.1}$$

where $0 \le t \le T$, x_0 has probability density $p_0(x)$, w_t, v_t are standard Brownian motions. We also require that $Q(x) \ge \lambda > 0$ (e.g., $Q(x) = 1 + x^q, q$ even). Let the degrees of P, Q, R be p, q, r respectively and set,

$$\begin{aligned} P(x) &= \Sigma_{i=0}^{p} a_i x^i \\ Q(x) &= \Sigma_{i=0}^{q} b_i x^i \\ R(x) &= \Sigma_{i=0}^{r} c_i x^i \end{aligned} \tag{4.2.2}$$

The DMZ of (4.2.1) is

$$\begin{aligned} dU(t,x) = \Big[&\frac{1}{2}Q^2U_{xx} + (2QQ_x - P)U_x \\ &+ (Q_x^2 + QQ_{xx} - P_x - \frac{1}{2}R^2)U\Big]dt \\ &+ R(x)U\,dy_t \\ U(0,x) &= p_0(x) \end{aligned} \tag{4.2.3}$$

Again we convert to the robust version by defining,

$$V(t,x) = U(t,x)\exp[-R(x)y(t)]$$

to obtain,

$$\begin{aligned} V_t(t,x) &= A(x)V_{xx} + B(t,x)V_x + C(t,x)V \\ V(0,x) &= p_0(x), \quad 0 \le t \le T \end{aligned} \tag{4.2.4}$$

where,

$$\begin{aligned} A(x) &= Q^2/2 \\ B(t,x) &= 2QQ_x - P + Q^2 R_x y_t \\ C(t,x) &= (Q_x^2 + QQ_{xx} - P_x - R^2/2) + (2QQ_x - P)R_x y_t \\ &\quad \frac{Q^2}{2}[R_{xx}y_t + R_x^2 y_t^2] \end{aligned} \tag{4.2.5}$$

We proceed as before, and show existence, uniqueness, and asymptotic properties of solutions. Introducing the weight function, and transformed robust equation,

$$u(t,x) = V(t,x)\exp[\psi(x) - \gamma t]$$

where $\psi(x) = A_k x^k$, (the choice of which will be made clear. Also, certain changes will be needed if k is odd), we obtain,

$$u_t = a(x)u_{xx} + b(t,x)u_x + c(t,x)u \tag{4.2.6}$$

where, from (3.2.18), we have,

$$\begin{aligned} a(x) &= A(x) \\ b(t,x) &= -2A(x)\psi_x + B(t,x) \\ c(t,x) &= C(t,x) - B(t,x)\psi_x + A(x)[(\psi_x)^2 - \psi_{xx}] - \gamma \end{aligned} \tag{4.2.7}$$

As before, we must show that conditions of Besala are satisfied (cf. [2] and chap (3)). Since $Q(x) \geq \lambda > 0$, condition a) with $a(x) \geq \tilde{\lambda} > 0$ is met. Next we make $c(t,x) \leq 0$. We concentrate only on the degrees of the polynomials, so we set the degree of $Q^2(x) = (2q), \deg(PR_x y_t) = (p+r-1)$ etc. The $c(t,x)$ corresponds to a sum of polynomials whose degrees are in parentheses:

$$\begin{aligned} \deg(c(t,x)) = (2q-2)&+(2q-2) - (p-1) - (2r) \\ &+ (2q+r-1)y_t - (p+r-1)y_t \\ &+ (2q+r-2)y_t + (2q+2r-2)y_t^2 \\ &- (2q-2+k) + (p+k-1) \\ &- (2q+r-2+k)y_t \\ &+ (2q+2k-2) - (2q+k-2) \end{aligned} \tag{4.2.8}$$

We have included the y_t factor for reference. The reader can check with (4.2.5) to see which polynomials correspond to which in the above. From (4.2.8) we can see which terms will be dominant asymptotically. The reader can perhaps also see at this point why $\psi(x)$ must be a polynomial and, why for asymptotic purposes, only its leading term is relevant. The possible candidates are: $(2q+2r-2), (p+k-1), (2q+2k-2), (2q+r+k-2), (p+r-1)$, and,of course, the determination of which is the most dominant will depend on the relationship between p, q, r and k. We break this up into cases:$p\{>,=,<\}2q+r-1$.

Case 1: $p > 2q+r-1$ In this case we let $k = r, \psi(x) = A_r x^r$ Thus the dominate term will be:

$$a_p r(-c_r y_t + A_r) x^{p+r-1}$$

For large positive x, we have,

$$r a_p (A_r - c_r y_t) < 0 \tag{4.2.9}$$

Thus if $a_p > 0$ we have,

$$0 < A_r < |c_r y_t| \text{ or } |\frac{A_r}{c_r}| < |y_t| \tag{4.2.10}$$

where y_t is reset to never be zero and to be the same sign as c_r. Similarly, if $a_p < 0$,

$$|\frac{A_r}{c_r}| > |y_t| \tag{4.2.11}$$

Remember that resetting is only a theoretical tool in our analysis, as we will conduct our numerical work on the DMZ, which is driven by differences $\Delta y(t)$ in the observations. Thus it does not "see" resetting. We need it only as a formal tool in the proof of existence and uniqueness and in the determination of asymptotic bounds.

For $x < 0$ and $p+r-1$ not even, there seems to be no easy way to control the sign unless r is also odd, in which case, we construct $\psi(x)$ as follows,

$$\psi(x) = \begin{cases} A_r(+)x^r & A_r(+) > 0 \text{ large } x > 0 \\ A_r(-)x^r & A_r(-) < 0 \text{ large } |x|, x < 0 \\ \text{polynomial of degree } \leq r \text{ for } |x| \text{ small} \end{cases}$$

This construction is designed to insure $\psi(x)$ is smooth. We can set $A_r(+) = -A_r(-)$ and in fact we can take the even reflection of $\psi(x)$ from R^+ to R^-. The inequality signs are the reverse of what we have above for a_p positive and negative respectively.

To continue, we assume for purposes of illustration that $a_p > 0$ and $p + r - 1$ and r is even. Thus,

$$|\frac{A_r}{c_r}| < |y(t)|$$

We have shown that $c(t,x) < 0$ as $|x| \to \infty$, so we must now calculate γ so that $c(t,x) \leq 0$ for all $x \in R$. Let us assume, that we can make use of symbolic computation, which is now available, and easily applied to the analysis of polynomials. A reasonable but probably not the most efficient way of doing this is as follows:

1). Choose a value of A_r

2). Symbolically compute derivative of $c(t,x)$ on computer

3). Evaluate zeroes of this polynomial

4). Find maximum of $c(t,x)$

5). Choose γ to make $c(t,x) \leq 0$

6). Continue experimentation to yield nearly "nearly" optimal choices of $A_r$and γ.

The determination of what is nearly optimal will become clear in the attempt to obtain bounds of $U(t,x)$ which are as tight as possible. Our last condition is that $c(t,x) - b_x(t,x) - a_{xx}(x) \leq 0$. But this involves only the subtraction from $c(t,x)$ of polynomials whose leading terms are strictly less than those in $c(t,x)$. Thus the asymptotic behavior is the same as before, but a new choice of γ is necessary to ensure negativity. We proceed with symbolic computation as before to obtain a new value of γ that satisfies all conditions. Again we assume

$$p_0(x)\exp[\theta x^r] \leq M \quad \forall x \in R$$
$$M < \infty \quad 0 < \theta < -[A_r - c_r y(t)] \quad \forall t \in [0,T] \tag{4.2.12}$$

Choose $\tilde{A}_r, \epsilon > 0$, such that,

$$\tilde{A}_r + \epsilon < |c_r y(t)|$$

Thus $\tilde{A}_r$ and ϵ can each be one-half of previously chosen A_r. As we did in former chapter, set

$$\begin{aligned} \tilde{\psi}(x) &= \psi(x) + \epsilon x^r \\ \tilde{u}(t,x) &= u(t,x)\exp[\epsilon x^r] \end{aligned} \tag{4.2.13}$$

Then $\tilde{u}$ also satisfies an equation of the form (4.2.6) whose coefficients are formed by replacing ψ with $\tilde{\psi}$ in the definitions of coefficients in (4.2.7). Thus $\tilde{u}$ will have a solution by Besala and, since its initial condition is bounded, $\tilde{u}$ will be bounded as well. Also, since $\epsilon > 0$, $u \to 0$ as $|x| \to 0$. We therefore have uniqueness in the class of functions,

$$U(t,x)\exp[A_r x^r - \gamma t - R(x)y(t)] \to 0 \tag{4.2.14}$$

The most important consideration for computational purposes, is the comparison theorem. Consider, as in Theorem 3 of chapter 3,

$$\begin{aligned} w_1(t,x) &= u^k(t,x) \\ w_2(t,x) &= \exp[\lambda t - \mu\phi(x)] \end{aligned} \tag{4.2.15}$$

where $u^k(t,x)$ is $u(t,x)$ a resetting interval, and $\phi(x) = x^r$. Set $T = \frac{\partial}{\partial t} - \mathcal{L}$ where $\mathcal{L}$ is the operator in the transformed robust equation. We must have $-Tw_2 \leq 0$ Working out the necessary algebra gives, as dominant terms,

$$-P(x)rx^{r-1}\mu, -P(x)R_x(x)y(t), P(x)\psi_x(x)$$

and we need as before,

$$a_p r[-\mu - c_r y(t) + A_r]x^{p+r-1} < 0 \tag{4.2.16}$$

we already have,

$$\frac{A_r}{|c_r|} \leq |y(t)|$$

by resetting. Thus we need,

$$\frac{A_r - \mu}{|c_r|} < |y(t)|$$

Thus set $\mu = A_r/2 > 0$, and the inequality will be valid.

From this we can calculate the value of λ needed to make the polynomial $-Tw_2 \leq 0$, just as we did before with symbolic computation.

We have evaluated all of our parameters and we conclude that,

$$U(t,x) \leq \exp[(\lambda+\gamma)T]\exp[(A_r-\mu)x^r - R(x)y(t)] \tag{4.2.17}$$

Note that our choice of γ depends in large part on our choice of A_r, so we should try to optimize this equation as much as possible.

Now we will require by resetting,

$$\min|y(t)| = \frac{A_r-\mu}{|c_r|} + 1 \tag{4.2.18}$$

and we find,

$$\begin{aligned}\max[(A_r-\mu)x^r - R(x)y(t)] \\ = -\tilde{R}(x) = -\left(\sigma_{i=0}^{r-1} c_i x^i + |c_r|x^r\right)\end{aligned} \tag{4.2.19}$$

Thus,

$$U(t,x) \leq \exp[(\gamma+\lambda)T]\exp[-\tilde{R}(x)] \tag{4.2.20}$$

Case 2: $p = 2q + r - 1$

We see that $p + k - 1 = 2q + r + k - 2$ for all k in this case, but if $k > r$, then $Q^2(\psi_x)^2$ will be dominant and $c(t,x)$ will tend to $+\infty$. Likewise if $k < r$, then $Q^2R_x^2y^2(t)$ will be dominant and we have the same problem. So we must set $k = r$ to get as the dominant term,

$$\frac{Q^2}{2}(\psi_x)^2 - \psi_x Q^2 R_x y(t) + P(x)\psi_x + \frac{Q^2}{2}R_x^2 y^2(t) \tag{4.2.21}$$

and this yields for large x,

$$\frac{1}{2}r\left(b_q^2 r A_r^2 - 2A_r b_q^2 c_r r y(t) + 2a_p A_r + b_q^2 c_r^2 r y^2(t)\right)x^{p+r-1} \tag{4.2.22}$$

Thus, we must choose A_r to be between the roots,

$$A_r(\pm) = \frac{-(a_p - b_q^2 c_r r y(t)) \pm \sqrt{a_p^2 - 2a_p c_r r b_q^2 y(t)}}{b_q^2 r} \tag{4.2.23}$$

For these roots to be real, we immediately see that we need as a resetting requirement,

$$\frac{|a_p|}{2|c_r| r b_q^2} > |y(t)| \tag{4.2.24}$$

Recall that for $x > 0$ we must have $A_r > 0$. Can we do this? If so, then the greatest root in (4.2.23) must be of the form,

$$-(r_1 - r_2) + \sqrt{r_1^2 - 2r_1 r_2} > 0 \tag{4.2.25}$$

where $r_1 = a_p, r_2 = b_q^2 c_r r y(t)$ and the above must be true for all $t \in [0, T]$, with the help of resetting, of course. But (4.2.25) can be re-written as,

$$-(r_1 - r_2) + \sqrt{(r_1 - r_2)^2 - r_2^2} > 0 \tag{4.2.26}$$

and we see that we cannot have $r_1 - r_2 > 0$ for this implies,

$$\begin{aligned} \left(\sqrt{(r_1 - r_2)^2 - r_2^2}\right)^2 &> (r_1 - r_2)^2 \\ -(r_2)^2 &> 0 \end{aligned} \tag{4.2.27}$$

which is impossible. But $a_p > 0$ implies $r_1(r_1 - 2r_2) > 0$ or $r_1 > r_2$. Thus we cannot have $a_p > 0$. We continue exactly as we did in case 1 with the assumption (4.2.12) and $a_p < 0$. We calculate A_r, γ noting that the other condition, $c(t,x) - b_x(t,x) - a_{xx}(t,x) \leq 0$ yields a new value for γ, (but no terms higher than x^{p+r-1}). Uniqueness and the comparison theorem are proved similarly.

Case 3: $p < 2q + r - 1$

Again we need $k = r$ to control the sign, for if $k > r$, $Q^2(\psi_x)^2$ will dominate, and if $k < r$, $Q^2 R_x^2 y^2(t)$ will dominate, and this will once more produce undesirable asymptotic behavior. Thus set $k = r$ to get as the dominant term,

$$\frac{Q^2}{2} R_x^2 y^2(t) + \frac{Q^2}{2}\psi_x^2 - Q^2 R_x \psi_x y(t) \tag{4.2.28}$$

The leading term for large x is, when made negative,

$$b_q^2 r^2 (c_r y(t) + A_r)^2 < 0 \tag{4.2.29}$$

which is clearly impossible. We conclude that we cannot force $c(t,x) \leq 0$, and that our method is unworkable in this case.

3. Linear State in R^2 with Nonlinear Observations

We consider the equations,

$$\begin{aligned} d\bar{x}_t &= A\bar{x}_t dt + B dw_t \\ dy_t &= c|\bar{x}_t| dt + dv_t \end{aligned} \tag{4.3.1}$$

where

$$A = \begin{pmatrix} -\alpha & 0 \\ 0 & -\alpha \end{pmatrix} \qquad B = \begin{pmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{pmatrix} \tag{4.3.2}$$

and where $\alpha > 0$, $\beta_1 \neq \beta_2$, $w_t, v_t \in R$, are independent and standard Brownian motions, $|\bar{x}| = \sqrt{x_1^2 + x_2^2}$, and c is real. The initial probability density of $\bar{x}_0$ is $p_0(\bar{x})$.

The above model is relevant in the analysis of ship radar tracking, as done at the Naval Research Laboratory.

The DMZ associated with (3.1) is,

$$\begin{aligned} dU(t,x) = {} & (\mathcal{L}^* - c^2/2|\bar{x}|^2)U dt + c|\bar{x}_t| dy_t \\ & \Big[\frac{\beta_1^2}{2}\frac{\partial^2 U}{\partial x_1^2} + \frac{\beta_2^2}{2}\frac{\partial^2 U}{\partial x_2^2} + \frac{\partial}{\partial x_1}(\alpha x_1 U) + \frac{\partial}{\partial x_2}(\alpha x_2 U) - \frac{c^2}{2}(x_1^2 + x_2^2)\Big] dt \\ & + c|\bar{x}_t| U dy_t \end{aligned} \tag{4.3.3}$$

We have used the Stratonovich version of the DMZ where $\mathcal{L}^*$ is the adjoint of the infinitesimal generator of (4.3.3). The robust transformation of (4.3.3) is as follows, let $V(t,x) = \exp[-c|\bar{x}|y(t)]$, then,

$$\begin{aligned} V_t = {} & \frac{\beta_1^2}{2} V_{x_1 x_1} + \frac{\beta_2^2}{2} V_{x_2 x_2} \\ & + \Big[\alpha x_1 + \beta_1^2 c \frac{x_1}{|\bar{x}|} y(t)\Big] V_{x_1} \\ & + \Big[\alpha x_2 + \beta_2^2 c \frac{x_2}{|\bar{x}|} y(t)\Big] V_{x_2} \\ & + \Big[2\alpha + \alpha|\bar{x}| y(t) + c^2 y^2(t) - \frac{c^2}{2}(x_1^2 x_2^2) \\ & + c\frac{\beta_1^2}{2}\frac{x_2^2}{|\bar{x}|^3} y(t) + c\frac{\beta_2^2}{2}\frac{x_1^2}{|\bar{x}|^3} y(t)\Big] V \end{aligned} \tag{4.3.4}$$

Note that the above is in the form:

$$V_t = A_1(\bar{x}) V_{x_1 x_1} + A_2(\bar{x}) V_{x_2 x_2} + B_1(t,\bar{x}) V_{x_1} + B_2(t,\bar{x}) V_{x_2} + C(t,\bar{x}) V$$

We have also made use of the fact that,

$$\left(\frac{x_i}{|\bar{x}|}\right)_{x_i} = \frac{x_j^2}{|\bar{x}|^3} \quad i \neq j$$

From this we can construct the transformed robust equation with,

$$u(t,\bar{x}) = V(t,\bar{x})\exp[\psi(\bar{x}) - \gamma t]$$

where $\psi(\bar{x}) = d_1x_1^2 + d_2x_2^2$. Here, $d_i > 0$, so $\psi(\bar{x}) \to +\infty$ as $|\bar{x}| \to +\infty$ We have,

$$\begin{aligned} u_t = {} & \frac{\beta_1^2}{2}u_{x_1x_1} + \frac{\beta_2^2}{2}u_{x_2x_2} \\ & + u_{x_1}[B_1(t,\bar{x}) - \beta_1^2\psi_{x_1}] \\ & + u_{x_2}[B_2(t,\bar{x}) - \beta_2^2\psi_{x_2}] \\ & + u[C(t,\bar{x}) - \gamma \\ & + \frac{\beta_1^2}{2}(\psi_{x_1}^2 - \psi_{x_1x_1}) + \frac{\beta_2^2}{2}(\psi_{x_2}^2 - \psi_{x_2x_2}) \\ & - \psi_{x_1}B_1(t,\bar{x}) - \psi_{x_2}B_2(t,\bar{x})] \end{aligned} \tag{4.3.5}$$

Again we notice that (4.3.5) has the form:

$$u_t = a_1(\bar{x})u_{x_1x_1} + a_2(\bar{x})u_{x_2x_2} + b_1(t,\bar{x})u_{x_1} + b_2(t,\bar{x})u_{x_2} + c(t,\bar{x})u$$

We see that,

$$\begin{aligned} c(t,\bar{x}) = {} & 2\alpha + \alpha|\bar{x}|y(t) - \frac{c^2}{2}(x_1^2x_2^2) \\ & + c\beta_1^2\frac{x_2^2}{|\bar{x}|^3}y(t) + c\beta_2^2\frac{x_1^2}{|\bar{x}|^3}y(t) + c^2y^2(t) \\ & + \beta_1^2(2d_1^2x_1^2 - d_1) + \beta_2^2(2d_2^2x_2^2 - d_2) \\ & - 2d_1x_1\left[\alpha x_1 + \beta_1^2\frac{x_1}{|\bar{x}|}y(t)\right] \\ & - 2d_2x_2\left[\alpha x_2 + \beta_2^2\frac{x_2}{|\bar{x}|}y(t)\right] - \gamma \end{aligned} \tag{4.3.6}$$

The dominant term for large $|\bar{x}|$ is seen to be,

$$2d_1\left(d_1\beta_1^2 - \alpha - c^2/2\right)x_1^2 + 2d_2\left(d_2\beta_2^2 - \alpha - c^2/2\right)x_2^2 \tag{4.3.7}$$

Thus to make this term tend to negative infinity we set,

$$d_1 = \frac{\alpha}{2\beta_1^2}, \quad d_2 = \frac{\alpha}{2\beta_2^2} \tag{4.3.8}$$

Note that $\alpha > 0$ so $d_i > 0$ as required, and we are guaranteed that, $c(t, \bar{x}) \to -\infty$ as $|\bar{x}| \to +\infty$.

The Besala theorem [2] requires only that $c(t, \bar{x})$ be negative; smoothness conditions are not necessary. However, the term $\frac{x_i^2}{|\bar{x}|^3}$ is unbounded as $x_j \to 0$ for $i \neq j$. (In fact, this term will also appear in $b_{j,x_j}(t, \bar{x})$ later on). For reasons which will become clear, choose $\sigma > 0$, and let our domain be $D = R^2 \cap B^c(0, \sigma)$ where $B(0, \sigma)$ is the open disc of radius σ. The choice of σ will be determined later.

Finding the maximum of $c(t, \bar{x})$ on D is not easily performed analytically. Thus the following method, to be used on a computer, will suffice. Reset $y(t)$ so that $sgn(y(t)) = -sgn(c)$.* So if $c > 0$, as we will assume from now on, we set $-M_1^2 < y(t) < -M_2^2$. Now replace $y(t)$ in (4.3.6) with $-M_2$ to get $\tilde{c}(t, \bar{x})$. Thus $\tilde{c}(t, \bar{x}) \geq c(t, \bar{x})$. Using a computer and the values of d_1, d_2 in (4.3.6), find the maximum of $\tilde{c}(t, \bar{x})$ on D, and then choose γ so that $\tilde{c}(t, \bar{x}) \leq 0, \forall x \in D$. Note how our resetting criterion is reflected in our determination of γ.

Now we see that if σ is small enough, and since the coefficient of $\frac{x_i^2}{|\bar{x}|^3}$ is negative, by resetting, the choice of γ is unaffected as σ gets arbitrarily small. We shall comment on this fact in the sequel.

Now Besala's Theorem in two dimensions [2] also proves that the existence of a solution of (4.3.3) corresponds to showing that, in addition to the above,

$$c(t, x) - b_{1,x_1} - b_{2,x_2} - a_{1,x_1x_1} - a_{2,x_2x_2} \leq 0 \tag{4.3.9}$$

We know that $a_{i,x_ix_i} = 0$, for $i = 1, 2$, while,

$$b_{i,x_i} = \alpha + \beta_i^2 c \frac{x_j^2}{|\bar{x}|^3} y(t) - 2\beta_i^2 d_i \tag{4.3.10}$$

---

* The function $sgn(z) = \begin{cases} 1 & z > 0 \\ 0 & z = 0 \\ -1 & z < 0 \end{cases}$

where $i \neq j$. We see that when we perform the addition in (4.3.6) the troublesome terms involving $\frac{x_i^2}{|\bar{x}|^3}$ precisely cancel. Also note that due to our choice of d_i, $\alpha - 2\beta_i^2 d_i = 0$. A new choice of γ need not be selected as we are subtracting only positive terms.

Finally note that $\beta_i^2/2 > 0$ also meets Besala's requirements.

To show uniqueness we choose $\epsilon_1, \epsilon_2 > 0$ such that,

$$d_i + \epsilon_i < \frac{\alpha}{\beta_i^2} \tag{4.3.11}$$

We also assume that,

$$\begin{gathered} p_0(\bar{x}) \exp[\theta_1 x_1^2 + \theta_2 x_2^2] \leq M < +\infty \\ \text{for } \theta_i < \frac{\alpha}{\beta_i^2} \end{gathered} \tag{4.3.12}$$

Thus, as we did in chapter 3, we let,

$$\begin{aligned} \tilde{\psi}(\bar{x}) &= \psi(\bar{x}) + \epsilon_1 x_1^2 + \epsilon_2 x_2^2 \\ \tilde{u}(t, \bar{x}) &= u(t, \bar{x}) \exp[\epsilon_1 x_1^2 + \epsilon_2 x_2^2] \end{aligned} \tag{4.3.13}$$

where $\tilde{u}(t, \bar{x})$ is obtained by replacing $\psi(\bar{x})$ by $\tilde{\psi}(\bar{x})$ in the definition of the coefficients of the PDE of $u(t, \bar{x})$. Thus $\tilde{u}(t, \bar{x})$ solves a new PDE and will also have a solution due to Besala's theorem, [2]. Because the initial condition is bounded, $\tilde{u}(t, \bar{x})$ is bounded. And because $\epsilon_i > 0, u(t, \bar{x}) \to 0$ as $|\bar{x}| \to \infty$. Application of the maximum principle as in chapter 3 provides a unique solution in the class of functions which obey, for $\tilde{\theta} < \theta_i$,

$$u(t, \bar{x}) \exp[\tilde{\theta}_1 x_1^2 + \tilde{\theta}_2 x_2^2 - c|\bar{x}|y(t)] \to 0$$

as $|\bar{x}| \to \infty$

Next comes the comparison theorem. Let

$$\begin{aligned} w_1(t, \bar{x}) &= u(t, \bar{x}) \\ w_2(t, \bar{x}) &= \exp[\lambda t - \mu \phi(\bar{x})] \end{aligned} \tag{4.3.14}$$

where $\lambda, \mu > 0$ are to chosen and $\phi(\bar{x}) = x_1^2 + x_2^2$. Define $T = \frac{\partial}{\partial t} - \mathcal{L}$ to be a parabolic operator where $\mathcal{L}$ is the operator associated with $u_t = \mathcal{L}u$ in (4.3.5). We need to show that $-Tw_2 \leq 0$. (Note that $Tw_1 = 0$; consult chapter 3 for more details about the comparison theorem). We must show that,

$$\begin{aligned} &\beta_1^2[2\mu x_1^2 - 1]\mu + \beta_2^2[2\mu x_2^2 - 1]\mu \\ &- 2\mu x_1\Big[\alpha x_1 + \beta_1^2 c\frac{x_1}{|\bar{x}|}y(t) - 2\beta_1^2 d_1 x_1\Big] \\ &- 2\mu x_2\Big[\alpha x_2 + \beta_2^2 c\frac{x_2}{|\bar{x}|}y(t) - 2\beta_2^2 d_2 x_2\Big] \\ &+ c(t,\bar{x}) - \lambda \leq 0 \end{aligned} \tag{4.3.15}$$

The leading terms for $|\bar{x}| \to +\infty$ are, after algebraic simplification,

$$\Sigma_{i=1}^2\Big(2\mu^2\beta_i^2 - \frac{\alpha^2}{4\beta_i^2}\Big)x_i^2 \tag{4.3.16}$$

where we have used, $d_i = \alpha/2\beta_i^2$. Thus we can set,

$$\mu = \frac{1}{4}\min_{i=1,2}[d_i] \tag{4.3.17}$$

and be assured that the leading terms in (4.3.15) tend to $-\infty$ as $|\bar{x}| \to \infty$. From this λ can be chosen to make $Tw_2 \leq 0$ on D. Again, if σ gets arbitrarily small, no change in λ is required due to negative coefficient, brought on by resetting, in front of $\frac{x_i^2}{|\bar{x}|^3}$.

We can conclude that, with the resetting requirement, $-M_1^2 \leq y(t) \leq M_2^2$

$$U(t,\bar{x}) \leq M\exp[(\lambda+\gamma)T]\exp[(\mu - d_1)x_1^2 + (\mu - d_2)x_2^2 - |c||\bar{x}|M_1^2] \tag{4.3.18}$$

Note that since $\mu - d_i < 0$, our bound tends to 0 as $|\bar{x}| \to +\infty$ very fast.

The only problem we have is near the origin, but in the sort of problems this model is designed for, the probability of finding the state $\bar{x}_t$ near the origin is very small, so we can assume that $p_0(\bar{x}) \to 0$ as $|\bar{x}| \to 0$ and so our initial density plays the role of creating the disc $B(0,\sigma)$ for us, thus avoiding any numerical problems in this region.

4. The n-Dimensional Bilinear Filtering Problem

By now our methods should seem fairly adaptive. We apply them now to the n-dimensional bilinear filtering problem. We have

$$\begin{aligned} d\bar{x}_t &= A\bar{x}dt + B\bar{x}dw_t \\ dy_t &= C\bar{x}_t dt + d\bar{v}_t \end{aligned} \tag{4.4.1}$$

where $\bar{x}_t, \bar{y}_t \in R^n$ and $dw_t \in R, d\bar{v}_t \in R^n$ are independent. A, B, C are all $n\tilde{n}$ matrices. Also, $\bar{x}_0 \sim p_0(\bar{x}) : R^n \to R$, and $y(0) = 0$, with $0 \leq t \leq T < \infty$.

The first requirement we make is that $B^T B$ is diagonal, for if this were not the case, we could use the transformation $\bar{x} \to P\bar{x}$ to obtain,

$$\begin{aligned} d\bar{x}_t &= P^{-1}AP\bar{x}dt + P^{-1}BPdw_t \\ d\bar{y}_t &= CP\bar{x}dt + d\bar{v}_t \end{aligned} \tag{4.4.2}$$

Thus, when we form the DMZ to yield

$$(P^{-1}BP)^T(P^{-1}BP) = P^T B^T BP$$

if we also assume that $P^{-1} = P^T$. Thus, as $B^T B$ is symmetric, with $det(B) \neq 0$, choose P so that

$$P^T B^T BP = D \tag{4.4.3}$$

where $D = [d_{ii}]$ is a diagonal matrix. Henceforth, we will assume that this transformation has already been done.

Now the DMZ of (4.4.1) is

$$\begin{aligned} dU(t,\bar{x}) = &\sum_i d_{ii}/2x_i^2 U_{x_i x_i} \\ &+ \sum_i \beta^2[2d_{ii}x_i^2 - \sum_j a_{ij}x_j\beta^2]U_{x_i} \\ &+ \Big[\sum_i \beta^2[d_{ii} - \sum_j a_{ij}\beta^2] - \frac{1}{2}\bar{x}^T C^T C\bar{x}\Big]U \\ &+ (C\bar{x}, \bar{y}_t)U \end{aligned} \tag{4.4.4}$$

The expression $(C\bar{x}, \bar{y}(t))$ is the scalar dot product. We observe that if $n = 1$, then $d_{11} = b^2$, and we get the same result as before. Now, to remove the degeneracy, we introduce a change of coordinants,

$$\frac{\psi(\cdot)}{\psi x_i} = e^{-z_i}\frac{\psi(\cdot)}{\psi z_i} \tag{4.4.5}$$

which implies

$$W(t, \bar{z}) = U(t, e^{z_1}, e^{z_2}, ..., e^{z_n})$$

with

$$\begin{aligned} U_{x_i} &= e^{-z_i}U_{z_i} \\ U_{x_i x_i} &= -e^{-2z_i}U_{z_i} + e^{-2z_i}U_{z_i z_i} \end{aligned} \tag{4.4.6}$$

The reader should see that the new coordinants effectively remove the degeneracy. Our new DMZ is thus

$$\begin{aligned} dW = &\sum_i d_{ii}/2[W_{z_i z_i} - W_{z_i}] \\ &+ \sum_i [2d_{ii} - \sum_j a_{ij}]W_{z_i} \\ &+ \sum_i [d_{ii} - \sum_j a_{ij} - \frac{1}{2}\epsilon^T C^T C\epsilon]W(t, z) \end{aligned} \tag{4.4.7}$$

where

$$\epsilon = (e^{z_1}, e^{z_2}, ..., e^{z_n})$$

The robust transformation, which is of the form:

$$V(t, \bar{z}) = W(t, \bar{z}) \exp[-(C\epsilon, \bar{y}(t))] \tag{4.4.8}$$

is then followed by the transformation:

$$u(t, z) = V(t, z) \exp[\psi(\bar{z}) - \gamma t] \tag{4.4.9}$$

where $\psi(\bar{z})$ is the weight function, and as yet to be determined. This yields,

$$\sum_i a(\bar{z})u_{z_i z_i} + \sum_i b_i(t, \bar{z})u_{z_i} + c(t, \bar{z})u \tag{4.4.10}$$

We are now in a position of applying Besala's Theorem, as described in the last chapter. The coefficients of (4.4.10) are

$$\begin{aligned}
a_i(\bar{z}) =& d_{ii}/2 \\
b_i(t,\bar{z}) =& -d_{ii}\psi_{z_i} - (\sum_j a_{ij} - \frac{3}{2}d_{ii} + d_{ii}(C\epsilon, \bar{y}(t)) \\
c(t,\bar{z}) =& \sum_i d_{ii}/2\beta^2[(\psi_{z_i})^2 - \psi_{z_i z_i}\beta^2] \\
& - \sum_i \psi_{z_i}\beta^2[d_{ii}/2(C\epsilon, \bar{y}(t)) - (\sum_j a_{ij} - \frac{3}{2}d_{ii})\beta^2] \\
& + \sum_i \beta^2[(C\epsilon, \bar{y}(t))^2 + (C\epsilon, \bar{y}(t))\beta^2] \\
& - (C\epsilon, \bar{y}(t))(\sum_j a_{ij} - \frac{3}{2}d_{ii}) \\
& + \sum_i (d_{ii}/2 - \sum_j a_{ij}) - |C\epsilon|/2
\end{aligned} \tag{4.4.11}$$

As anticipated, we need

$$\psi(\bar{z}) = \sum_i A_i e^{z_i} + B_i z_i \tag{4.4.12}$$

The rest of the details follow through, largely due to the decoupling allowed for by $B^T B$ being diagonal. In the interests of brevity, for the results can be derived in a straightforward but rather tedious manner, we simply state that

$$A_i \in \beta^2((C\bar{y})_i - \left|\frac{c_{ii}}{\sqrt{d_{ii}}}\right|, (C\bar{y}(t))_i + \left|\frac{c_{ii}}{\sqrt{d_{ii}}}\right|\beta^2) \tag{4.4.13}$$

The reader will recognize (4.4.13) as the natural generalization of

$$A \in (cy - |\frac{c}{b}|, cy + |\frac{c}{b}|) \tag{4.4.14}$$

found in the previous chapter.

It follows that we must find $\epsilon_j > 0$ such that

$$|y_j| < \epsilon_j$$

such that the coefficients in front of *all* the e^{2z_i} terms will be negative. Once again, we note that $d\bar{y}(t)$ does not detect the resetting that must be done to keep $y(t)$ to

with a certain magnitude, but which preserves the dynamics. Also, $A_i > 0$ so that $\psi(\bar{z}) \to +\infty$ as $|\bar{z}| \to \infty$. Thus, we have

$$0 < A_i < \sum_j c_{ij} y_j + \left| \frac{c_{ii}}{\sqrt{d_{ii}}} \right|$$

This will imply that we should set

$$A_i = \left| \frac{\kappa_{ii}}{2\sqrt{d_{ii}}} \right| \tag{4.4.15}$$

Again, for the sake of brevity, we state our other results, which can be naturally, albeit laboriously, derived. Existence and uniqueness offer no surprises. For the various parameters, we have,

$$\begin{aligned} B_i &\leq \min\left[0, \frac{2\delta_{ii} - \sum_j a_{ij}}{d_{ii}}\right] \\ \mu_i &\leq \frac{|c_{ii}|}{12\sqrt{d_{ii}}} \\ M &= \exp \beta^2 [\sum_i (\mu_i^2 d_{ii}/2 + d_{ii}\mu_i B_i(B_i/2 - 1) \\ &\quad + (\mu_i - B_i)(3d_{ii}/2 - \sum_j a_{ij}) + d_{ii} - \sum_j a_{ij}) T\beta^2] \end{aligned} \tag{4.4.16}$$

We also have,

$$K_i = \min \beta^2 [-(B_i + \mu_i), \frac{|c_{ii}|}{\sqrt{d_{ii}}} \beta^2] \tag{4.4.17}$$

where $K_i < 0$. All of which has a bound of

$$\begin{aligned} U(t, \bar{x}) &\leq M \Pi_{i=1}^n \left(x_i^{K_i} \exp[-K_i x_i] \right) \\ &\forall (t, \bar{x}) \in [0, T] \times R_+^n \end{aligned} \tag{4.4.18}$$

5. Conclusion

This chapter was intended as a demonstration that the methods of chapter 3 could be expanded to include a broader range of problems, some of them higher dimensions. This is clearly in order as our interests are in the field of n-dimensional signal processing.

We also argue that our methods extend an already wide class of problems that are suitable for numerical analysis. These include the cases of bounded or sublinear coefficient growth.

However, we might remark that in practice, given such problems we might be inclined to use numerical experimentation to determine the finite domain, than to resort to theoretical bounds that may prove to be too difficult to obtain.

With this part of our project complete, throughout the remainder of this research effort, we will employ the operating assumption that such finite domains have been found with an acceptable degree of accuracy.

References for Chapter 4

[1] Baras, Blankenship, and Hopkins, "Existence, Uniqueness, and Asymptotic Behavior of Solutions to a Class of Zakai Equations with Unbounded Coefficients", *IEEE Trans. Automatic Control*, vol. AC-28, No. 2, Feb. 1983.

[2] P. Besala, "Fundamental Solution and Cauchy Problem for a Parabolic System with Unbounded Coefficients," *J. Diff. Eqs.* vol. 33, pp. 26-38, 1979.

[3] Hopkins, W. E., *Nonlinear Filters of Nondegenerate Diffusions with Unbounded Coefficients,* Ph.D dissertation, Univ. of Maryland, 1982.

5. Finite Difference Schemes for the Zakai Equation

1. Introduction

In this chapter we discuss the notions of weak convergence and the development of a finite difference scheme especially designed for PDE generated by stochastic processes, which is indeed the case for the Zakai equation.

The finite difference scheme we offer for the n-dimensional Zakai equation lends itself very well to probabilistic interpretation. This is an implicit scheme that provides stability for a greater range Δt and Δx values than does the explicit method. In fact, the discrete solution of the linear system converges weakly to the true solution as $\Delta x, \Delta t \to 0$ *independently of each other*. It is also very stable.

We also show that the inherent probabilistic interpretation of the linear system allows us to delineate the structure of the associated matrix, at least when the PDE is discretized over a rectangular domain. This will prove useful in the numerical analysis of the system and the making of our choice of relaxation schemes within the framework of the Multigrid algorithm.

Main Results: identification of an implicit finite difference scheme. Derivation of properties of the associated matrix of the linear system relevant to its numerical analysis.

2. Finite Difference Methods for PDE's and their Probabilistic Interpretation

We provide a brief review of weak convergence. More details can be found in Kushner, [1].

The basic idea is that when we discretize the Zakai equation (as a prelude to numerical analysis), the resulting linear system can be viewed as a transition probability density for a Markov chain that corresponds to the discretized state variable we are trying to conditionally estimate. Thus, the statistical properties predicted by the *finite* system converge to the true statistical properties of our continuous system.

Let us now show how the probabilistic interpretation of differential equations can yield insights into the numerical analysis performed on them.

The following example is from Kushner [1]. Consider the equation,

$$a(x)V_{xx}(x) + f(x)V_x(x) + k(x) = 0$$

$$V(0) = \alpha, \quad V(1) = \beta \tag{5.2.1}$$

$$a(x) > \epsilon > 0,\ a, f \in C^1((0,1)),\ k(x),\ \text{bounded}$$

We associate with the above equation the stochastic differential,

$$dX = f(x)dt + (2a(X))^{\frac{1}{2}}dw$$

$$X(0) = x \tag{5.2.2}$$

which has a unique solution as f and a are Lipschitz.

Let $\tau = \inf[t : X(t) \notin (0,1)]$, which is the exit time of $X(t)$ for $(0,1)$. Because $a(x) > \epsilon > 0$, it can be shown that $E_x(\tau) < \infty$, for all $x \in (0,1)$, where x is the initial condition of (5.2.2), in much the same way as in section one.

Thus the solution of (5.2.1) is,

$$V(x) = E_x[\int_0^\tau k(X(s))\,ds + \alpha(P_x(X(\tau) = 0)) + \beta(P_x(X(\tau) = 1))], \quad x \in (0,1) \tag{5.2.3}$$

We will now provide a special finite difference approximation to (5.3.1) whose solutions will tend to its true solution. Define $1/h$ to be an integer and replace

$$V_{xx} \to [V(x+h) - 2V(x) - V(x-h)]/h^2 \tag{5.2.4}$$

$$V_x(x) \to \begin{cases} [V(x+h)-V(x)]/h \text{ if } f(x) \geq 0 \\ [V(x)-V(x-h)]/h \text{ if } f(x) < 0 \end{cases} \tag{5.2.5}$$

The reader will note that we used forward and backward approximations of V_x when $f(x)$ is respectively positive and negative. This is normally not done but there is a good reason for doing so. Now for each h, define,

$$\begin{aligned} Q_h(x) &= 2a(x) + h|f(x)| \\ p^h(x, x \pm h) &= [a(x) + hf^{\pm}(x)]/Q_h(x) \\ \Delta t^h(x) &= h^2/Q_h(x), \quad x = 0, \pm h... \end{aligned} \tag{5.2.6}$$

Let $p^h(x,y) = 0$ when $y \neq x \pm h$, x being a multiple of h. Now put (5.2.4)-(5.2.5) into (5.2.1), collect terms, divide by $Q_h(x)$ to get,

$$\begin{aligned} &V^h(x) = V^h(x+h)p^h(x,x+h) + V^h(x-h)p^h(x,x-h) + k(x)\Delta t^h(x) \\ &\qquad x = h, ...1-h \\ &V^h(0) = \alpha, \quad V^h(1) = \beta \end{aligned} \tag{5.2.7}$$

This equation has an interesting probabilistic interpretation. The $p^h(x,y)$ are ≥ 0, and sum over y to unity for each x. We conclude that $p^h(x,y)$ is a transition probability function for Markov chain on the state space $0, \pm h, ...$ Let $\{\varsigma_n^h, n = 0, 1, 2...\}$ be the random variables of this chain, and define N_h to be the exit time of the chain from $(0,1)$. Thus $N_h \equiv \min\{n : \varsigma_n^h \notin (0,1)\}$. Using this, equation (5.2.3) can be written,

$$V^h(x) = E_x[\sum_{i=0}^{N_h-1} k(\varsigma_i^h)\Delta t_i^h + b(\varsigma_{N_h}^h)] \tag{5.2.8}$$

where we use $\Delta t_i^h = \Delta t^h(\varsigma_i^h)$. We see at once the similarity between (5.2.3) and (5.2.8). But what we need to know is under what conditions and in what sense does V^h tend to V, the solution of (5.2.1).

To answer this, we note that

$$\begin{aligned} &E[\varsigma_{n+1}^h - \varsigma_n^h | \varsigma_n^h = x] f(x)\Delta t^h(x) \\ &cov[\varsigma_{n+1}^h - \varsigma_n^h | \varsigma_n^h = x] = 2a(x)\Delta t^h(x) \\ &\qquad + \Delta t^h(x)(h|f(x)| - f^2(x)\Delta t^h(x)) \\ &\qquad = 2a(x)\Delta t^h(x) + o(\Delta t^h(x)) \end{aligned} \tag{5.2.9}$$

These terms are the same as we would get for the conditional increments of the solutions to (5.2.1) over an interval Δ. This suggests a connection of $\varsigma^h(\cdot)$ to the diffusion $X(\cdot)$. This is indeed the case and it is at this point that the concept of weak convergence will play a dominant role. Now, since $\varsigma^h(\cdot)$ is discrete, we interpolate to make it a continuous process. We set,

$$\begin{aligned} t_0^h &= 0 \\ t_n^h &= \sum_{i=0}^{n-1} \Delta t_i^h \\ \varsigma^h(t) &= \varsigma_n^h \text{ on } [t_n^h, t_{n+1}^h) \end{aligned} \tag{5.2.10}$$

which implies $\varsigma^h(\cdot)$ is constant on random intervals.

Define $\rho^h = t_{N_h}^h$, the escape time of $\varsigma^h(\cdot)$ from $(0,1)$. Thus (5.2.8) becomes,

$$V^h(x) = E_x[\int_0^{\rho^h} k(\varsigma^h(s)\,ds + b(\varsigma^h(\rho^h))] \tag{5.2.11}$$

We still have $E(\rho^h) < \infty$ as before. Now (5.2.11) converges weakly or in distribution to (5.2.3), and this limit is the same by virtue of the uniqueness of the solution of the stochastic differential of X_t.

We can generalize the foregoing techniques to problems of the form which have terms such as V_t or $\sum_{i,j} a_{ij}(x)V_{x_ix_j}$. Again we associate with the PDE a diffusion process governed by

$$\begin{aligned} X(t) &= x + \int_s^t f(X(v),v)\,dv + \int_s^t \sigma(X(v),v)\,dw(v) \\ X(s) &= x \end{aligned} \tag{5.2.12}$$

where f, σ are Lipschitz.

We will require, for reasons that will become clear, that

$$a_{ii}(x,t) - \sum_{i\neq j, j} |a_{ij}(x,t)| \geq 0 \quad i = 1,...n \tag{3.13}$$

so as to guarantee,

$$p^{h,\Delta}(x,x) = 1 - \frac{\Delta}{h^2}[h\sum|f_i(x,t)| + 2\sum a_{ii}(x,t) - \sum_{i\neq j,j}|a_{ij}(x,t)|] \geq 0 \tag{5.2.14}$$

Also define M_Δ by $M_\Delta \Delta = T$ and for $t = 0, \Delta ... \Delta(M_\Delta - 1)$ define

$$p_t^{h,\Delta}(x, x \pm e_i h) = \frac{\Delta}{h^2}\left[a_{ii}(x,t) - \sum_{i \neq j, j} |a_{ij}(x,t)| + h f^{\pm}(x)\right]$$

$$p_t^{h,\Delta}(x, x + e_i h - e_j h) = p_t^{h,\Delta}(x, x - e_i h + e_j h) = \frac{\Delta}{h^2} a_{ij}^{-}(x,t)$$

$$p_t^{h,\Delta}(x, x + e_i h + e_j h) = p_t^{h,\Delta}(x, x - e_i h - e_j h) = \frac{\Delta}{h^2} a_{ij}^{+}(x,t)$$

$$p_t^{h,\Delta} = 0 \quad \forall x, y \text{ not covered above}$$

Again we see that $p_t^{h,\Delta}(x,t) \geq 0$ and sum over y to unity for each x. We conclude that $\{p_t^{h,\Delta}(\cdot,\cdot)\}$ is a one-step transition function for a Markov chain, whose random variables we denote by $\{\varsigma_n^{h,\Delta}\}$. Thus $P[\varsigma_{n+1}^{h,\Delta} = y | \varsigma_n^{h,\Delta} = x] = p_{n\Delta}^{h,\Delta}(x,y)$. We allow for nonhomogeneity in the chain if $a(\cdot,\cdot), f(\cdot,\cdot)$ depend on t.

We can substitute (5.2.4)-(5.2.5) into the equation,

$$(\frac{\partial}{\partial t} + \mathcal{L})V = -k(x,t) \tag{5.2.15}$$

to obtain,

$$V^{h,\Delta}(x, n\Delta) = \sum_y p_{n\Delta}^{h,\Delta}(x,y) V^{h,\Delta}(y, n\Delta + \Delta) + k(x, n\Delta)\Delta, \quad n < M_\Delta \tag{5.2.16}$$

where

$$V^{h,\Delta}(x,T) = b_T(x), \quad x \in G_h$$

$$V^{h,\Delta}(x,t) = b_1(x,t) \quad t \leq T, \quad x \notin G_h \tag{5.2.17}$$

Multiplication of all terms in the finite difference equations by h^2 yield

$$0 = -Q_h(x)V^h(x) + \sum_y p^h(x, x \pm e_i h) Q_h(x) V^h(y) + h^2 k(x). \tag{5.2.18}$$

The above can be written,

$$V^{h,\Delta}(x, n\Delta) = E_{x,n} V^{h,\Delta}(\varsigma_{n+1}^{h,\Delta}, n\Delta + \Delta) + k(x, n\Delta)\Delta$$

$$n < M_\Delta, \quad x \in G_h \tag{5.2.19}$$

where $t = n\Delta, \varsigma_n^{h,\Delta} = x, E_{x,n}$ implies expectation given $\varsigma_n^{h,\Delta} = x$.

Let

$$N(h,\Delta) = \begin{cases} \min\{n : \varsigma_n^{h,\Delta} \notin G_h \text{ if } n \leq M_\Delta \\ \infty \text{ if } \varsigma_n^{h,\Delta} \in G_h \forall n \leq M_\Delta \end{cases} \tag{5.2.20}$$

The latter condition is due to the fact that we have defined the chain only up to time M_Δ. The unique solution to (5.2.19) is

$$V^{h,\Delta}(x,n\Delta) = E_{x,n}\Big[\sum_{i=0}^{(M_\Delta \wedge N(h,\Delta)-1)} k(\varsigma_i^{h,\Delta}, i\Delta)\Delta\Big] + b_T(\varsigma_{M_\Delta}^{h,\Delta}) I_{\{M_\Delta < n(h,\Delta)\}} + b_1(\varsigma_{N(h,\Delta)}^{h,\Delta}, N(h,\Delta)\Delta) I_{\{M_\Delta \geq N(h,\Delta)\}} \tag{5.2.21}$$

It is shown in Kushner [1] that $V^{h,\Delta}(x,t) \to V$ as $h, \Delta \to 0$ *independently of each other.*

We have again that

$$E_{x,n}[\varsigma_{m+1}^{h,\Delta} - \varsigma_m^{h,\Delta} | \varsigma_m^{h,\Delta} = y] = f(y)\Delta \tag{5.2.22}$$

$$\begin{aligned} cov_{x,n}[\varsigma_{m+1}^{h,\Delta} - \varsigma_m^{h,\Delta} | \varsigma_m^{h,\Delta} = y] &= 2a(y)\Delta + h\Delta\tilde{f}(y) \\ &\quad - \Delta^2 f(y) f'(y) \\ &= \Sigma_h(y)\Delta \end{aligned} \tag{5.2.23}$$

and we can set

$$\varsigma_{m+1}^{h,\Delta} = \varsigma_m^{h,\Delta} + f(\varsigma_m^{h,\Delta})\Delta + \beta_m^{h,\Delta} \quad m \geq n \quad \varsigma_n^{h,\Delta} = x \tag{5.2.24}$$

where $\{\beta_m^{h,\Delta}\}$ is an orthogonal sequence whose conditional covariance agrees with (5.2.3).

From this we can interpolate

$$\begin{aligned} F^{h,\Delta}(s) &= \Sigma_{t \leq i\Delta + \Delta \leq s} f(\varsigma_i^{h,\Delta})\Delta, \\ B^{h,\Delta}(s) &= \Sigma_{t \leq i\Delta + \Delta \leq s} \beta_i^{h,\Delta} \end{aligned}, \tag{5.2.25}$$

for $s \in [t,T]$. Thus, dropping the subscript n, we have,

$$\varsigma^{h,\Delta}(s) = x + F^{h,\Delta}(s) + B^{h,\Delta}(s) \quad s \in [t,T] \tag{5.2.26}$$

It turns out, (see Kushner [1]), that,

$$E_{x,t} g(z(\cdot)) \to E_{x,t} g(X(\cdot)), \quad h, \Delta \to 0 \tag{5.2.27}$$

This says in words that $g(z(t))$ converges weakly to $g(X(t))$.

We can draw a connection with the above by noting that in the classical heat equation in n dimensions, where $f(\cdot) = 0, a_{ij} = \Delta\sigma^2/2$, the non-negativity condition (5.2.14) reduces to $1 - n\Delta\sigma^2/h^2 \geq 0$, or $n\Delta \leq h^2/\sigma^2$, which is otherwise known as the von Neumann condition for stability of the finite difference approximation of the heat equation.

We now have a theory that tells us how to set up a finite difference scheme in which we can be sure that the statistical properties of the finite, discrete Markov chain closely approximate the true solution.

3. Implicit versus Explicit Schemes

So many numerical schemes exist for solving PDEs that it often seems difficult to make a proper choice among them. We briefly present here a set of guidelines that will play a role in our decision-making.

Our model problem will be the heat equation, the simplest of the parabolic PDEs. We have

$$\begin{aligned} u_t &= u_{xx}, \quad t > 0, \\ u(x,0) &= f(x), \quad -\infty < x < \infty \end{aligned} \tag{5.3.1}$$

We can discretize in a natural way to obtain,

$$\begin{aligned} &\frac{U(x,t+\Delta t) - U(x,t)}{\Delta t} - \frac{U(x+\Delta x) - 2U(x,t) + U(x-\Delta x,t)}{\Delta^2 x} = 0 \\ &U(x,0) = f(x) \end{aligned} \tag{5.3.2}$$

Letting $\lambda = \Delta t/\Delta^2 x$ we get

$$U(x,t+\Delta t) = \lambda U(x+\Delta x,t) + (1-2\lambda)U(x,t) + \lambda U(x-\Delta x,t).$$

Thus if $\lambda \leq 1/2$ we will have $m \leq U(x,t) \leq M$ if $m \leq f(x) \leq M$.

The condition $\lambda \leq 1/2$ insures numerical stability, otherwise there would be positive and negative oscillation in $U(x,t)$ for some initial conditions. For the n-dimensional heat equation, we would have to have $\lambda \leq 1/(2n)$, which is a burdensome restriction when n is large. Even for $n = 1$ we have the problem that Δt has

to very small, so that for large T, one would have to discretize the interval $[0,T]$ into a large number of subintervals, the result being a computationally intensive procedure. This could pose a problem for us with respect to the Zakai equation, for we may need Δt smaller than the sampling rate, which would require the solution of the PDE in a number of time-steps within the sampling rate interval. However, it must be admitted that explicit schemes have the numerically attractive feature of being reducible to matrix-vector calculations, at the expense of constraining our freedom of choice for Δt and Δx. Indeed, in the one dimensional case, we have

$$U_{k+1} = TU_k, \tag{5.3.3}$$

where T is a tridiagonal matrix with row entries

$$\lambda, \quad 1-2\lambda, \quad \lambda,$$

centered at the diagonal.

Implicit methods have been developed to counteract the confining nature of the stability criterion of explicit schemes. These methods have the advantage of being stable for all values of λ, which means that Δx and Δt can be chosen independently. However, the method suffers from greater numerical complexity.

In an implicit scheme we replace the approximation to u_t in (5.3.1) by

$$\frac{U(x,t+\Delta t)-U(x,t)}{\Delta t} = \frac{U(x+\Delta x,t+\Delta t)-2U(x,t+\Delta t)+U(x-\Delta x,t+\Delta t)}{\Delta^2 x} \tag{5.3.4}$$

the result being

$$\lambda U(x+\Delta x,t+\Delta t)-(1+2\lambda)U(x,t+\Delta t)+\lambda U(x-\Delta x,t+\Delta t) = -U(x,t), \tag{5.3.5}$$

this yields a linear system of the form

$$AU_{n+1} = BU_n \tag{5.3.6}$$

where A and B are known matrices, that is stable in all values of λ.

This approach is suitable only if the problem is defined on a compact set with respect to the space variables, otherwise the matrix is infinite dimensional. But we were given $f(x)$ for $x \in R$, and so in general this technique will not be applicable. (This case is called the initial value problem.) However, if $|f(x)| < \epsilon$ for $[a,b]^c$, then we can use the compact set $[a,b]$ as the domain for the x variable. Generalizations to higher dimensions are obvious. This is precisely what we will do for the Zakai equation, where $f(x)$ corresponds to an initial density that obeys,

$$f(x) \geq 0, \quad \max_{|x| \to +\infty} \{f(x) \text{ on } [x, +\infty)\} = 0 \tag{5.3.7}$$

This fact can be exploited by transforming the Zakai equation to a parabolic PDE and using the theorems appropriate to this class of problems that show that its solution are bounded in a way that is directly related to its initial condition. See chap. 3 for more details.

We conclude that an implicit scheme is appropriate for our purposes, as we wish to choose Δx and Δt independent of the other, the latter variable being given to us in the form of a sampling rate. It is also stable for all choices of Δt and $\Delta x_1, \Delta x_2, ...$

4. Schemes for the Zakai Equation

We will use the methods of Legland, ('81, [2]) who advocates the implicit scheme

$$(I + \Delta t A)u_{k+1} = \Psi_k u_k \tag{5.4.1}$$

where Ψ_k is a diagonal matrix where each diagonal entry is of the form:

$$(\Psi_k)_{ii} = \exp[h(x_i) \cdot (\Delta y_t)_i - \frac{\Delta t}{2}|h(x_i)|^2] \tag{5.4.2}$$

and $A = -L^*$, the operator L^* being associated the Zakai equation.

Equation (5.4.2) is based on a numerical resolution that stems from an approximation the stochastic integral. First we have

$$du_t - L^* u = uh(x)dy_t, \tag{5.4.3}$$

which we write in discrete form as

$$u_{k+1} - u_k + Au_{k+1} \approx \int_{t_k}^{t_{k+1}} uh(x)^T \cdot dy_t \tag{5.4.4}$$

where the left-hand side has Au_{k+1} instead of Au_k to insure an implicit scheme. Now a standard approximation of the right-hand side to the stochastic integral is

$$\int_{t_k}^{t_{k+1}} uh(x)^T \cdot dy_t \approx u_k[h(x)^T \cdot \Delta y_{t_k} + \frac{|h(x)|^2}{2}(\Delta^2 y_{t_k} - \Delta t)]. \tag{5.4.5}$$

combining terms u_{k+1} on one side and u_k on the other yields (5.4.5).

The n-dimensional domain that Legland [2] constructed is as follows. Over *all* of R^n, and given $\epsilon > 0$, we form G_ϵ. Thus $(x_1, x_2, ..., x_n)$ if there exists a $K = (k_1, k_2, ..., k_n) \in Z^n$ such that $x_i = k_i\epsilon$.

Legland's approach was to define the domain over all of R^n so as to avoid issues pertaining to boundary conditions. (These can be treated separately.) However, in order to ensure uniqueness and convergence as $\Delta t, \epsilon \to 0$, he required the following conditions:

For the elliptic diffusion operator,

$$L = a_{ij}D_{ij}(\cdot) + b_iD_i(\cdot) \tag{5.4.6}$$

where

$$D_{ij} = \frac{\partial^2}{\partial x_i \partial x_j}$$
$$D_i = \frac{\partial}{\partial x_i} \tag{5.4.7}$$

is derived from the state equation:

$$dx_t = b(x_t)dt + \sigma(x_t)dw_t$$

$$a_{ij} = (\sigma^T\sigma)_{ij}, \quad b_i = i^{th} \text{ component of } b(x_t)$$

We have

$$a_{ij} \in B_b(R^n)$$
$$|b_i(\cdot)| \leq C(R + |x|) \tag{5.4.7}$$

and for the observation equation:

$$dy_t = h(x_t)dt + dv_t$$
$$h(x) \in C_b(R^n) \tag{5.4.8}$$

(Here, $C_b(r^n)$ is the space of bounded functions on R^n.

Remark: These conditions are stronger than those mentioned in the previous two chapters, which allow for coefficients of polynomial growth.

To combine our results of weaker conditions on coefficients with those of Legland,[2] we choose a function $\psi(x) \in C^\infty(R^n)$ where

$$\psi(x) = 1 \text{ on } \Omega$$

where Ω is the compact domain found by exploiting the asymptotic behavior of the solution of the Zakai equation. In addition, $\psi(x)$ has the asymptotic behavior such that

$$\psi^2 a_{ij} \in C_b(R^n)$$
$$|\psi b_i| \leq C(R + |x|) \tag{5.4.9}$$
$$|\psi h| < M, \quad \text{for some } M \text{ on } R^n$$

This corresponds to perturbing the state equation by

$$dx_t = b(x)\psi(x)dt + \sigma\psi(x)dw_t \tag{5.4.10}$$

($\psi(x)$ could be a matrix if necessary.) For example, for the bilinear equation in one dimension $\psi(x) = O(1/x)$ will be suitable.

Remark: Using this transformation, the theorems of Legland will follow through, and the finite difference scheme on Ω will be what we would obtain had we not used $\psi(x)$, (but without which the theorems of Legland would not be applicable). But as existence and uniqueness have already been decided for us by other methods, we are interested in weak convergence of the finite difference scheme only on the compact set Ω; the behavior of the PDE solution outside of this set is already known to us, as it has been bounded by an appropriate tolerance.

Thus the use of ψ is merely a technical contrivance to accommodate the results of Legland. For example, by proving that the linear system provides solutions that converge weakly to the true solution as $\epsilon \to 0$, Legland [2] showed that the statistical properties of the discretized solution approaches that of the true solution. In effect, the finite probability distribution converges in law to the unnormalized density of the Zakai equation.

The matrix $(I + \Delta tA)$ can now be explored. The reader should note how we make use of the probabilistic information inherent in the matrix to derive its properties. Also, we should also remember that these properties are important from the point of view of numerical analysis, in particular, with regard to the relaxations methods we will ultimately advocate for use in the Multigrid schemes to be described in subsequent chapters.

We first use the scheme by Kushner already described and construct, for fixed

Δx, a sequence of unit vectors $r_i, 1 \le i \le m$ in R^m. Then we have,

$$
\begin{aligned}
Lv(x) =& \frac{1}{2}\sum_i a_{ii}\frac{v(x+\Delta x r_i) - 2v(x) + v(x-\Delta x r_i)}{\Delta^2 x} \\
&+ \frac{1}{2}\sum_{i\neq j}\Big(\frac{1}{2\Delta x}a_{ij}^+(x)\Big(\frac{v(x+\Delta x r_i+\Delta x r_j) - v(x+\Delta x r_i)}{\Delta x} \\
&- \frac{v(x+\Delta r_j) - v(x)}{\Delta x} + \frac{v(x) - v(x-\Delta x r_j)}{\Delta x} \\
&- \frac{v(x-\Delta x r_i) - v(x-\Delta x r_i - \Delta x r_j)}{\Delta x}\Big) \\
&- \frac{1}{2\Delta x}a_{ij}^-(x)\Big(\frac{v(x+\Delta x r_i) - v(x+\Delta x r_i - \Delta x r_j)}{\Delta x} - \frac{v(x) - v(x-\Delta x r_i)}{\Delta x} \\
&+ \frac{v(x+\Delta x r_j) - v(x)}{\Delta x} - \frac{v(x-\Delta x r_i + \Delta x r_j) - v(x-\Delta x r_i)}{\Delta x}\Big)\Big) \\
&+ \sum_i \Big(b_i^+(x)\frac{v(x+\Delta x r_i) - v(x)}{\Delta x} - b^-(x)\frac{v(x) - v(x-\Delta x r_i)}{\Delta x}\Big)
\end{aligned}
$$

We can now write

$$L_h v(x) = \sum_{y\in G_h} L_h(x,y)v(y). \tag{5.4.11}$$

And this implies

$$L_h(x,y) = -\frac{1}{\Delta^2 x}\sum_i (a_{ii}(x) - \frac{1}{2}\sum_{j,i\neq j}|a_{ij}(x)|) - \frac{1}{\Delta x}\sum_i |b_i(x)|$$

$$L_h(x, x+\Delta x r_i) = \frac{1}{2\Delta^2 x}(a_{ii}(x) - \sum_{i,i\neq j}|a_{ij}(x)|) + \frac{1}{\Delta x}b_i^+(x)$$

$$L_h(x, x-\Delta x r_i) = \frac{1}{2\Delta^2 x}(a_{ii}(x) - \sum_{j,i\neq j}|a_{ij}(x)|) + \frac{1}{\Delta x}b_i^-(x)$$

$$L_h(x, x+\Delta x r_i + \Delta x r_j) = L_h(x, x-\Delta x r_i - \Delta x r_j) = \frac{1}{2\Delta^2 x}a_{ij}^+ \text{ for } i\neq j$$

$$L_h(x, x+\Delta x r_i - \Delta x r_j) = L_h(x, x-\Delta x r_i + \Delta x r_j) = \frac{1}{2\Delta^2 x}a_{ij}^-(x) \text{ for } i\neq j$$

$$L_h(x,y) = 0 \text{ for } x, y \text{ not in } G_h$$

We note that $L(x,y) \ge 0$ provided that

$$a_{ii}(x) - \sum_{j,i\neq j}|a_{ij}(x)| \ge 0, \tag{5.4.12}$$

and that $A(x,y) = -L(x,y)^*$, where L^* is simply the transpose of L.

What does $L(x,y)$ look like in matrix form? For the one dimensional case it is tridiagonal with negative entries down the main diagonal and with non-negative entries elsewhere as shown in eq. (5.4.13).

$$\begin{pmatrix} - & + & 0 & . & . \\ + & - & + & 0 & . \\ 0 & + & - & + & 0 \\ & & \cdots & & \\ \cdots & & & + & - \end{pmatrix} \tag{5.4.13}$$

Now suppose the Markov process is at a point x_j. The examining the j^{th} row of L gives three values centered at the diagonal. The absolute value of the diagonal term, or $Q(x)$, is the expected time for the process to remain at x_j starting from the time of its arrival. The value $L_{i,i-1}/Q(x)$ is the transitional probability of going to the point x_{i-1} given the Markov process was at x_i. And the term $L_{i,i+1}/Q(x)$ is the transitional probability of going to x_{i+1} given we are at x_i. We also have the property, as a result of this probabilistic interpretation,

$$\sum_{j;j\neq i} \frac{L_{ij}}{|L_{ii}|} = 1, \tag{5.4.14}$$

implying $L_{ij} \leq |L_{ii}|$.

Note that if $A = -L^T$ then $I + \Delta t A$ has a sign convention negative to that depicted in (5.4.13). Also the diagonal terms will all exceed unit magnitude.

Legland, [2], used the above scheme in an application to radio astronomy. His problem was in one-dimension. We will extend his results to higher dimensions.

Now suppose we have a problem in two space variables and one time variable. Then if our initial density was say, a Gaussian, we could construct a rectangular domain in the space variables. Thus, at each time-step, we are dealing with a rectangular domain in 2-space and attempting to solve the linear system,

$$(I + \Delta t A)u_{k+1} = \Psi_k u_k. \tag{5.5.15}$$

Note that the matrix on the left is independent of time and the observations y_t.

Now suppose we use a *natural ordering scheme* for the grid points. This means that we choose a point at the upper left corner of the grid and label the value of u associated with it $u_n^{(1)}$, i.e., as the first component of the vector u_n. As we move left to right we also form $u_n^{(2)}, u_n^{(3)}, \ldots$ and so on. Such a numbering scheme may not always be permissible or desirable for general domains, but it has an especially attractive appeal here. Furthermore we have a probabilistic interpretation for the matrix $L(x,y)$.

We find that we have a block tridiagonal matrix where the diagonal blocks are of the form as in (5.4.16), while the off-diagonal submatrices are also tridiagonal but with non-negative entries.

$$\begin{pmatrix} D_{11} & D_{12} & 0, & & 0\ldots,0 \\ D_{21} & D_{22} & D_{23} & 0,\ldots 0 & \\ & & 0,\ldots,0 & & \\ 0,\ldots,0 & D_{k,k-1} & D_{k,k} & D_{k,k+1} & 0,\ldots,0 \\ & & 0,\ldots,0 & & \\ 0,\ldots 0, & & 0 & D_{n,n-1} & D_{n,n} \end{pmatrix} \tag{5.4.16}$$

This is the most general case as we are allowing for mixed second order partials. If no mixed partials are present, there are only four nearest neighboring points to choose from. In which case the two *outer* bands in (5.4.16) only unit bandwidth.

Now consider a grid point in the i^{th} row of the domain. This corresponds to the i^{th} row-block of the matrix $L(x,y)$. Now let the grid point be k points from the left. Go to the i^{th} row diagonal block submatrix and examine its k^{th} row. The diagonal entry on the k^{th} of this submatrix is negative, and the absolute value of this term is the expected mean time for the Markov chain to remain at this point. Then the entries to the left and right of this diagonal value value correspond, when divided by the absolute value of the diagonal term, to the transition probabilities of the Markov chain going to the left or right, given we were at the center point. Of course, the Markov chain could also go to the top or bottom row of the domain, or to the northwest, northeast or southwest, southeast points, there being the eight nearest neighbors to choose from in the general case. Transition probabilities of going to

these points can be obtained by observing the submatrices of $L(x,y)$ corresponding to the $(i,i+1)$ block and the $(i,i-1)$ block. Here, on the k^{th} row of each of these submatrices, we find the three numbers centered at the diagonals. In the $(i,i+1)$ block, the three numbers when read from left to right, correspond to probability of moving to the upper left hand corner, directly above, or the upper right hand corner. Similarly, the three values centered at the k^{th} diagonal row of the $(i,i-1)$ block submatrix correspond, when read from left to right, to moving to the lower left hand corner, directly below, and to the lower right hand corner.

We therefore have the

Claim: The matrix $I + \Delta t A$ identical in structure but has non-negative entries opposite in sign to that depicted in (5.4.13).

Generalizations to higher dimensions are also possible, at least for the case of rectangular domains with natural ordering. In three dimensions we would still have a tridiagonal block structure, with the central diagonal band being a repeat of what we had in two dimensions and the outer blocks being tridiagonals corresponding to the points to which the Markov chain could jump, there being at most 27 such points. We would still have positive diagonal entries with non-positive entries elsewhere. Extensions to even higher dimensions still have the same structure: with the preceding matrix being repeated as the central band and the outer block matrices corresponding to the "new" points that the Markov chain can jump owing to our being in a higher dimension. All such matrices are band-limited and very sparse. We might remark at this point that, because of the sparsity of the matrix, storage of the non-zero entries will not be too burdensome.

For simplicity of exposition, we will consider domains in R^n in the form of hypersquares with uniform grid spacing. Generalizations to hyper-rectangular domains will be clear.

Claim: Consider the sequence of hypercubes G_d with n^d points, where d is the dimension of the problem space and n is a fixed number (and equal to the width

of the cube.) Let D_d be the matrix of 0's and $+,-$ signs in the same pattern as would be found in $I + \Delta tA$ corresponding to the Zakai equation discretized on the n^d cube. Thus D_1 obeys the pattern found in (5.4.13) and D_2 obeys the pattern found in (5.4.16).

Then if $I + \Delta tA$ is an $n^d \times n^d$ matrix defined on G_d, $I + \Delta tA$ defined on G_{d+1} is a $n^{d+1} \times n^{d+1}$ of the form:

$$D_{d+1} = \begin{pmatrix} D_d & T & 0,...0 & & 0 \\ T & D_d & T & 0,...0 & 0 \\ 0, & 0,...,0 & & 0,...0 & \\ 0,...,0 & T & D_d & T & 0 \\ 0,...,0 & & & T & D_d \end{pmatrix} \tag{5.4.17}$$

where T is a tridiagonal matrix of *negative* entries. There are also n blocks in D_{d+1}.

This result follows from our natural ordering scheme. We begin our labeling with a d-dimensional hyperplane cutting through G_{d+1} near the boundary. The numbering of this hyperplane yields a matrix pattern identical to what we had in D_d. Now for any point, we have nearest neighbors that correspond to the points in the d-dimensional hyperplane already accounted for, and also nearest neighbors corresponding to the additional increase in dimension found in G_{d+1}. As we can go in a plus or minus direction along this dimensional we have two possibilities, each corresponding to the two submatrices T on both sides of D_d. Remember that T, along with D_d, are matrices that have the same *structure* as we would actually have upon deriving $I + \Delta tA$. Hence, T is a tridiagonal matrix with negative entries.

Now L^* is simply the transpose of L, hence it has the same structure. Thus the matrix

$$(I - \Delta tL^*) = (I + \Delta tA) \tag{5.4.18}$$

has positive terms along the main diagonal. Now we define:

Defn: A matrix $B = b_{ij}$ of order n has strong diagonal dominance if, for all i,

$$|b_{ii}| > \sum_{\substack{j=1 \\ j \neq n}}^{n} |b_{ij}| \tag{5.4.19}$$

Claim: For each dimension d, the matrix $I + \Delta tA$ has strong diagonal dominance. For we have:

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |A_{ij}| \Delta t \leq \Delta t A_{ii} \tag{5.4.20}$$

by (5.4.13). Thus

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} \Delta t < 1 + \Delta t A_{ii} \tag{5.4.21}$$

We make some further observations. If d is the dimension of the problem, and if there are no mixed partials in the Zakai equation, we have at most $2d+1$ nonzero entries in each row vector of our matrix $I + \Delta tA$. Allowing for mixed partials, we have at most 3^d nonzero entries in each row vector.

We form the,

Defn: Let the sparsity measure μ of a matrix be defined as

$$\frac{\text{number of nonzero entries in } I + \Delta tA}{\text{total number of entries in } I + \Delta tA}$$

Claim: For a problem in d dimensions in the discretized hypercube of n^d points, we have

$$\frac{2d+1}{n^d} \leq \mu(I + \Delta tA) \leq \left(\frac{3}{n}\right)^d \tag{5.4.22}$$

This follows as our matrix is of dimension $n^d \times n^d$, thus, for the case of no mixed partials:

$$\frac{n^d(2d+1)}{(n^d)^2} = \mu(I + \Delta tA)$$

Note that if n is kept constant, which is tantamount to having a sequence of hypercubes with constant width but growing in points as n^d and dimension increases, the sparsity is bounded by an exponentially decreasing function $(\frac{3}{n})^d$, if, as is natural to assume, $n > 3$.

Claim: Consider the sequence of hypersquares whose total number of grid points obey the relation n^d, where n is a constant. Then, for dimension d, the bandwidth

of $I + \Delta tA$ obeys

$$b_d = b_{d-1} + 2(n+1) \quad \text{for } d \geq 2 \tag{5.4.23}$$

We note that $b_1 = 3$.

This follows since the derivation of $I + \Delta tA$ for a given dimension (≥ 2) showed that, for the diagonal band of submatrices, each submatrix had the same structure (same $+, -$ and 0 scheme) as was fond in the entire matrix of $I + \Delta tA$ in the preceding dimension. For example, in two dimensions we have a sequence of diagonal blocks any one of which is formed exactly the same way as was done for the one dimensional case. This same pattern is true in general. The $n + 1$ term occurs because our labeling is such that we must move $n + 1$ points to reach the nearest neighbor corresponding to the $(d+1)^{th}$ dimension.

Question: Why don't we find $(I + \Delta tA)^{-1}$ and simply execute the operation:

$$u_{n+1} = (I + \Delta tA)^{-1} u_n?$$

This is a multiplication and therefore simpler to perform than finding the solution of the linear system. Also $(I + \Delta tA)^{-1}$ is precomputable, which would seem to be an added incentive.

But

$$||I + \Delta tA|| = \max_i (I + \Delta tA)_{ii} \left| 1 + \sum_{\substack{j=1 \\ j \neq i}} \frac{I + \Delta tA)_{ij}}{(I + \Delta tA)_{ii}} \right| \tag{5.4.24}$$

and since

$$0 > \sum_{\substack{j=1 \\ j \neq i}} \frac{(I + \Delta tA)_{ij}}{(I + \Delta tA)_{ii}} \geq -1$$

it follows that we could have a nearly vanishing quantity in the $|\cdot|$ term. This would give $||I + \Delta tA||^{-1} >> 1$, which will lead to instabilities.

Weak convergence of the discrete solutions to the true solutions of the Zakai equation in a way that is independent of how Δt and Δx to to zero was shown by Legland, ('81, [2]), although restrictions were made on the Zakai equations requiring

$h(x)$, $\sigma(x)$ to be bounded and $a(x)$ to be at most linear. However, if the PDE problem can be truncated to a bounded domain in a way described in chapter 3, then the same results should hold.

5. Conclusion

We have presented a brief introduction to the theory of weak convergence for finite difference schemes for PDEs. These schemes yield a natural probabilistic interpretation to the underlying stochastic process, which in turn is very useful in describing the structure of the matrix built by our implicit scheme. Also, the discrete solutions to these schemes converge weakly to the true solution as Δx and Δt tend to zero independently of each other.

We have also used the following chain of reasoning in this chapter. First the asymptotic estimates described in an earlier chapter lend themselves to the convenient formation of a rectangular domain. This in turn led to a natural, left-right ordering scheme. The determination of the structure of the resulting matrix in our linear system was aided by its inherent probabilistic interpretation, stemming from the special, implicit finite difference scheme. This revealed a block tridiagonal matrix, denoted as $(I + \Delta tA)$, which has positive diagonal entries and is non-positive elsewhere.

Now when we turn to Multigrid techniques, which can be viewed as any other linear system solvers, knowing the structure of $(I+\Delta tA)$ gives us useful information, such as the fact that the strict diagonal dominance available in all n-dimensional problems, implies that our matrix will require no pivoting. Also, a real, strictly diagonally domain matrix has a non-zero determinant and is positive definite. (For a proof, see Young, [3].)

We will also have to choose a relaxation scheme, which is an iterative process designed to improve our first estimate of the solution. There is a multitude of such schemes, (see Young, [3]), but our choice is determined by the structure of the linear

system. It turns out that we have all the information we need to make an intelligent choice, which will be demonstrated in subsequent chapters.

References for Chapter 5

[1] Kushner, H. and Yu, C., "Probability Methods for the Convergence of Finite Difference Approximations to Partial Differential Equations," *Jour. of Math. Analysis and Appl.*, vol. 43, 1973.

[2] Legland, F., *Estimation de Parametres dans les Processes Stochastiques en Observations Incomplete,* Ph.D thesis, L'Université de Paris, IX-Dauphié, 1981.

[3] Young, D., *Iterative Solution of Large Linear Systems,* Academic Pr. 1971.

6. Elements of Multigrid Theory

1. Introduction

This chapter is intended in part only as an overview of the Multigrid theory. We cannot possibly give a detailed survey of all of its technical fine points, nor would this even be desirable, as much of the general theory would only vaguely be related to our own, or future, research. Our own results, which are directly relevant to the Zakai equation, will also be presented.

Although we will keep our account relatively straightforward, the reader should be forewarned of the following facts. Multigrid theory is still in its infancy, having essentially begun in the late 70's; there are many elementary questions that have yet to be answered. It is therefore more an art than a science, and many of the techniques are *ad hoc* and somewhat lacking in theoretical validity. This also means widespread differences in notation and orientation among the various authors, (for a bibliography of the Multigrid literature, see Brand, [2]). For example, the theory has found perhaps its widest audience among researchers in fluid dynamics, and so the literature is often biased in their favor and their more specialized problems.

Another contention is that theoretical research, especially in the form of convergence and complexity analysis, is viewed by the practitioners as being too pessimistic when providing bounds on performance. They prefer what is known as *model problem* investigations, in which the numerical results of a typical problem are treated as a benchmark for all similar cases. This, of course, lacks generality, but bounds on performance inferred from such work are usually tighter than those of the theoretical approach.

We will employ this same approach. The standard model problem that is used throughout the Multigrid literature is Poisson's equation on a two-dimensional rectangle. While not discounting the usefulness of this work, we prefer more relevant models, so we proceed with a Multigrid analysis of the heat equation and continue to

the more general Fokker-Planck equation. While the techniques we use are standard tools of Multigrid theory, we believe the work presented here is original.

The outline of this chapter is taken largely from Brandt, [3]-[4], Stüben and Trottenberg, [12], with convergence results from Hackbusch [8]-[9]. Our strategy is as follows. We assume little or no familiarity with the Multigrid algorithm on the part of the reader, who will soon find that the theory is not conceptually difficult, just somewhat detailed. Therefore we will begin with simple examples and proceed to add more and more complexity as we go, rather than start off with the full abstract theory.

The reader is also encouraged to peruse the excellent *Multigrid Methods* published by Springer-Verlag, [8].

Main Results: demonstration of the relevance of the Multigrid algorithm to rapid numerical analysis of the Zakai equation, development of Fokker-Planck equations as Model problems of the Multigrid method. Questions regarding convergence and stability of methods with respect to parametric changes explored.

2. The Multigrid Perspective

The fundamental idea of Multigrid or MG theory is easy to understand, and especially transparent using only two grids and in two dimensions, but by no means is the theory restricted to this special case. Suppose a PDE to be discretized on the finer grid G_h, which we assume for the sake of simplicity to have uniform grid spacing h, and that an approximate solution u^h to the discrete $L^h U^h = f^h$ is given. (Like all numerical methods, the MG method requires an initial approximate solution. Usually, such approximations are obtained essentially for free; for example, a solution from a "nearby" problem could be the supplier. For the time-dependent problems which interest us, the logical candidate is the value of the solution at the previous time-step). Here, the term h refers to the fineness of the grid, which has been determined from an implicit finite difference approximation of the PDE and

its stability requirements. Now imagine that we had the true discrete solution U^h. Then we could find the Fourier expansion of the difference on the finer grid G_h,

$$u^h - U^h \sim \sum_{k=0}^{\infty} a_k e^{i\omega_k x}. \tag{6.2.1}$$

Now conduct a *relaxation* on the fine grid. This involves the use of an *iterative* procedure such as Gauss-Seidel, Jacobi, factorization, or many others. While these methods will be described in more detail later in this paper, suffice it to say at present that, given a good initial approximation as a starting point for iteration, *all* such methods are very effective at reducing high frequency error. Thus, after only a few relaxation sweeps or iterations, such as four or five or even less, the terms in the Fourier series in (6.2.1) are transformed:

$$\sum a_k e^{i\omega_k x} \Rightarrow \sum \tilde{a}_k e^{i\omega_k x},$$

where

$$a_k \approx \tilde{a}_k \text{ for } k \text{ "small"},$$

and

$$|a_k| >> |\tilde{a}_k| \text{ for } k \text{ "large."}$$

The value of the solution after smoothing can be measured by examining the *defect equation*,

$$d^h = f^h - L^h u^h, \tag{6.2.2}$$

which is derived by defining,

$$v^h = U^h - u^h, \tag{6.2.3}$$

implying that we solve,

$$d^h = L^h v^h. \tag{6.2.4}$$

It is clearly in our interests to make d^h as small in norm as possible, and since L^h is assumed to be linear, continuous and to have full rank, this implies v^h is also small in norm as is desired.

Because of this newly obtained smoothness, the defect equation can be transferred to a *coarser* grid without the loss of too much information, since the highly oscillatory error components have died down. Then the correction denoted by v^{2h} which solves

$$L^{2h}v^{2h} = d^{2h}$$

can be found, by directly solving this less computationally expensive equation. (We will only briefly remark that in the multiple grid case, once transferred to the coarser grid, the identical procedure can be repeated with a still coarser grid. This in effect, repeats the whole process.) The correction is transferred back again to the finer grid by interpolation, which yields v^h and the sum $u^h + v^h$ is then 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.

Three elements of the MG method can so far be identified:

1). error smoothing by relaxation

2). calculation of corrections on coarser grids and recursive application

3). combination with nested iteration.

We will go over the above ideas in more detail in this chapter.

At this point one might well ask: why use Multigrid methods at all? Why not solve the linear system directly, or exclusively use iterative methods and allow them to converge? The answer is that direct solvers have a computation time that grows linearly with n, the width of the finest grid, while Multigrid methods can actually do better than this. What is more, 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 theory, such schemes are used only for smoothing the high-frequency error as a prelude to intergrid trans-

fers, which, we might 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 a ideal candidate for investigation.

It is difficult to pinpoint when and how Multigrid methods originated. The smoothing effects of relaxation techniques were known at least in the mid 1940's and the use of coarser grids for an efficient solution to a system on a finer grid was investigated in the 1950's, but no smoothing was utilized. These came under the heading of *reduction methods*. Also, Russian mathematicians employed the concept of nested iteration in the early 60's. (See Stüben and Trottenberg [12]for a discussion of this early history).

Multigrid theory as we know it today is due largely to Brandt, who developed it in the 70's (see [3]-[4]). He introduced nonlinear techniques, adaptive grids, the possibility of local refinement and the use of "local Fourier analysis" for studies in convergence and optimization. These terms will be expounded upon later.

Research in Multigrid methods began in earnest after 1977, most of it centering around Finite Element methods and numerical fluid mechanics. Problems such as singular perturbation phenomena, transonic flow, shocks, the treatment of Euler equations and the Navier-Stokes equations received special attention.

What has been the response to MG techniques over the last decade? Numerical analysts who gave their loyalty to older, more established methods were frequently resentful of the new theory, mainly for two reasons: the original theoretical analysis

gave rather pessimistic upper bounds on complexity, and attempts to duplicate the algorithm often did not make use of the various "tricks" the MG experts had at their disposal. Hence performance was less than expected. In the case of *standard problems* (simple elliptic 2-dimensional problems of second order), MG techniques were first thought to fare worse than direct fast solvers and other methods. The development of expertly designed programs have shown that MG methods are at least competitive in these areas in addition to being more readily extended to higher dimensions. In fact, their decisive advantage is that they can be applied easily to problems which do not meet the requirements demanded by direct fast solvers and capacitance matrix techniques. These latter techniques have had perhaps their best success with Poisson's equation.

Finally, we mention that there is still active research in Multigrid theory still going on. Most of it centers around the following issues:

1). Multigrid methods for nonlinear PDEs.

2). Complicated domain problems, such as crack deformation studies or domains with very sharp corners.

3). Adaptive techniques involving domain shut-down or extension, or structural changes in the algorithm itself.

4). Integration of Multigrid techniques with Finite Element methods and their more difficult grid topologies.

5). Irregular boundary conditions.

Fortunately, as the problems listed above are far from resolved, we will avoid these issues in our own research.

This section was designed only to provide a brief overview of Multigrid theory. The next section will give a more detailed presentation.

3. Multigrid Algorithms

We now present here, in earnest, a brief explanation of the theory and practice of multigrid algorithms.

In its most general setting, multigrid problems are stated as the following:

Find $u \in H$ such that $a(u,v) = f(v), \quad \forall v \in H,$

where H is a Hilbert space, $a(\cdot,\cdot)$ is a continuous symmetric bilinear form on $H \times H$, $f : H \to R$ is a continuous linear functional. In general, for elliptic PDEs, we also assume that $a(\cdot,\cdot)$ is coercive, which implies , $a(v,v) \geq c_0||v||^2, \quad \forall v \in H,$ and which guarantees a unique solution. However, the parabolic case can also be included and certain degeneracies allowed as long as uniqueness and stability of the discretized linear system is guaranteed.

The above problem will, of course, correspond to a PDE of the form,

$$LU(\bar{x}) = f(\bar{x}) \quad \bar{x} \in \Omega \subset R^d, \tag{6.3.1}$$

with suitable boundary conditions. We will also assume throughout this discussion that L is linear. For purposes of illustration, it will be more useful to concentrate on the formulation in (6.3.1). In performing our numerical analysis, we would begin by discretizing it in a pre-assigned way on some bounded uniform grid G_h with mesh size h, which would have n^d points. Assume, for the sake of simplicity that the domain is a rectangle. The resulting finite-difference equations would be

$$L^h U^h(\bar{x}^h) = 0 \quad \bar{x}^h \in G_h, \quad U^h = \Phi^h \text{ on } \partial G_h \tag{6.3.2}$$

Now assume our grid-points to be arranged on the rectangle in such a way that $U^h(i,j)$ corresponds to a value of U^h at the point (i,j). Then we can map this array of point values into a vector where

$$U^h(i,j) \to U_{(i-1)n+j}$$

where n corresponds to the width of the rectangle. Hence the equation in (6.3.2) can be viewed as representing this a linear system with unknown vector solution

U^h. Generalizations for higher dimensions also exist. Of course, this is just a formal representation. When we come to a discussion of how to implement the Multigrid method on an actual system, different representations will be needed.

Thus $U^h(\bar{x})$ is a discrete approximation to $U(\bar{x})$ and at various stages of the solution process we will have on G_h an approximation to U^h which we can denote as u^h.

The multigrid approach involves adding to G_h a sequence of coarser uniform grids. Let G_{2h} be such a grid, with its mesh size of $2h$. Then one way of obtaining the approximation u^h to U^h is first to obtain an approximate solution u^{2h} which corresponds to the problem,

$$L^{2h}u^{2h}(\bar{x}^{2h}) = \bar{x}^{2h}, \quad \bar{x}^{2h} \in G_{2h}, \tag{6.3.3}$$

which is less expensive as it contains half as many unknowns. Then u^h would be obtained by interpolation,

$$u^h = I_{2h}^h(u^{2h}),$$

where the symbol $I_{2h}^h(\cdot)$ stands for the interpolation process from G_{2h} to G_h. The simplest interpolation scheme would be the linear case in one dimension defined by

$$I_{2h}^h(u^{2h}(\bar{x}^h)) = \begin{cases} u^{2h}(\bar{x}^h) & \text{if } \bar{x}^h \in G(h) \\ \frac{1}{2}(u^{2h}(\bar{x}^h - \bar{h}) + u^{2h}(\bar{x}^h + \bar{h})), & \text{otherwise,} \end{cases}$$

where $\bar{h} = (h, h, ...h)$, d times. If the above is a first-order interpolation then higher-order interpolations are also possible and can be performed as sequence of one-dimensional interpolation. Sometimes the higher-order approach is necessary to produce better approximations to u^h. It can be shown that the optimal order depends in a simple way on three numbers: *a*), the order of the differential equation, *b*), the order of the discretization error, and *c*), the order of the derivatives we seek to approximate. (The reader should also convince himself that obtaining an approximate solution u^{2h} could also be performed by solving an equation similar to (6.3.2) on G_{4h}, and then using interpolation. Hence a sequence of nested grids can be readily constructed.)

Of course, other interpolation and injection schemes are available. In the case of two dimensions, we assume we have grids where points have been numbered in such a way that the coarser grid obeys,

$$G_{2h} = \{x_{2h} = 2\kappa h : \kappa \in Z^2\}.$$

Examples could therefore be $x_h \in (-3h, 2h) \in G_h$ while $(-6h, 4h) \in G_{2h}$.

Now, as an example of injection, let $w_h(x)$ be a function defined on G_h. Define $V \subset Z^2$ and α_κ to be a sequence of real numbers corresponding to each $\kappa \in V$. Then,

$$I_h^{2h} w_h(x) = \sum_{\kappa \in V} \alpha_\kappa w_h(x + h\kappa).$$

A typical choice of such an injection operator is to define V to contain a subset of the set of elements of the form:

$$(\pm 1, \pm 1), \quad (0, \pm 1), \quad (\pm 1, 0), \quad (0, 0).$$

The injection operator would then have elements of the form:

$$\begin{pmatrix} \alpha_{-1,-1} & \alpha_{0,1} & \alpha_{1,1} \\ \alpha_{-1,0} & \alpha_{0,0} & \alpha_{1,0} \\ \alpha_{-1,1} & \alpha_{0,1} & \alpha_{1,1} \end{pmatrix} \tag{6.3.4}$$

of which a typical example would be the Full Weighting operator:

$$\frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{6.3.5}$$

Using the vector notation discussed before, with U^{nm} vector, the injector can be thought of as a $nm/4 \times nm$ matrix whose row values correspond in a natural way to appropriate values taken from (6.3.5). The result will be a $nm/4$ vector.

In a similar way an interpolation operator from G_{2h} to G_h can also be defined. To do this we define, for each $y \in G_{2h}$,

$$w_{h,y}(x) = \begin{cases} \beta_\kappa w_{2h}(y) \text{ for } x = y + h\kappa \text{ with } \kappa \in V \\ 0 \text{ for } x = y + h\kappa \text{ with } \kappa \notin V \end{cases}$$

Thus we set

$$w_h(x) = \sum_{y \in G_{2h}} w_{h,y}(x).$$

As in the previous case, the collection of coefficients $\beta_{i,j}$ can vary but the most frequently used method is the bilinear interpolator represented by

$$\frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{6.3.6}$$

As in the remark above, a matrix interpretation is also possible for the interpolator, namely as a $nm \times nm/4$ matrix with row values corresponding to the expression in (6.3.6).

It turns out that the full weight injection and bilinear interpolation operators are adjoint to one another. Extensions to higher dimensions are also possible.

The interpolation accuracy depends on the smoothness of U^h. In fact, U^h is often so smooth that if the orders of the interpolation and the coarse grid operator are high enough so as to properly exploit that smoothness, then $u^h = I_{2h}^h(u^{2h})$ will satisfy,

$$||u^h - U|| = O(||U^h - U||).$$

In words, we say that u^h solves $L^h U^h(\bar{x}^h) = f^h$ "to the level of the truncation error," which is the best that any numerical procedure can reasonably hope to attain. However, in such cases, the fine grid would not really be needed at all, the coarser grid already providing a solution of sufficient accuracy. Thus if the fine grid G_h is at all needed, the approximation u^h will require considerable improvement.

Now that we have u^h, we wish to compute a correction to it. What we need is an approximation to $v^h = U^h - u^h$. Such an approximation $\tilde{v}^h$ to v^h would yield $u^h + \tilde{v}^h$, which would serve as our "correction." We find that,

$$L^h v^h = d^h, \quad v^h = \phi^h \text{ on } \partial G_h \tag{6.3.7}$$

which corresponds to

$$L^h u^h = f^h - d^h \quad u^h = \Phi^h - \phi^h \text{ on } \partial G_h, \tag{6.3.8}$$

where we have assumed that L^h is linear. Equations (6.3.7)-(6.3.8) are known as the residual equations and it turns out that the right hand side, d^h, will usually fluctuate rapidly on G_h with wavelength less that $4h$. If we try to solve (6.3.7) on the coarser grid, as we tried before with (6.3.2), we will lose this high-frequency information and obtain a very poor approximation to v^h.

Presumably, what is needed is an error smoother, i.e., some operation on u^h so that the error $v^h = U^h - u^h$, or d^h, is no longer rapidly oscillating. An efficient error smoother is obtained by *relaxation sweeps*. For purposes of illustration, a good example is the Gauss-Seidel relaxation sweep. (This is not the method we would propose in general, as it is not very suitable for parallel implementation, but it, perhaps, best demonstrates the concept of relaxation). In this process, the old value $u^h(\bar{x}^h)$ at each point is replaced by a new value which is computed so that (6.3.8) is satisfied at that particular point $\bar{x}^h$. Each point $\bar{x}^h$ is then scanned one by one in a prescribed manner. After one sweep the system (6.3.8) is not solved, due to coupling of the equations, but the new approximation u^h is hopefully better than the old one.

In principle, with more sweeps, this technique can actually solve (6.3.8), but not very efficiently, due to its slow convergence. However, the technique appears to remove the high-frequency, components of the rapidly oscillating error very well in just a few sweeps. (For a more indepth discussion of this behavior, see chapter 8.)

When relaxation sweeps tend to converge (successive differences between them become small), they should be discontinued. Now we have a function, $\tilde{v}^h = U^h - \tilde{u}^h$ where $L^h \tilde{v}^h = \tilde{d}^h$ which can be successfully solved on G_{2h} without loss of too much accuracy, since the high frequency components of the error are now virtually gone. Thus we would have $L^{2h}(v^{2h}) = I_h^{2h}(f^h)$ where I_h^{2h} might simply be an injection from G_h to G_{2h}. The role of relaxation methods in multigrid techniques is not to reduce the error as much as to smooth it out so as to allow such coarse-grid approximations. In fact, relaxation sweeps make up most of the multigrid work.

To recapitulate what we have done so far:

1). We went from a fine grid to a coarse grid to solve less expensive approximation.

2). New solution u^h was obtained by interpolation from coarse to fine grid.

3). Error in fine grid solution was smoothed out by relaxation.

4). Converted back to coarse grid for improved solution.

5). Now have solution of form $u^h + v^h \approx U^h$. This can be further smoothed if desired.

The process can be further generalized by noting that (6.3.8) can be treated as above, namely, by obtaining an initial approximation from a still coarser grid G_{4h}, then relaxing in G_{2h}, and then obtaining a coarse grid correction using G_{4h} again.

A remark on the varieties of grid designs is in order. Throughout this chapter we will be using the simplest grid design (G_h, G_{2h}), only because it is the most accessible. But it would be a mistake to suggest that this is the only option open to us. We therefore briefly describe some other standard grid designs, with the understanding that still others are available.

Four possibilities of G_H, the coarser complement to G_h, are:

1). Standard coarsening: $H = 2h$

2). Semicoarsening: x_1-coarsening, $H = (2h_{x_1}, h_{x_2})$ or x_2-coarsening, $H = (h_{x_1}, 2h_{x_2})$.

3). Red-Black coarsening: There are many variations here. For a simple example, consider a square grid, $h = h_{x_1} = h_{x_2}$. Then G_H can be identified with a stretched, rotated grid of mesh size $H = \sqrt{2}h$:

$$G_H = \{x = h\kappa : \kappa = (\kappa_1, \kappa_2) \in Z^2, \kappa_1 + \kappa_2 \text{ even}\}.$$

4). Quadrupling $h : H = 4h$.

Each of these designs is especially suited to certain kinds of relaxation schemes, which will be discussed in a chapter devoted to such methods. Other details and

variations of the Multigrid method exist as well and have been especially designed for nonlinear problems. More information can be found in Stüben and Trottenberg, [12].

In the words of Brandt, [4]: "The multigrid methods are systematic methods of mixing relaxation sweeps with approximate solutions of residual equations on coarser grids. The residual equations are in turn also solved by combining relaxation sweeps with corrections through still coarser grids, etc. The coarsest grid is coarse enough to make the solution of its algebraic system inexpensive compared with, say, one relaxation sweep over the finest grid."

Our problem then is on a rectangle in R^d with a regular n^d point grid. The complete Multigrid cycle can now be described. The number of grids used is typically three or four, although in principle one can proceed from a fine grid of processors all the way down to a single processor. The idea is still the same, smooth on finer grids, using coarser grids to give approximations for the relaxations, and direct solve with the coarsest grid.

Case 1: 2-level grid

One complete multigrid cycle would do the following:

1). Given an initial approximation, conduct a few relaxations sweeps on the finest grid.

2). Inject the defect equation to coarse grid and direct solve. With half the grid points, this will be half the computational expense of using the first grid.

3). Interpolate from the coarse to fine grid and using the approximation from 2), continue relaxation sweeps. The result should be within the truncation error.

Case 2: 3-level grid

For one cycle we proceed as before:

1). Smooth on finest grid with initial approximation

2). Inject defect equation to second grid and smooth, with zero as an initial solution

3). Inject to coarsest grid and direct solve

4). Interpolate to second grid and smooth

5). Interpolate to finest grid and smooth

For two inner cycles on three grids we have an inner loop that acts like the 2-level case:

1). Smooth on finest grid with initial approximation

2). Inject to second grid and smooth

3). Inject to coarser grid and direct solve

4). Interpolate to second grid

(Up to now it is the same as before. Now the inner loop begins. Compare with Case 1.)

5). Smooth on second grid

6). Inject to coarsest grid and direct solve

7). Interpolate to second grid and smooth

The inner loop is now finished.

8). Interpolate to finest grid and smooth

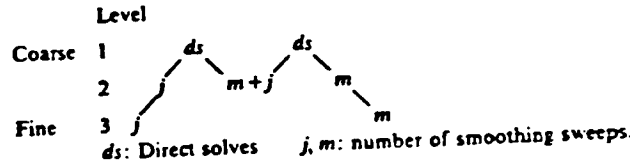
The case of three cycles on a 3-level system contains two inner loops identical to the 2-level method. Extension to the 4-level case is equally straightforward. By now the reader can easily follow the diagrams in fig. (6.3.1). Examination of fig. (6.3.1) reveals why MG practitioners say that their programs exhibit a "V" or "W" shape structure, which corresponds to one or more inner cycles of the MG algorithm. Once again, the number of smoothing sweeps may vary on each grid, and this number will be preselected.

The following questions come to mind:

1). For a given problem, how many grids and cycles should one use?

2). What choice among the many available smoothing algorithms should be made, and how is the number of sweeps to be performed on each grid to be selected?

One would hope that the Multigrid theory would be sufficiently advanced that,



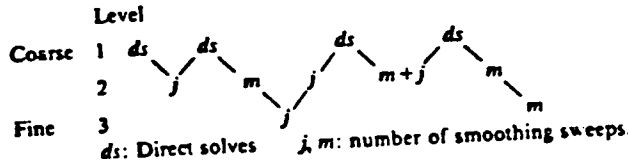


Figure 6.3.1 Schematics of Multigrid Algorithms

given a problem, one could derive a recipe of grid levels, cycle and smoothing sweep numbers from which to write an MG program that provides truncation error accuracy in optimal time. Unfortunately, the MG theorists are unable to offer such advice except in special cases, and even then their estimates turn out to be more conservative than optimal. As Brandt, [4], points out the typical problem is that the numerical analyst, new to the intricacies of the multigrid method, will construct a program to solve a PDE, but finding that it will work, will still wonder if this is the best efficiency on could hope for. Because of this, the current strategy among the practitioners is to exploit model problems that are representative of a class of problems. While convergence can already be guaranteed by theory, any program optimization can be performed on the model and shown to hold for all "related" problems. Thus, it is mostly by numerical experimentation, guided by theoretical

considerations, that lead to optimal designs.

For our purposes, we would use a Zakai equation with the dy_t term set equal to zero. (This leaves the Fokker-Planck component.) Then we can fine-tune the algorithm so as to work as fast as possible, and then show our design will still hold when observations drive the equation. A discussion of convergence for the model problem of this equation will be provided in a later section.

As for relaxation methods, the number of sweeps on each grid could also be determined by model problem analysis, but the following considerations would have to be made regarding the actual choice of the relaxation method:

1). The method must be suitable for the PDE being analyzed. Brandt [4] argues that the choice of relaxation technique is the most problem-dependent aspect of the MG method. The idea is to smooth out the high-frequency error components as rapidly as possible.

2). The method must be suitable for high-speed parallel implementation. The reader will see that such methods vary widely in this regard.

3). If desired, the method must be suitable for asynchronous operation with the necessary architectural over-head being as low-cost as possible. Later we will show why such a feature might be desirable.

One can imagine the class of relaxation methods being subjected to an approval rating under each of the above criteria in turn. Under each category, an appropriate subset is formed. Presumably, the intersection of the three sets would form the choices we could make. In a subsequent chapter, we will do exactly this, and show that such an intersection is non-empty. We will then provide the appropriate relaxation method to be incorporated into the Multigrid algorithm to solve the Zakai equation.

4. The Recursive Structure of Multigrid

The reader has probably already detected the built-in recursive nature of the MG algorithm. This becomes especially lucid when a pidgin ALGOL language is used to describe the program, as will be seen in a later chapter. For now we will remain content with a more detailed listing of a one-cycle full Multigrid program. Let there be K point-grids which we will denote by $G_1, G_2, ..., G_K$ with the finest being G_K and the coarsest being G_1.

The finest grid contains the problem:

$$L^K U^K = f^K. \tag{6.4.1}$$

Smoothing Part I:

Given an initial approximation to the problem in (6.4.1), smooth j_1 times to obtain u^K.

Coarse-grid correction:

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

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}:

$$L^{K-1} v^{K-1} = d^{K-1}, \tag{6.4.2}$$

by performing $c \geq 1$ iterations of the Multigrid method, but this time we will be using the grids $G_{K-1}, G_{K-2}, ..., G_1$ applied to equation (6.4.2).

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

Compute the corrected approximation on G^K,

$$u^K + \tilde{v}^K. \tag{6.4.3}$$

Smoothing Part II:

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

The reader will detect the recursive structure of the algorithm entering just after eq. (6.4.2). 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 structure as in fig. (6.3.1). Only if $c > 1$, would we have a "W shape" structure.

In chapter 7, we will show how this recursive structure can be exploited to provide a quantitative complexity analysis of the Multigrid algorithm. We will, in turn, use these same results, to estimate performance times of the algorithm with respect to our own signal processing needs.

5. Outline of Proof of Convergence for the Multigrid Algorithm

This section will review some standard theorems from Multigrid convergence theory. They will later be applied to our own Zakai equation.

The proof of convergence of MG techniques rests on the elementary properties of iterative processes. We consider a problem of the form

$$Ax = b \tag{6.5.1}$$

where $x, b \in R^n$ and A is an $n \times n$ invertible matrix. Given an initial guess, x_0 of the true solution x, we can find an iterative method of the form

$$x_{n+1} = (I - MA)x_n + Mb \tag{6.5.2}$$

where M is nonsingular. Equation (6.5.2) obeys the structure of a *consistent* method, in which the true solution, x, would be fixed point of the mapping in

(6.5.2), i.e.,

$$x = (I - MA)x + Mb \tag{6.5.3}$$

which is easily verified by inspection, using (6.5.2). Convergence to the true solution, however, will depend on the spectral radius of $I - MA$, which in turn depends on the choice of M and the given properties of A.

Also note that in the case of the equation,

$$L^h U^h = f^h, \tag{6.5.4}$$

we can write the formula for obtaining a new approximation to the solution U^h from an old one as follows,

$$\tilde{u}^h = (I - ML^h)u^h + Mf^h. \tag{6.5.5}$$

Now we have from (6.5.2) that

$$x_k = (I - MA)x_{k-1} + Mb$$

$$x_p = (I - MA)x_{p-1} + Mb$$

and so

$$x_k - x_p = (I - MA)(x_{k-1} - x_{p-1}). \tag{6.5.6}$$

Letting $k - p = \gamma \geq 0$ for a moment, we have

$$x_k - x_p = (I - MA)^p(x_\gamma - x_0),$$

where we have used the iterative structure of the process. Then if we define:

$$\rho(I - MA) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } I - MA\} \tag{6.5.7}$$

then $\rho(I - MA) < 1$ implies that, if γ is kept constant, but p (and hence $k = \gamma + p$) is allowed to go to infinity, we have

$$||x_{p+\gamma} - x_p|| \to 0.$$

The arbitrariness of γ implies that the Cauchy sequence converges.

Our goal is to use the same concept for the MG method. We clearly need an operator formalism to do this, and so, to this end, we define S_k as the smoothing operation on grid G_k. Thus the new approximation using the old one $\tilde{u}^k$ for the smoother will be

$$\tilde{u}^k = S_k u^k = (I - ML^k)u^k + Mf,$$

where we assume M is invertible and the smoother consistent. And we will therefore say that 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^h, 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^h, 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^h, which forces $\rho(I - ML^h)$ 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 will 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

$$\tilde{u}^k = \mathbf{M}_k u^k + I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1} f^k.$$

$\mathbf{M}_k$ will be the Multigrid operator on grid G_k we will concentrate on, for it is its spectral radius that determines whether iteration converges or not. By $\mathbf{M}_k^c$ 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:

$$\mathbf{M}_2 = S_2^{j_2}(I_2 - I_1^2(L^1)^{-1}I_2^1L^2)S_2^{j_1}$$
$$\mathbf{M}_{k+1} = S_{k+1}^{j_2}(I_{k+1} - I_k^{k+1}(I_k - \mathbf{M}_k^c)(L^k)^{-1}I_{k+1}^kL^{k+1})S_{k+1}^{j_1}. \tag{6.5.8}$$

These equations can be easily shown by induction. First look at the definition of $\mathbf{M}_2$ and note that we have consistency since $S_2^j(U^2) = U^2, L^2U^2 = f^2$ by assumption. Equation (6.5.8) is also a slight modification of (6.5.2), but it still consistent since $S_2(\cdot)$ is consistent. Note that the role of M is played by $I_1^2(L^1)^{-1}I_2^1$, which is indeed an approximation to L^2, in keeping with an earlier remark that this should keep $\rho(\mathbf{M}_2) < 1$. The initial approximation is acted on starting from the right, with j_1 sweeps of the smoothing operator $S_2^{j_1}$ to give $\tilde{u}^2$. The term $L^2(\cdot)$ is then applied. However, *before* injection, write

$$\begin{aligned} L^2\tilde{u}^2 &= f^2 - f^2 + L^2\tilde{u}^2 \\ &= f^2 - d^2 \end{aligned} \tag{6.5.9}$$

Now apply the linear operations to yield,

$$\begin{aligned} &\mathbf{M}_2L^2\tilde{u}^2 + I_1^2(L^2)^{-1}I_2^1f^2 = \\ &\qquad S_2^{j_2}(\tilde{u}^2 + I_1^2(L^1)^{-1}I_2^1d^2 - I_1^2(L^1)^{-1}I_2^1f^2) + I_1^2(L^1)^{-1}I_2^1f^2. \end{aligned} \tag{6.5.10}$$

Note that $f^2 = L^2U^2$, and we assume

$$I_1^2(L^1)^{-1}I_2^1L^2U^2 = U^2. \tag{6.5.11}$$

In other words, our theoretical approach assumes "nearly perfect" succession of interpolation and injection operations with no round-off noise. Thus $S_2^{j_2}(U^2) = U^2$, and this cancels with the outside term in (6.5.10). (Presumably, in practice, if equation (6.5.11) were only approximate, the j_2 relaxations would yield virtually the same result.)

Of course,

$$I_1^2(L^1)^{-1}I_2^1d^2 = v^2,$$

by definition. The rest of the equations can be derived similarly.

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.) Then it interpolates the result via I_1^2 to the grid G_2. The result is v^2. The reader can see that we will finally apply j_2 smoothing sweeps to

$$\tilde{u}^2 + v^2,$$

which agrees with eq. (6.5.8). Generalizations from this special 2-grid case are straightforward. The reader should also note the self-referential structure in eq. (6.5.8). Once again, note that we are only interested in that part of the formula in (6.5.8) that determines convergence; in particular, we must show $\rho(\mathrm{M}_k) < 1$.

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

$$\begin{aligned} M_k^{k-1} &= S_k^{j_2}(I_k - I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1} L^k) S_k^{j_1} \\ A_k^{k+1} &= S_{k+1}^{j_2} I_k^{k+1} : G_k \to G_{k+1} \\ A_{k+1}^k &= (L^k)^{-1} I_{k+1}^k (L^{k+1})^{-1} S_{k+1}^{j_1} : G_{k+1} \to G_k \end{aligned} \tag{6.5.12}$$

Thus we can write,

$$\mathrm{M}_{k+1} = M_{k+1}^k + A_k^{k+1} \mathrm{M}_k^c A_{k+1}^k,$$

a fact the reader can verify by inspection of eq. (6.5.8).

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

(Remark: we denote Stüben and Trottenberg as S & T.)

Theorem: (S & T, [12]) Let the following estimate be assumed known for $k \leq K-1$,

$$||M_{k+1}^k|| \leq \sigma, \quad ||A_k^{k+1}|| \cdot ||A_{k+1}^k|| \leq C. \tag{6.5.13}$$

Then,

$$||\mathrm{M}_K|| \leq \nu_K, \tag{6.5.14}$$

where ν_K is recursively defined by

$$\nu_2 = \sigma, \quad \nu_{k+1} = \sigma + C(\nu_k)^c, \quad k = 2, 3, ..., K-1. \tag{6.5.15}$$

(As stated before, this somewhat awkward notation is in keeping with our grid labeling. As there is no $\mathbf{M}_1$, there is likewise no ν_1.)

Proof: The result is straightforward. We first note that

$$\mathbf{M}_2 = M_2^1,$$

and so

$$||\mathbf{M}_2|| = ||M_2^1|| \leq \sigma = \nu_2.$$

Thus

$$||\mathbf{M}_{k+1}|| \leq ||M_{k+1}^k|| + ||A_k^{k+1}||\,||A_{k+1}^k||\,||\mathbf{M}_k||^c.$$

Replacing $||\mathbf{M}_k||^c$ by $(\nu_k)^c$ gives the result. **Q.E.D.**

To develop a feel for these bounds, consider first that

$$||M_k^{k-1}|| \leq ||S_k||^{j_1+j_2}||I_k - I_{k-1}^k(L^{k-1})^{-1}I_k^{k-1}L^k||. \tag{6.5.16}$$

Now the smoothing operator is defined in such as way as to have a bounded norm, in fact, usually $\rho(S_k) < 1$. Clearly it is the other term in (6.5.16) that should have a very small norm, depending on the smoothness of the coefficients of L, the fineness of the grid, the ratio of the two grid sizes, and the accuracy of the injection and interpolation operators.

As for A_k^{k+1} and A_{k+1}^k we have

$$||A_k^{k+1}|| \cdot ||A_{k+1}^k|| \leq ||S_{k+1}||^{j_1+j_2}||I_k^{k+1}||\,||I_{k+1}^k||\,||(L^k)^{-1}||\,||L^{k+1}||.$$

Clearly, the term $||(L^k)^{-1}||\,||L^{k+1}|| \approx 1$. The rest is bounded uniformly.

Following Hackbusch, [9], we make some additional observations on the bounds above. Using the Euclidean norm, we assume that

$$||L^K S_K^j|| \leq \nu(j)h^{-\alpha} \tag{6.5.17}$$

for some positive α. For fixed $j, \nu(j) = \nu$ obeys

$$1 \leq \nu \leq \nu_{max}(h)$$

with $\nu_{max}(h) = \infty$ or $\nu_{max}(h) \to \infty$ as $h \to 0$. Also $\nu(j) \downarrow 0$ as $j \to +\infty$.

These results are not hard to see. $||S_K|| = O(1)$ when L is sufficiently regular, while $||L^K|| = O(h^{-\alpha})$, as in the case of L being a differential operator of order $2m$, in which case $\alpha = 2m$. Note also that while the bound in (6.5.16) will decrease if $j \to \infty$, this would not be desirable due to the computational expense of having too many relaxation sweeps.

We also say that we have a good approximation capability if

$$||(L^K)^{-1} - I_{K-1}^K (L^{K-1})^{-1} I_K^{K-1}|| \leq Ch^\alpha, \tag{6.5.18}$$

which seems reasonable as eq. (6.5.18) is bounded from above by,

$$||(L^K)^{-1}|| + ||I_{K-1}^K|| \, ||(L^{K-1})^{-1}|| \, ||I_K^{K-1}|| \leq Ch^\alpha$$

as long as L^K is invertible on *all* grid levels, even allowing for injection and interpolation error. (A more careful analysis of such error along with round-off noise will be conducted in a later chapter.) Note that if the operator L is a regular and second-order, then $\alpha = 2$.

The reader can see that the above observations of Hackbusch are consistent with the requirements of the previous theorem. Namely,

$$||M_k^{k-1}|| \leq ||S_k^{j_2}|| \, ||(L^k)^{-1} - I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1}|| \, ||L^k S_k^{j-1}|| \leq Ch^\alpha h^{-\alpha} = C, \tag{6.5.19}$$

as required by (6.5.13).

Hackbusch generalizes this line of investigation to general domains and finite element systems. For details, see Hackbusch, [9].

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,

$$c = 2, \text{ for } k = 2, 3, ..., K-1, \tag{6.5.20}$$

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

$$c_k = \begin{cases} 1 & k \text{ odd} \\ 2 & k \text{ even} \end{cases} \tag{6.5.21}$$

which produces the MG algorithmic structure in fig (6.3.1).

Theorem: (S & T, [12]) For eq. (6.5.20) we have the following estimates if $4C\sigma \leq 1$:

$$||\mathbf{M}_K|| \leq \nu = \left(1 - \sqrt{1 - 4C\sigma}\right)/2C \leq 2\sigma, \quad K \geq 2, \tag{6.5.22}$$

and for eq. (6.5.21), if $4C^2(1+C)\sigma \leq 1$,

$$||\mathbf{M}_K|| \leq \begin{cases} \left(1 - \sqrt{1 - 4C^2(1+C)\sigma}\right)/2C^2 \leq 2\sigma(1+C) & (K \text{ even}) \\ \left(1 - 2C^2\sigma - \sqrt{1 - 4C^2(1+C)\sigma}\right)/2C^3 \leq \sigma(1+2C)/C & (K \text{ odd}) \end{cases} \tag{6.5.23}$$

Proof: The derivation of these equations require algebraic manipulations more complicated than insightful, so the reader is referred to Hackbusch, [9], for details. However, the less tighter but often quite adequate bounds are fairly easy to obtain.

In the first case, with $c = 2$, we have

$$\mathbf{M}_2 \leq \sigma \leq 2\sigma.$$

Then, assuming that $||\mathbf{M}_i|| \leq 2\sigma$ for $2 \leq i \leq k$, we have

$$||\mathbf{M}_{k+1}|| \leq \sigma + C(2\sigma)^2,$$

and if $4C\sigma < 1$,

$$\mathbf{M}_{k+1}|| \leq \sigma + \frac{1}{4\sigma}(2\sigma)^2 = 2\sigma.$$

Note that $0 < \sigma < 1/2$ implies the convergence of the $K-$level set-up.

The second case can be done in the same way. Again $||\mathbf{M}_2|| \leq \sigma$, and clearly,

$$\sigma \leq 2\sigma(1+C),$$

and so,

$$||\mathbf{M}_2|| \leq 2\sigma(1+C)$$

as expected.

Now assume the theorem is true up to k, and that $k+1$ is even. Then, using (6.5.23),

$$\begin{aligned} ||M_{k+1}|| &\leq \sigma + C(\sigma(1+2C)/C) \\ &= 2\sigma(1+C), \end{aligned}$$

which agrees with (6.5.23).

Now assume $k+1$ is odd. Then,

$$\begin{aligned} ||M_{k+1}|| &\leq \sigma + C(2\sigma(1+C))^2 \\ &= \sigma\big(1 + [4\sigma C^2(1+C)](1+C)/C\big) \end{aligned}$$

and using $4\sigma C^2(1+c) \leq 1$,

$$\begin{aligned} ||M_{k+1}|| &\leq \sigma(1 + (1+C)/C) \\ &= \sigma(1+2C)/C \end{aligned}$$

which agrees with (6.5.23). **Q.E.D.**

Remark 1: 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. (6.5.22). For example, if $C = 1$, then (6.5.22) yields,

$$\nu \leq 0.113 \text{ if } \sigma \leq 0.1$$

Typically, $C \geq 1$, but not very large. Hackbusch [9] treats an example of Poisson's equation using Jacobi sweeps for various values of j_1, j_2. In the Euclidean norm $C = 1$ and is independent of j_1 and j_2. For the spectral norm $||\cdot|| = \sqrt{\rho(AA^*)}$, we obtain $C \leq \sqrt{2}$ for all j_1, j_2 but $C \downarrow 1$ if $j_1 \to \infty$.

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 $c > 2$.

Remark 2: If we define c as in (6.5.21), the bounds in $||\mathrm{M}_K||$ do not fare as well as in the case of $c = 2$. As an example, let K be even. Then an upper bound of $||\mathrm{M}_K||$ tends to $\sigma(1+C)$ if σ is small enough (instead of $\nu = \sigma$ in the previous case). The trade-off is that there is less numerical work to perform than for $c = 2$.

Remark 3: One might be tempted to only use the V-cycle, i.e., to let $c = 1$. But our theorem yields no K-independent upper bound for $||\mathrm{M}_K||$ if $C \geq 1$. One way around this might be to set $c = 1$ for $K \geq k \geq K - k_0$ and $c = 2$ otherwise. For appropriate values of k_0, only a small increase in computational work would ensue, and our theorem could be adjusted to yield K-independent bounds of $||\mathrm{M}_K||$.

Remark 4: A remark on norms is in order. There are many reasonable possibilities, and different choices of norms will often lead to very different results. A general observation is that the spectral radius $\rho(\cdot)$ is less sensitive with respect to algorithmic changes than most of the other norms. As an example, norms can vary considerably depending in j_1 and j_2, the number of iterations of the first and last smoothing operators, (see (6.5.8),) whereas $\rho(\cdot)$ depends only on $j_1 + j_2$.

The operator norm $||\cdot||_S$ corresponding to the Euclidean inner product on G_h is the *spectral norm:*

$$||M||_S = \sqrt{\rho(MM^*)} \tag{6.5.24}$$

where M is any linear operator on G_h.

For positive-definite, symmetric operators L^h, the *energy norm* is mainly of theoretical interest:

$$||M||_E = ||(L^h)^{1/2} M (L^h)^{-1/2}||_S = \sqrt{\rho(L^h M (L^h)^{-1} M^*)}. \tag{6.5.25}$$

Even using different injection operators can have dramatic effects with different norms. If we convert from the Full Weighting (6.3.5) to the straight injection operator, $\rho(\mathrm{M}_k)$ will change very little, while,

$$||\mathrm{M}_k||_S = ||\mathrm{M}_k||_E = \infty \tag{6.5.26}$$

(See Stüben and Trottenberg [12] for an explanation of this.) The above behavior is typical of straight injection within Multigrid processes. Thus for *theoretical* purposes, when the above norms are most often used, straight injection is therefore useless, whereas in practice, it often performs better than Full Weighting operators. This perhaps demonstrates once again the somewhat inadequate estimates rendered by theoreticians, and the need for numerical experimentation, as testified to by the MG practitioners.

6. The Fokker-Planck Equation as a Model Problem

Virtually all Multigrid literature uses Poisson's equation on a rectangle as a model problem. The reason is due to its simplicity. By using a rectangular domain and reasonable data, the difficulties encountered in Poisson's equations, such as singularities in the spherical harmonics of gravitational and electrostatic phenomenon, can be avoided. The key observation we can make about Poisson's equation is that the matrix formed by standard finite difference schemes is symmetric, for most of the simplier relaxation methods. Because of this, an explicit calculation of the smoothing factor, as well as the spectral norm, is possible. But in practice, such symmetry will not be available. This will arise in our case because of the presence of the first partial in our PDE.

We propose a custom-made model problem for our purposes. It is the Fokker-Planck equation. We propose to study it by first examining a simpler case, and then building up the complexity until we have the general Fokker-Planck component of the Zakai equation. This approach will bring us into the heart of the Multigrid algorithm, and will be very instructive, as well as providing methods to estimate performance of the algorithm on a given problem.

Remark: While we will incorporate techniques used by all MG researchers, the investigation into this class of model problems is, to the best of the author's knowledge, original.

Our task then, is to calculate the smoothing factor and spectral norm of the multigrid operator in the presence of symmetry, using techniques based on Stüben and Trottenberg, [12], and then to calculate the change in the norm if the symmetry is "broken." This will all be done first with constant coefficients. For the case with variable coefficients, which is what we would be dealing with in general, we will use the techniques pioneered by Brandt, known as "local Fourier analysis." This approach is based on the observation that the Multigrid algorithm is "local" in the sense that operations at a point are performed as functions of the nearest neighbors at that point. This is clearly true for relaxation, injection and interpolation. Brandt argues that the way to find out how the Multigrid algorithm affects the value of the PDE solution at a certain point, is to "freeze" the value of the coefficients at that point and perform the kind of analysis described thus far for the simplified Multigrid operator. Analyzing the resulting behavior over the space of all possible points can provide information on the smoothing factor and the spectral radius of the MG operator.

To begin, we will investigate the use of the Multigrid Algorithm on the simplest version of the class of the Fokker-Planck equations, namely, the Heat equation:

$$\frac{\partial u}{\partial t} = \Delta u \text{ in } R^2 \quad u(0,x) = u_0(x) \tag{6.6.1}$$

We restrict our attention first to two dimensions for the sake of clarity, but extensions to higher dimensions will be possible.

Although the problem is defined on R^2, we assume for convenience that $u_0(x)$ is such that we can truncate the domain into the unit square, and that the solution $u(t,x) \leq \epsilon$ elsewhere. This information is provided by an application of the comparison theorem.

The application of the finite difference scheme described in chapter 5 yields

$$Lv(x) = \sum_{i=1}^{2} \frac{v(x + h_i r_i) - 2v(x) + v(x - h_i r_i)}{2h_i^2} \tag{6.6.2}$$

where $r_1 = i$ and $r_2 = j$, the standard unit normal vectors of R^2. If we number the points in the two-dimensional grid from left to right, there being N_1 points to a row, and N_2 points to a column, we have a matrix L_{ij} corresponding to (6.6.2) which has row vectors of the form:

$$[0,...0,1/2h_1^2,0,...,1/2h_2^2,-\frac{h_1^2+h_2^2}{h_1^2h_2^2},1/2h_2^2,0,...,0,1/2h_1^2,0,...0] \tag{6.6.3}$$

There are $n-1$ zero vectors between the $1/2h_i^2$ on the right and the left of the "$-\frac{h_1^2+h_2^2}{h_1^2h_2^2}$" entry. Note that had we had mixed partials, we would have nonzero entries adjacent to the now isolated "$1/2h_i^2$."

Now $A = -L^T$, which yields a matrix with row vectors that are the negative of what appears in (6.6.3). Thus our matrix $(I + \Delta tA)$ has row vectors of the form:

$$\Delta t[0,...,-1/2h_1^2,0,...,-1/2h_2^2,1/\Delta t+\frac{h_1^2+h_2^2}{h_1^2h_2^2},-1/2h_2^2,0,...,-1/2h_1^2,0,...0] \tag{6.6.4}$$

Recall that we are given $u_0(x)$. This too is formed into a vector. We thus have the sequence of vectors $\{u_n\}$ found by

$$(I + \Delta tA)u_{n+1} = \Psi_n u_n, \tag{6.6.5}$$

as was defined in the chapter on weak convergence. How will the Multigrid operator act on this system?

First of all, the spectral radius of the MG operator is determined entirely by the matrix $I + \Delta tA$, along with the choice of relaxation scheme. For purposes of illustration, we will employ the Jacobi relaxation scheme, defined as

$$S_h(\omega) = I_h - \omega(1+\Delta t\frac{h_1^2+h_2^2}{h_1^2h_2^2})^{-1}(I_h + \Delta tA) \tag{6.6.6}$$

Other relaxation schemes do not lend themselves so easily to analytical treatment. It would be helpful to have the eigenfunctions of $S_h(\omega)$. Now suppose that

$$S_h(\omega)v = \lambda v,$$

then

$$[I_h - \omega \frac{h_1^2 + h_2^2}{h_1^2 h_2^2}(I_h + \Delta t A)]v = \lambda v$$

or

$$(I + \Delta t A) = -1/(\omega \frac{h_1^2 + h_2^2}{h_1^2 h_2^2})(\lambda - 1)v, \tag{6.6.7}$$

thus the eigenfunctions of the two matrices are the same.

Claim: An examination of the matrix $(I + \Delta t A)$ reveals it to be symmetric, thus if $(x_1, x_2) = (n_1 h_1, n_2 h_2)$ is a point on the grid G_h, and if that point corresponds to the number "j", in our left-right numbering scheme, then the j^{th} component of the eigenvector of $I + \Delta t A$ is

$$k \sin(\frac{n_1 \pi}{2h_1} x_1) \sin(\frac{n_2 \pi}{2h_2} x_2) \tag{6.6.8}$$

Here $1 \leq n_1 \leq N_1, 1 \leq n_2 \leq N_2$, where $1/N_1 = h_2, 1/N_2 = h_2$. The coefficient "k" can be chosen so as to normalize the eigenvectors.

The key to understanding why (6.6.8) is true is by exploiting the symmetry of $I + \Delta t A$. This symmetry is a direct consequence of the fact that nearest neighbor points corresponding to D, E and B, C surrounding the point A are pairwise identical. Thus when we multiply by a vector whose components agree with (6.6.8), we have

$$\begin{aligned} & D \sin(\beta_1(x_1 - h_1)) \sin(\beta_2 x_2) \\ & + B \sin(\beta_1 x_1) \sin(\beta_2(x_2 - h_2)) \\ & + A \sin(\beta_1 x_1) \sin(\beta_2 x_2) \\ & + C \sin(\beta_1 x_1) \sin(\beta_2(x_2 + h_2)) \\ & + E \sin(\beta_1(x_1 + h_1)) \sin(\beta_2 x_2) \end{aligned} \tag{6.6.9}$$

Here $\beta_1 = n_1 \pi, \beta_2 = n_2 \pi$. Using trigonometric identities we obtain:

$$2D \cos(n_1 \pi h_1) + 2B \cos(n_2 \pi h_2) + A \tag{6.6.10}$$

Claim: We conclude from the preceding equation, that the eigenvalues of $S_h(\omega)$ are:

$$\chi_n = 1 - \omega(1 - \alpha \Delta t / h_1^2 \cos(n_1 \pi h_1) - \alpha \Delta t / h_2^2 \cos(n_2 \pi h_2)) \tag{6.6.11}$$

where

$$\alpha = (1 + \Delta t \frac{h_1^2 + h_2^2}{h_1^2 h_2^2})^{-1}$$

and $n = (n_1, n_2) \in Z^2$ and $|n| = \max\{n_1, n_2\}$.

To get a feel of this term, suppose $h_1 = h_2$, then

$$\alpha \Delta t/h^2 = \frac{\Delta t/h^2}{1 + 2\Delta t/h^2}$$

which we verify to obey, after simple manipulation,

$$0 < \frac{1}{h^2/\Delta t + 2} < 1/2 \tag{6.6.12}$$

Now we make use of a theorem cited in Young, [14].

Theorem: (Young, [14]) If the Jacobi method with $\omega = 1$ converges, then the Jacobi over-relaxation method will work for $0 < \omega \leq 1$.

We verify that, in our case, if $\omega = 1$, we have

$$\chi_n = 1 - (1 - \alpha \Delta t/h_1^2 \cos(n_1 \pi h_1) - \alpha \Delta t/h_2^2 \cos(n_2 \pi h_2)) \tag{6.6.13}$$

It is clear that is $h_1 = h_2 \approx 0$ that we have

$$2\alpha \Delta t/h^2 < 1.$$

The same is clearly true in the more general case of $h_1 \neq h_2$.

Now note that, because of (6.6.12), there exists no n_i such that

$$\alpha \Delta t/h_i^2 \cos(n_i \pi h_i) = 1/2,$$

for if there were, we could use it to find the maximal eigenvalue χ_n so that we would have, upon using these values of n_i,

$$\chi_n \approx 1 - \omega(1 - 1/2 - 1/2) = 1.$$

But this does not happen. Optimizing (6.6.13) means we must take the gradient.

Claim: Taking the gradient of χ_n with respect to n_1, n_2, we find that the smallest values of these terms will supply the maximal eigenvalue. Thus,

$$\rho(S_h) = |\chi_{1,1}| = 1 - \omega(1 - \alpha\Delta t/h_1^2 \cos(\pi h_1) - \alpha\Delta t/h_2^2 \cos(\pi h_2) \qquad (6.6.14)$$

We also find that the optimal value of ω *with respect to convergence rate* is $\omega = 1$.

As an example, suppose that $h_1 = h_2$, with $\omega = 1$. Then

$$\chi_{1,1} = 2\alpha\Delta t/h^2.$$

If $h^2 \approx \Delta t$, then $\chi_{1,1} \approx 2/3$, and if $h \approx \Delta t$, then $\chi_{1,1} \approx 2/(\Delta t + 2) \approx 1$. While it is more likely that $h^2 \approx \Delta t$, we can see that the Jacobian method has poor convergence.

The situation is quite different in regard to the smoothing properties of the Jacobian operator. This in fact, is the more important aspect of the relaxation method, as we never allow it to converge anyway.

The basic idea of smoothing in the context of the Multigrid method is to decrease the amplitude of the highly oscillatory portion of the residual. In effect, the relaxation method is used as a low pass filter prior to transferring the equation to a coarser grid, where highly oscillatory functions would be poorly represented. Furthermore, a better choice of ω will improve this smoothing property immensely.

To explain more fully, we expand the errors before and after one relaxation sweep, to yield

$$v_h = u_h - w_h, \qquad \bar{v}_h = u_h - \bar{w}_h, \qquad (6.6.15)$$

into discrete eigenfunction series:

$$v_h = \sum_{|n| \le N-1}, \qquad \bar{v}_h = \sum_{|n| \le N-1} \chi_n \alpha_n \phi_n \qquad (6.6.16)$$

The smoothing properties of $S_h(\omega)$ are measured by distinguishing *low* and *high* frequencies (with respect to the coarser grid G_{2h} used.) We define

$$\begin{aligned} &\text{low frequencies: } \phi_n \text{ with } |n| < N/2 \\ &\text{high frequencies: } \phi_n \text{ with } n/2 \le |n| \le N-1 \end{aligned} \qquad (6.6.17)$$

Define the *smoothing factor* $\mu(h,\omega)$ of $S_h(\omega)$ (and its supremum $\mu^*(\omega)$ over h) as the worst factor by which high frequency error components are reduced per relaxation step. That is,

$$\begin{aligned} \mu(h,\omega) &= \max\{|\chi_n| : N/2 \le |n| \le N-1\} \\ \mu^*(\omega) &= \sup_h\{\mu(h,\omega)\} \end{aligned} \tag{6.6.18}$$

Claim: Examining (6.6.18) we find that in our case

$$\begin{aligned} \mu(h,\omega) = \max\{&|1-\omega|, |1-\omega(1+\alpha\Delta t/h^2\cos(\pi h)), \\ &|1-\omega(1+\alpha\Delta t/h^2\cos(\pi h)|\} \end{aligned} \tag{6.6.19}$$

The result is obtained by determining the greatest possible values of χ_n where $n/2 \le |n| \le N-1$. This will clearly occur at four possible values of n, namely

$$n = \begin{cases} (N/2, N/2) \\ N/2, n-1) \\ (N-1, N/2) \\ (N-1, N-1) \end{cases}$$

We see that for $n = (N/2, N/2)$ we have

$$|\chi_{N/2,N/2}| = |1-\omega|$$

and for $n = (n/2, N-1)$ we obtain

$$|\chi_{N/2,N-1}| = |1-\omega(1+\alpha\Delta t/h^2\cos(\pi h))|$$

and for $n = (N-1, N-1)$ we have

$$|\chi_{N-1,N-1}| = |1-\omega(1+2\alpha|dt/h^2\cos(\pi h)|$$

Also, by taking the supremum over small h, we have

$$\mu^*(\omega) = \max\{|1-\omega|, |1-\omega(1+\alpha\Delta t/h^2)|, |1-\omega(1+2\alpha\Delta t/h^2)|\} \tag{6.6.20}$$

We remark that if $\omega, x > 0$, then

$$|1-\omega(1+x)| < |1-\omega(1+2x)|$$

only when

$$\frac{1}{1+\frac{3}{2}x} < \omega$$

as can be demonstrated by algebraic manipulation. Here $x = \alpha\Delta t/h^2$.

Now we find $\mu^*(\omega)$ for some sample values. As a demonstration, we assume the $h^2 \sim \Delta t$, so that

$$\frac{\Delta t/h^2}{1+2\Delta t/h^2} \approx 1/3.$$

Some sample values of $\mu^*(\omega)$ for different values of ω is shown in Table (6.6.1).

ω	$\mu^*(\omega)$
0	1
1/4	3/4
1/2	1/2
3/4	1/4
1	$2\alpha\Delta t/h^2 \approx 2/3$

Table (6.6.1)

Sample Values of $\mu^*(\omega)$

Now we ask: what will happen if we examine equation

$$\frac{\partial u}{\partial t} = a^2\Delta u \tag{6.6.21}$$

where $0 < \epsilon < a^2$? Note that we still have symmetry in our relaxation and difference operators. We see that we have a change in the eigenvalues of $S_h(\omega)$ as given by

$$\chi_n(a) = 1 - \omega(1 - \alpha a^2\Delta t/h_1^2\cos(n_1\pi h_1) - \alpha a^2\Delta t/h_2^2\cos(n_2\pi h_2)) \tag{6.6.22}$$

where we have a new value of

$$\alpha(a) = (1 + \Delta t a^2(\frac{h_1^2+h_2^2}{h_1^2h_2^2}))^{-1} \tag{6.6.23}$$

The same results would hold as before except that we would have to recalculate using $\alpha(a)$ as Δt is replaced by $a^2\Delta t$. Taking the derivative of $\frac{a^2\Delta t/h^2}{1+2a^2\Delta t/h^2}$ with respect to a^2 we obtain

$$\frac{\Delta t/h^2}{1+2a^2\Delta t/h^2}$$

and conclude that the greatest changes occur when a^2 is small. For large a^2 we have virtually no change in our earlier results. Thus if $a^2 = \epsilon$ and $2\epsilon\Delta t/h^2 << 1$ then

$$\alpha a^2\Delta t/h^2 \approx \epsilon\Delta t/h^2$$

and our new smoothing factor has an optimal ω at $\omega = 1$ and would now be $2\epsilon\Delta t/h^2 = \mu^*(1)$. Of course, if ϵ is too small the equation becomes ill-defined.

Now we examine the equation

$$\frac{\partial u}{\partial t} = a^2\Delta u + cu \tag{6.6.24}$$

where $0 < \epsilon \leq a^2$ and a, c are independent, arbitrary parameters. We now have

$$\alpha(a,c) = (1 + \Delta t a^2\frac{h_1^2+h_2^2}{h_1^2h_2^2} + \Delta tc)^{-1}$$

and we make the

Claim: The eigenvalues of $S_h(\omega)$ are now,

$$\chi_n = 1 - \omega(1 - a^2\alpha(a,c)\Delta t/h_1^2\cos(n_1\pi h_1) - a^2\alpha(a,c)\Delta t/h_2^2\cos(n_2\pi h_20 \tag{6.6.25}$$

which follows from using the equations in (6.6.9).

Remark: We see at once that $\alpha(a,c)$ could be infinite if, (letting $h_1 = h_2 = h$),

$$1 + \Delta t(a^2/h^2 + c) = 0$$

which occurs if

$$-1/\Delta t - 2a^2/h^2 = c$$

However, in our case, c, which will later be thought of as a function of x, will be bounded as we assume that we are working on a compact domain, derived by asymptotic estimates of the solution of the Zakai equation. Thus Δt and h would have to be chosen to avoid such singularities. In the rest of our analysis, we will assume that such problems have been avoided by the careful choice of parameters.

Now we analyze what happens if we have

$$\frac{\partial u}{\partial t} = a^2 \Delta u + \sum_{i=1}^{2} b_i u_{x_i} + cu \tag{6.6.26}$$

Here the b_i are constants. In the finite difference scheme that we use for our problem, we have

$$\begin{aligned} & b_i \frac{v(x + h_i r_i) - v(x)}{h_i} \text{ if } b_i > 0 \\ & |b_i| \frac{v(x) - v(x - h_i r_i)}{h_i} \text{ if } b_i < 0 \end{aligned} \tag{6.6.27}$$

This gives row vector for $I + \Delta t A$ of the form

$$\Delta t[0, ..., -a^2/(2h_1^2), ... - a^2/(2h_2^2), 1/\Delta t + a^2 \frac{h_1^2 + h_2^2}{h_1^2 h_2^2} + c + |b_1|/h_1 + |b_2|/h_2,$$
$$- a^2/(2h_2^2) - |b_1|/h_1, 0, ..., -a^2/(2h_2^2) - |b_2|/h_2, 0, ...0]$$

and if one or both $b_i < 0$ we would have to switch the appropriate b_i/h_i to its counter point position of the opposite side of the diagonal.

Now write

$$(I + \Delta t A) = (I = \Delta t A)_{sym} + B \tag{6.6.28}$$

where B corresponds to the *off-diagonal* terms that are contributing to the breakdown of symmetry in the original matrix. Thus B is composed of terms of the the form $|b_i|/h_i$, and has zeroes down its diagonal. The matrix $(I + \Delta t A)$ still has $|b_i|/h_i$ terms in its diagonal.

Therefore our relaxation operator is

$$S_h(\omega) = I_h - \omega\alpha((I + \Delta t A)_{sym} + B) \tag{6.6.29}$$

where

$$\alpha = (1 + \Delta t(a^2 \frac{h_1^2 + h_2^2}{h_1^2 h_2^2} + |b_1|/h_1 + |b_2|/h_2 + c))^{-1} \tag{6.6.30}$$

We see that we have a problem where we know the eigenvalues of one system:

$$Fv = \lambda v$$

and want to know the change in eigenvalues induced by perturbing F by $F + \Delta F$. In our case, we know (or can find) the eigenvalues associated with

$$F = S_h(\omega) + \omega\alpha B$$

which is symmetric. What we want to know is the change in χ_n due to

$$\Delta F = -\omega\alpha B \tag{6.6.31}$$

We invoke a theorem of Faddeeva, [6], namely

Theorem: (Faddeeva, [6]) Let F be an $n \times n$ matrix with

$$\begin{aligned} Fv_i = \lambda v_i = 0, \qquad & ||v_i|| = 1 \\ F^T u_i - \mu_i u_i = 0, \qquad & (u_i, v_i) = 1 \end{aligned} \tag{6.6.32}$$

Then

$$\Delta\lambda_i = \mu_i^T \Delta F v_i$$

and hence

$$|\Delta\lambda_i| \leq |||DF||_\infty$$

Claim: Now in our case F is symmetric so $v_i = u_i$. Furthermore,

$$\begin{aligned} ||\Delta F||_\infty &= ||\omega\alpha B||_\infty \\ &= \omega\alpha 2\Delta t \sum_{i=1}^{2} |b_i|/h_i \end{aligned} \tag{6.6.33}$$

Thus we see that as a worse possible case, that we might have $|b_i|/h_i$ so large that $||\Delta F||_\infty \approx \omega$, further restricting our choice of ω to insure convergence and a good smoothing factor. But in general, we would probably have

$$\omega\alpha||B||_\infty << 1$$

and so will produce a negligible difference in the original norm or smoothing factor.

Thus we could find the smoothing factor as a function of ω for the case where $B = 0$, (but with the b_i terms included in the diagonal portion of $(I + \Delta t A)$. and then conduct a worst case analysis of the form

$$\mu^*(\omega) + \omega\alpha||B||_\infty$$

Example

$$h_i = 10^{-1}, \Delta t = 10^{-3}, b_1 = 100, b_2 = 10, a = 10, c = 10$$

the $\alpha||B||_\infty \approx 1/4$. Of course, this is a worst case analysis, and almost certainly too conservative an estimate.

The Effect of Coarse-Grid Correction

Now we can begin to estimate the norm of the two grid Multigrid operator. We introduce the coarse-grid correction operator:

$$K_h^{2h} = I_h - I_{2h}^h (L^{2h})^{-1} I_h^{2h} L^h \tag{6.6.34}$$

and note the

$$\mathrm{M}_2 = S^{j_2}(\omega) K_h^{2h} S^{j_1}(\omega).$$

We quickly review some results of Stüben and Trottenberg, [12]. It turns out that the (at most) four subspaces of G_h:

$$E_{h,n} = span\{\phi_{n_1,n_2}, \phi_{N-n_1,n-2}, -\phi_{N-n_1,N-n_2}, -\phi_{n_1,N-n_2}\} \quad (|n| \le N/2) \tag{6.6.35}$$

are invariant under K_h^{2h}, i.e.,

$$K_h^{2h} : E_{h,n} \to E_{h,n}, \qquad |n| \le N/2$$

Consequently, as $\{\phi_n\}, |n| \le N-1)$ forms an orthonormal basis of G_h, K_h^{2h} is equivalent to a block-diagonal matrix composed of most 4×4 submatrices. Stüben and Trottenberg provide a detailed description of these matrices.

Stüben and Trottenberg (hereafter denoted S & T) derived their results by working on the Poisson equation. By observing their work, we make our own

Observation: Equations of the form

$$\frac{\partial u}{\partial t} = a^2 \Delta u + cu \quad \text{on the rectangle} \tag{6.6.36}$$

where $a^2 \geq \epsilon^2 > 0$ provide matrices K_h^{2h} of the same structure. The reason is due once again to the symmetry provided by our natural ordering, and the fact that $\phi_n(x) = k \sin(n_1 \pi x_1) \sin(n_2 \pi x_2)$ are still eigenfunctions of the matrix $I + \Delta t A$ formed from (6.6.36). Indeed, we are mostly concerned with the "lower" frequency eigenvalues, i.e.,$(\{\phi_n\}, |n| < N/2)$ and L^h is designed do that these very functions are also eigenfunctions of *both* L^{2h} and L^h. This restricted class of functions enjoys some special properties under K_h^{2h}.

For example, we have after S & T, we define

$$\Phi_n(x) : k \sin(n_1 \pi x_1) \sin(n_2 \pi x_2) \quad (x \in G_{2h}, |n| \leq N/2 - 1)$$

On G_2h, the $\Phi_n(x)$ and the basis functions of $E_{h,n}$ coincide, i.e., the basis functions of $E_{h,n}$ as written in (6.6.35) are all equal at the grid points and so we set them equal to Φ_n.

$$I_{2h}^h : span\{\Phi_n\} \rightarrow E_{h,n}$$

$$I_h^{2h} : E_{h,n} \rightarrow span\{\Phi_n\}, |n| \leq N/2 - 1$$

$$I_h^{2h} \phi_n = (n_1 = N/2 \text{ and/or } n_2 = N/2)$$

For example, we have

$$I_h^{2h} \phi_{n_1,n_2} = (1 - \gamma)(1 - \varsigma) \Phi_{n_1,n_2}$$

for $|n| < N/2$, which precisely correspond to the lower frequency eigenfunctions. Here, $\gamma = \sin^2(n_1\pi/2N)$, and $\varsigma = \sin^2(n_2\pi/2N)$.

Thus we

Claim: $K_h^{2h}\phi_n$ is the same for *all* operators L^h where

1). $\{\phi_n\}$ are eigenfunctions of L^h.

2). $\{\phi_n\}$ $|n| < N/2$ are eigenfunctions of *both* L^h and L^{2h}.

This follows since $L^h\lambda_n\phi_n$ and

$$I_h^{2h}\phi_n = (1-\mu)(1-\varsigma)\lambda_n\phi_n$$

$$(L^{2h})^{-1}(1-\mu)(1-\varsigma)\lambda_n\phi_n = (1-\mu)(1-\varsigma)\phi_n$$

for $|n| < N/2$ since, under these conditions, $(L^{2h})^{-1}L^h\phi_n = \phi_n$.

These are precisely the properties of our own $L^h = I + \Delta tA$ *in the absence of* b_i.

S & T calculated K_h^{2h} to be equivalent to a block diagonal matrix with laborious but straightforward methods to compute its entries.

It follows that for L^h with the aforementioned properties, K_h^{2h} can be calculated completely. Thus, we need only determine the change in $\rho(K_h^{2h})$ when we "break" the symmetry by introducing the first partials. As before we write,

$$\frac{\partial u}{\partial t} = a^2\Delta u + \sum_i b_i u_{x_i} + cu \tag{6.6.37}$$

and this yields an L^h of the form:

$$(I + \Delta tA) = (I + \Delta tA)_{sym} + (I + \Delta tA)_{off-diag} \tag{6.6.38}$$

or

$$L^h = L^h_{sym} + \bar{L}^h$$

where, as before, $\bar{L}^h$ has only the off-diagonal terms contributed by $b_i u_{x_i}$, (it has zeroes down its main diagonal). The b_i terms reappear in the L^h_{sym} diagonal entries.

Thus we have

$$K_h^{2h} = I_h - I_{2h}^h (L_{sym}^{2h} + L^{\bar{2}h})^{-1} I_h^{2h} (L_{sym}^h + L^{\bar{h}})$$

Now writing,

$$(L_{sym}^{2h} + L^{\bar{2}h})^{-1} = (L_{sym}^{2h}(I + (L_{sym}^{2h})^{-1} L^{\bar{2}h}))^{-1} \tag{6.6.39}$$

Claim: The above expression can be approximated,

$$(I - (L_{sym}^{2h})^{-1} L^{\bar{2}h})(L_{sym}^{2h})^{-1}. \tag{6.6.40}$$

This follows since $||(L_{sym}^{2h})^{-1} L^{\bar{2}h}|| < 1$ due to the fact that the matrix $I + \Delta t A$ has strong diagonal dominance.

Claim: Replacing (6.6.40) into (6.6.39) we get

$$\begin{aligned} \rho(K_h^{2h}) \leq \rho(K_{h,sym}^{2h}) & \\ + ||I_{2h}^h (L_{sym}^{2h})^{-1} L^{\bar{2}h} (L_{sym}^{2h})^{-1} I_h^{2h} (L_{sym}^{2h} + L^{\bar{2}h})|| & \\ + ||I_{2h}^h (L_{sym}^{2h})^{-1} I_h^{2h} L^{\bar{h}}|| & \end{aligned} \tag{6.6.41}$$

An immediate observation we can make is that the term on the right hand side is $O(h)$.

To see this we give the worst case estimate of the middle term of (6.6.41):

$$\begin{aligned} &||I_{2h}^h||\,||I_h^{2h}||\,||(L_{sym}^{2h})^{-1}||^2\,||L^{\bar{2}h}||\,||(L_{sym}^{2h} + L^{2h}|| \\ &\leq ||I_{2h}^h||\,||I_h^{2h}||C_1 h^4 C_2/h C_3/h^2 \end{aligned} \tag{6.6.42}$$

Now S & T give $||I_{2h}^h||\,||I_h^{2h}|| \sim 1$ while $h^4 \cdot \frac{1}{h} \cdot \frac{1}{h^2} = h$ and

$$\frac{||L^{\bar{2}h}||}{||L_{sym}^{2h}||} \cdot \frac{||L_{sym}^{2h} + L^{\bar{2}h}||}{||L_{sym}^{2h}||} \sim 1$$

This follows from the fact that the b_i are also in the diagonal portion of L_{sym}^{2h}.

Similarly, for the last term, we have

$$||I_{2h}^h||\,||I_h^{2h}||\,||(L_{sym}^{2h})^{-1}||\,||L^{\bar{h}}|| \sim 1$$

Thus

$$\rho(K_h^{2h}) \leq \rho(K_{h,sym}^{2h}) + Ch \tag{6.6.43}$$

where $C \sim 1$, and this is a worst case estimate.

Application of Local Fourier Analysis

The analysis we have done thus far is applicable to Fokker-Planck equations of constant coefficients. that will decidedly not be the case for the Zakai equation, so we should have a way of incorporating this fact into our model problem analysis.

Brandt [4] derived local Fourier analysis just for such purposes. The idea is to be able to make estimates of μ^* and ρ^* by by exploiting the factor that we can do so for the constant coefficient case. Fourier analysis will be applicable since we are concerned with effects of or operator on low and high frequencies of the residual. And it is a local analysis in the the following sense: the effect of the Multigrid operator on a certain point is largely a local process, in that relaxation, interpolation, and injection are functions of nearest neighbor values. Brandt argues that the way to determine the effect of the MG operator on a point is to "freeze" the coefficient values at that point and then to perform an analysis similar to what we have done thus far. Then we can maximize this estimate as a function of the coefficient value which are in turn dependent on x.

The formal way to use local Fourier analysis is by choosing a point $x_0 \in G_h$ and form $\mathcal{L}^h$, the operator corresponding to L^h, but in which the coefficients of our PDE have values taken at x_0. Thus, if we have

$$\frac{\partial u}{\partial t} = a(x)\Delta u + c(x)u, \qquad x \in \Omega \tag{6.6.44}$$

and form $(I + \Delta t A)$ from this, we would need $(I + \Delta t A)|_{x_0}$ as the starting point in our analysis.

The resulting operator $\mathcal{L}^h$ is defined on the *infinite* grid

$$G_h = \{x = \kappa \cdot h : \kappa \in Z^2\}$$

where $\kappa \cdot h = (\kappa_1 h_1, \kappa_2 h_2)$. As the domain of $\mathcal{L}^h$ we consider the (complex) linear space $\mathcal{E}_h$ spanned by the frequencies

$$\phi(\theta, x) = e^{i\theta x/h} \quad (x \in G_h) \text{ for } \theta \in R^2, -\pi < \theta \leq \pi \tag{6.6.45}$$

Here, $x/h = (x_1/h_1, x_2/h_2)$.

Note that $\mathcal{E}_h$ is a space with a non-denumerable basis. This is the main formal differ3nce compared with our previous model problem analysis. The introduction of continuous θ allows for certain technical simplifications. The use of an infinite grid allow us to ignore boundary conditions, so as to treat them separately.

However, we still need symmetry in the L^h or $\mathcal{L}^h$ operator, so we still have to estimate changes in the maximal eigenvalue due to the symmetry breaking caused by the addition of first partials.

Under these circumstances it can be shown, (see Brandt, [4]) that once we have frozen coefficients,

$$\rho(\mathrm{M}_2) \text{ on } \mathcal{L}^h \approx \rho(\mathrm{M}_2) \text{ on } L^h$$

where the first estimate was taken by Fourier analysis and the second was found by the methods of the preceding pages.

Thus, in the case of variable coefficients,

$$\rho(\mathrm{M}_2) \leq \sup_{x \in G_h} \rho((x, \mathrm{M}_2) \tag{6.6.46}$$

where, for each given $x \in G_h$, we freeze coefficients and perform our usual analysis, and then take the supremum over all possible x.

Obviously, such an optimization problem should be done by computer programs, and fortunately, such software exists and is part of a larger package recommended by Brandt, [4]. Such packages will be discussed later. Also, the optimization need only be done for $x \in G$, our compact domain constructed by asymptotic estimates of the solutions of the Zakai equation. Optimizing over all of R^n will give results far less tight than desired.

The key result of this section is that "breaking the symmetry" of the finite difference scheme by introducing first partials perturbed the spectral norm only slightly. We therefore conclude that the behavior of the Zakai equation under the Multigrid operator is very similar to Poisson's equation with smooth data on rectangular domains. Thus, in this general sense, we should be quite content with Poisson's equation as our first, approximate model problem.

We also remark on some observations made by Stüben and Trottenberg, [12]. Specifically, that for small values of j, where $j = j_1 + j_2$ and $S_h(\omega)^{j_1}$ and $S_h(\omega)^{j_2}$ are the first and last Jacobi sweeps of the Multigrid algorithm applied to Poisson's equation, they obtained $(\mu^*(\omega))^j \approx \rho(\mathrm{M}_2)$. (Typically, $1 \leq j \leq 4$.) When j gets too large, $(\mu^*(\omega))^j$ is no longer a good predictor of $\rho(\mathrm{M}_2)$: the high smoothing effect is not fully exploited las the reduction of low error frequencies by one coarse-grid step is not good enough, or the smoothing effect is even partly destroyed by the coarse-grid corrector, (which introduces new high frequencies by itself). Typically,

$$\rho(\mathrm{M}_2) \approx constant/j$$

But this does not imply that we should have j as large as possible, as it increases the numerical time-complexity of the program.

The similarity with our problem and that of the behavior of Poisson's equation leads us to expect the same kind of results.

For example, S & T found that for a proper choice of $\omega = 0.8$ for Poisson's equation, that, for $j = 3$,

$$\rho(\mathrm{M}_2) = 0.216 = (\mu^*(.8))^3$$

and for $j = 4$,

$$\rho(\mathrm{M}_2) = 0.137 \neq (\mu^*(.8))^4 = 0.130$$

While we can expect different values of ω and j, we can certainly expect to reach $\rho(\mathrm{M}_2) \approx 0.1$.

Claim: Using the theorems of S & T in the last section, we can expect to have $\sigma = \rho(M_2) = 0.1$ and $C \approx 1$, and thus

$$||M_K|| \leq \nu_K$$

where

$$\nu_2 = \sigma$$

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

Here, $c = 1, 2$ and is the number of inner iterations of the Multigrid cycle, with $c = 1$ giving the V-cycle and $c = 2$ giving the W-cycle. Thus we can analyze the behavior of the Multigrid algorithm on the Zakai equation with the two-grid case, and extrapolate convergence behavior on more general grid configurations.

Entensions to higher dimensions are possible as symmetry is still being preserved, and this, after all, is what we are exploiting when we make our estimates. However, the analysis becomes more cumbersome and less enlightening.

When it comes time to actually make estimates of the norms mentioned in this chapter, we would actually have to do is either one of two possibilities:

1). Actually construct the M_2 operator in matrix form, and use programs to find the maximal eigenvalue. This is costly but possible, but then one could use the previous theorem if estimates of $||M_k||$ are needed.

2). Apply the MG program to problems with known solutions, obtained, perhaps by another numerical procedure, and determine the number of iterations needed to reach desired accuracy. For example, say we already have the solution to a model problem. Then, beginning with an approximation, we could count the number of total MG cycles needed to reach the desired accuracy. Now in the case of time-dependent problems, our initial estimate is the value of the solution at the previous time-step, denoted as $u^h(t - \Delta t)$. Then, in the case of the Zakai equation, with $\Delta y(t) = 0$,

$$||u^h(t - \Delta t) - u^h(t)|| \leq ||L^h|| \, ||u^h(t)|| \Delta t,$$

and we also know,

$$||u_p^h(t) - u^h(t)|| \leq ||\mathbf{M}_2||^p ||u^h(t - \Delta t) - u^h(t)||. \tag{6.6.47}$$

Of course, the left hand side of (6.6.47) is a known number, as $u^h(t)$ is known from another source, and we also know the number of cycles, p. Now replace $||u^h(t)||$ by $\max_{0 \leq t \leq T} ||U(t,x)|| = ||U_{max}||$ which can be found by using maximum theorems for parabolic PDEs similar to those discussed in chapters 3-4. We can now obtain an estimate of $||\mathbf{M}_2||$,

$$||\mathbf{M}_2|| \approx \Big(\frac{||u_p^h(t) - u^h(t)||}{||L^h||\,||U_{max}||\,\Delta t} \Big)^{1/p} \tag{6.6.48}$$

Once again, the MG practitioner must be prepared to use numerical experimentation, in addition to formal methods, to determine the design of the program.

It should be clear by now that the Zakai equation is an excellent candidate for Multigrid treatment. For despite subtle variations in the finite difference scheme, due to weak convergence requirements, we still have an implicit, and hence stable discretization of the PDE. This stability guarantees that we will have no problems in transferring to coarser grids.

Boundary conditions are, at least for now, no problem either. For example, in the case of n space variables, we have truncated the prior density, which is, say, a n-dimensional Gaussian, and formed a rectangular domain with zero boundary condition. Thus for each time-step, we must solve an equation of the form:

$$L^h U^h = f^h \tag{6.6.49}$$

where f^h is a function of the value of the value of U at the previous time-step.

Issues involving the choice of relaxation schemes, interpolation, injection, etc, will be discussed in a later chapter. Suffice it to say for now that the structure of the matrix formed by the implicit scheme lends itself naturally to a proper choice among the available methods for relaxation, as well as the other operations we have mentioned. This will be made clear in our chapter on relaxation methods.

We might also remark that since the linear system of our problem is of the form

$$(I + \Delta t A)u^{n+1} = BY_{n+1}u^n, \tag{6.6.50}$$

where the matrix $(I - \Delta t A)$ is a constant for each time-step, requires no set-up time and can be stored in the computing system in advance. The matrix BY_{n+1} is diagonal, (see chapter 5 for details), and hence the right hand side of (6.6.50) can be calculated in parallel, in a time that is virtually trivial compared with the rest of the computation.

In the performance analysis of the MG algorithm, we have already noticed that having a model problem is very important, in order to make predictions concerning convergence rates, computing times and so on. As demonstrated in this chapter, a natural model problem in our case is the Fokker-Planck component of the Zakai equation, obtained by setting $dy_t = 0$. This yields,

$$U_t = a(x)U_{xx} + b(x)U_x + c(x)U. \tag{6.6.51}$$

Note that the coefficients are not time-dependent, so at each time-step, the matrix L^h in (6.6.49) will be the same, albeit too complicated for formal analysis, as compared with Poisson's equation.

Numerical studies of this model problem could yield much of the same type of information that is provided by examples found in Hackbusch and Trottenberg, [8]. This includes estimates of the Multigrid operator's spectral norm, smoothing factors for a variety of relaxation schemes, computational intensity and perhaps other features. Such estimates are best made with numerical simulations, and package programs are available for just this purpose, as will be described in the sequel.

Finally, we note that as the matrix $(I + \Delta t A)$ is a constant, the Multigrid operator is unchanged by changes in y_t. Only the quality of the initial approximations determines the number of iterations. But (6.6.2) gives us just this estimate. Thus a convergence test, which would have to be done globally and could well take up

all of our allotted time for computation, need not be performed. It follows that *all program parameters are precomputable.*

7. Multigrid Applications to Time-Dependent Problems

This section is a brief remark on some observations of Brandt, [4].

From our point of view, it is disappointing to find that Multigrid experimentation with time evolution problems is rather sparse, but enough has been done to draw some conclusions. Brandt offers an interesting twist to the use of MG techniques for solving parabolic equations, and he calls it the *frozen τ method.*

We define what amounts to a *fine to coarse defect equation*, i.e., a correction to the coarse-grid equation designed to make its solution coincide with the fine-grid solution. Thus on grid G_{2h} we have

$$L^{2h}u^{2h} = f^{2h} + \tau_h^{2h}, \tag{6.7.1}$$

where

$$\tau_h^{2h} = L^{2h}I_h^{2h}u^h - I_h^{2h}L^h u^h. \tag{6.7.2}$$

We can now reverse our earlier viewpoint. Instead of regarding the coarse grid as a device for calculating the correction

$$u^{2h} = I_h^{2h}u^h,$$

to the fine grid solution, we can view the fine grid G_h as a device for calculating the correction τ_h^{2h} to the coarse-grid equation, and we call τ_h^{2h} the fine to coarse correction function, making it a kind of defect correction. The important observation here is that this particular defect correction depends essentially on high-frequency components.

In the context of the "frozen τ method," the τ_h^{2h} plays an important role in solving time-dependent problems. For each new system generated by a time-step, we use not only the solution at the previous time-step as our first approximation,

but we also employ the values of the previous fine to coarse correction functions, in the following way. Even before we have begun our computations on G_h, we transfer τ_h^{2h} from the previous problem to G_{2h} and use

$$L^{2h}u^{2h} = f^{2h} + \tau_h^{2h}, \tag{6.7.3}$$

instead of

$$L^{2h}u^{2h} = f^{2h}. \tag{6.7.4}$$

Then the error in u^{2h} will not depend on the high-frequency components, (as it would in solving (6.7.3)), but only in the *changes* in those components from their values in the previous problem.

Now it turns out that τ_h^{2h} also represents the local truncation error on mesh size $2h$ relative to mesh size h. More precisely, $\tau_h^{2h} \approx \tau^{2h} - \tau^h$ where

$$\tau^{2h} = L^{2h}U - LU. \tag{6.7.5}$$

Thus when τ_h^{2h} is reasonably small, we can arrange the MG algorithm to stop returning to the finer grid and simply concentrate more of the work on the coarser grids, there being little to be gained otherwise. This is especially useful in time-dependent problems where high-frequency error components begin to settle down quickly, as in heat equations with stationary solutions, or possibly in our case, when the conditional density is better able to track the state x_t after a sufficiently large σ field of observations has been constructed.

8. General Remarks

Remark: Brandt [4] makes some rather critical observations of the rigorous analysis usually performed by theoreticians. A cursory examination of such papers, (at least up to 1982), show that MG methods can solve, for a large and practical class of problems, the associated algebraic system of n equations, (n unknowns on the finest grid), to the level of the truncation error in less than Cn computer operations. Unfortunately, such statements are of little help to the practitioner since often no attempt is made to discern the nature of "C," or when such an attempt is made, it is often unrealistic. In typical cases the theoretical bound is $C \approx 10^8$, while in practice ti is closer to 10^2 and this is still *very* conservative, for in the next chapter we will examine some improvements in the complexity analysis and discover a much better bound. However, the very best bounds can be found only for the simplest cases, (equations with constant coefficients on a rectangular domain), and then only after a rather lengthly Fourier analysis.

Until more reliable theoretical techniques are available, MG practitioners will have to rely on model problem experimentation.

Remark: A number of multigrid software packages have emerged, and we will discuss some numerical experiments with them in a later chapter. These include *MG*00 programs [7] that are restricted to rectangular domains with different types of boundary conditions. There is a sample program listed in the Appendix of Stüben and Trottenberg, [8], designed for Poisson's equation with Dirichlet boundary conditions in the unit square–a very specialized version of *MG*00.

The *MG*01 programs [7] refer to second order equations with Dirichlet boundary conditions on non-rectangular bounded domains. These programs were designed with the intent of investigating the effect of changes in domain shape relative to multigrid performance–rather than on optimizing efficiency.

A very robust and efficient multigrid program is described by Wesseling [13], and denoted *MGD*1. This program is perceived by the user as any other linear

systems solver, and requires no insight in the properties of the multigrid method. It design features are very robust to even rather dramatic changes in operator coefficients.

More remarks on software packages will be made in chapter 9.

9. Conclusion

This chapter has been only a brief introduction to the basic features of Multigrid theory. Issues involving complexity and parallel implementation will be covered in subsequent chapters. Original results on the use of the Fokker-Planck equation as a model problem were also covered.

We began with a simple explanation of the MG algorithm and expanded it to include the general case. Related concepts such as relaxation schemes for smoothing error prior to intergrid transfer, the recursive structure of Multigrid grid, convergence behavior, and the relevance to time-dependent problems were covered.

The Multigrid algorithm was found to be similar to any other linear systems solver, where the system could be the implicit discretization of a linear PDE, either elliptic or parabolic. The program will converge to the level of the truncation error, i.e., if U is the true solution of the PDE, and U^h is the true solution to the discretized system, then the MG program will converge to u^h which obeys

$$||u^h - U^h|| \sim ||U^h - U||$$

in some appropriate norm. The only requirements are stability, which is guaranteed by the implicit scheme, and the existence of good smoothers, also guaranteed by invertible discrete representations of the PDE. The algorithm is highly efficient and will be shown to be imminently suitable for parallel implementation.

Deficiencies in the current Multigrid theory were touched upon, the that usually conservative estimates on convergence rates are possible. Some of these estimates were discussed. We concluded, along with the MG practitioners, that numerical ex-

perimentation will play a major role in the design and optimization of MG program parameters.

Much of this experimentation is performed on model problems which are representative of a class of problems. Any program designs can be made with the model problem, and robustness of the design, especially with respect to convergence can be demonstrated. This means that as the convergence rate of the Multigrid program is stable under small changes in the problem statement, we can pre-determine the number of program cycles needed for acceptable convergence and still be assured that this design feature be robust. Poisson's equation is the model problem of choice among the MG practitioners. The natural model problem for the Zakai equation is found by setting $dy_t = 0$ and analyzing only the Fokker-Planck component. We recommend this for future numerical studies, possible with some of the Multigrid package programs that were discussed. We also pointed out the strong similarities of our model problem and that of Poisson's equation. Convergence behavior with respect to $\Delta y(t)$ was shown to be robust, due to the constancy of the matrix in our linear system . This implies that *all program parameters are pre-computable, and will hold for all sample paths* y_t.

References for Chapter 6

[1] Bank, R. and Dupont, T., "An Optimal Order Process for Solving Finite Element Equations," *Mathematics of Computation*, vol. 36, No. 153, Jan. 1981.

[2] Brand, K., "Multigrid Bibliography," in Hackbusch and Trottenberg.

[3] Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation*, vol. 31, No. 138, 1977.

[4] ——, "Guide to Multigrid Development," in Hackbusch and Trottenberg.

[5] Chan, T. and Schreiber, R., "Parallel Networks for Multi-grid Algorithms: Architecture and Complexity," *SIAM J. Sci. Stat. Comput.*, vol. 6, No. 3, July, 1985.

[6] Fadeeva, P. K., *Computational Methods of Linear Algebra,* Freeman Pubs., p. 288, 1983.

[7] Foerster, H. and Witsch, K., "Multigrid Software for the Solution of Elliptic Problems on Rectangular Domains: *MG*00," in Hackbusch and Trottenberg.

[8] Hackbusch, W. and Trottenberg, U., eds., *Multigrid Methods,* Lecture Notes in Mathematics, Springer-Verlag, No. 960, 1982.

[9] Hackbusch, W., "Multigrid Convergence Theory," in Hackbusch and Trottenberg.

[10] Novak, Z., "Use of Multigrid Method for Laplacian Problems in Three Dimensions," in Hackbusch and Trottenberg.

[11] Ortega, J. and Voigt, R. "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Review*, vol. 27, No. 2, June, 1985.

[12] Stüben, K. and Trottenberg, U., "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications," in Hackbusch and Trottenberg.

[13] Wesseling, P., "A Robust and Efficient Multigrid Method," in Hackbusch and Trottenberg.

[14] Young, D., *Iterative Solution of Large Linear Systems*, Academic Pr. 1971.

7. The Complexity of the Multigrid Algorithm

1. Introduction

We saw at the end of chapter 2 how the one-dimensional Zakai equation, once approximated as an implicit finite difference scheme, which yields a linear system, can be solved directly in $O(n)$ time by a systolic array, using $O(n^2)$ processors. We also saw that if n changes, we can still keep the same architecture, although the number of processors will increase. However, as we are aiming for real-time processing, we need a computation speed for the calculation of the approximate conditional density less than one or .1 msec. for each sample of observational data. Now n is the width of the point-grid, so, for example, n^2 would be the number of grid points for the two-dimensional case. We can certainly imagine situations where the number n may grow fairly large, so even $O(n)$ may be too slow, and what will be even worse, systolic direct solvers have computation times that grow with the dimension of the problem. It follows that we must consider other computing strategies.

We have already been introduced to the Multigrid algorithm, and in this chapter we will examine its complexity and computing time properties. We will show, using methods of Chan and Schrieber, [3], that for a fully parallel multigrid algorithm can solve the linear system associated with the PDE in at most $O[(\log n)]$ time. This is derived by exploiting the recursive structure of the Multigrid algorithm. In principle, this computing time is *independent of dimension.* The increase in speed is achieved by a network similar to the systolic array in that it also enjoys regularity, local communication, and repetition of a single, simple processing element. The inevitable trade-off is in the form of a much greater tax on the available hardware; the array configuration is actually several interlaced processor grids, one on top of another, with somewhat more complicated inter-connections, so fabrication may be a problem. The more complicated interconnections also impose a

burden on computing speed. Thus we can expect a limit on the dimensional growth our problem can have due to real-time processing constraints. Also, complexity per unit processor must also increase, as will the number of these processors when the fineness of the grid, which is a function of n, is improved. So already we are far beyond the basic systolic array paradigm. We may also have to pay for the design with a loss of efficiency in that increases in the number of processors, (as may be required in the interests of accuracy or domain changes,) will not yield corresponding improvements in computation speed. Indeed, inefficiency, or perhaps more accurately, processor unemployment, which implies that processors will remain idle during much of the computation, is a chronic problem in Multigrid architectures. However, a method of circumventing this will be discussed, known as Concurrent Iteration, the trade-off being an increase in the complexity of the program, which in turn will tax the processor design. At any rate, as speed is our most important priority, this should not be too much of a problem.

Main Results: Application of complexity theory to our problem in the light of real-time processing constraint. Trade-off analysis of using various initial conditions to start the algorithm, an a critique of Concurrent Iteration.

2. Multigrid Algorithms

The reader is already assumed to have a familiarity of Multigrid theory, at least equivalent to that of the preceding chapter.

Our problem then, is the Zakai equation defined on $[0,T] \times \Omega$ where $\Omega \subset R^d$. We assume that Ω is a rectangle. The PDE is implicitly discretized to yield a linear system $L^K U^K = f^K$ defined on the K^{th} finest grid.

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 that 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 chapter, 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. Chan and Schreiber (whom we refer to as C & S, [3])

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. However, there does exist a design for which the efficiency falls off as $O[(\log n)^{-1}]$.

How important is efficiency to our problem? In general, increases in point grid size might be needed for enlargements of the domain or improvements in accuracy; this latter case is especially true for solutions which are known in advance to be highly oscillatory, or possess singularities or have irregular boundary conditions. Hence increasing the size of the point grid provides improves accuracy at the expense of poor speedup, not to mention increased hardware costs. Of course, most of the solutions to our problems will tend to be better behaved and we have already agreed that we can safely avoid irregular boundary conditions, at least for a large class of problems. As for accuracy, it is probably less important to us than it normally would be to a numerical analyst, as many estimator-detector problems would not require more than perhaps a few spaces past the decimal point. However, it will turn out that as our goal is to determine if a better, or at least equal speed can be achieved over that of the systolic direct solver's $O(n)$ time, then we may have to pay for it through a loss of efficiency. Fortunately, as indicated before, a speed of $O[(\log n)]$ is possible and may be worth the trade-off.

To return to the multigrid algorithm itself, we define, after Chan and Schreiber [3], BASICMG and FULLMG, where the former is a subroutine of the latter. The two algorithms differ in that BASICMG starts its computation on the finest grid and works its way down to the coarsest grid, while reducing error on a grid by a constant factor in optimal time, whereas FULLMG begins on the coarsest grid and works its way up to the finest grid, while reducing error to truncation error level in optimal time. The fundamental difference is that BASICMG begins with a given approximation, and then proceeds to smooth it and move the defect equation down the grid levels. FULLMG will immediately inject downward to the coarsest grid,

direct solve, interpolate upwards, and use this result as the first approximation to BASICMG. While FULLMG may certainly be useful in many applications, time-dependent problems offer a natural first choice as an initial approximation, namely the value of the solution at the previous time-step. However, we include FULLMG for the sake of completeness.

BASICMG is a recursive algorithm, although in practice it is usually implemented in a nonrecursive manner. It is not adaptive, in that iterations are not controlled by an examination of relative changes in the residuals, but instead are predetermined by parameters (c, j, m). The major computational work is in the smoothing sweeps (subroutine SMOOTH), which is usually composed of an implementation of successive-over-relaxation, Jacobi iteration, or the conjugate gradient method. For our purposes, these are among the only choices as such algorithms must be suitably "parallel". We will discuss how these architectures can be constructed later in the sequel.

We provide algorithms BASICMG and FULLMG below in Tables 7.2.1 and 7.2.2 respectively, written in a pidgin ALGOL. To demonstrate the mechanisms of BASICMG, consider the case of three grid levels and $c = 2$ cycles of iteration. On the third and finest grid we begin by smoothing the discretization error j times; here, $j \sim 3, 4$. This replaces our initial guess of the solution, which for us would have been the conditional density approximate at the immediately preceding time-step. We convert to the second grid while computing coarse grid correction $d^2 = L^2 v^2$, where $d^2 = f^2 - L^2 u^2$. Here the initial approximation to v^2 will be zero. Now we apply BASICMG but start on the second grid, using the defect equation as the new system. After smoothing we reduce to the coarsest grid, and directly solve the equation, and interpolate back to the second grid with a correction step. That is one inner cycle. We smooth again m times for coarse to fine grid correction; whereas we smooth j times for fine to coarse grid correction. Again we convert back to the coarse grid and direct solve, and go back to the second grid and smooth m times.

Our two inner cycles are over. Now interpolate back to the finest grid, smooth m times, and one total cycle of the algorithm is complete. This yields the W-cycle as discussed in the previous chapter.

The role of FULLMG is to interpolate approximate solutions on coarser grids as initial guesses for the BASICMG algorithm. This overcomes the frequent inability of BASICMG to reach truncation error accuracy in less than $O(n_K \log n_K)$ time, due to the poor quality of a general initial guess to the solution. Using the convergence results of BASICMG and the properties of the approximations of L^K, f^K, it will be shown that FULLMG computes a solution u^K with truncation error accuracy in $O(n_K)$ time. We have thus an interesting question of whether FULLMG, with its automatically generated initial approximation, could be better for some problems than using the value of the time-dependent solution at the previous time-step.

The accuracy and the convergence of the BASICMG algorithm obviously depends on three important factors: smoothing, coarse grid transfer, and fine grid correction. We require that smoothing sweeps annihilate the high frequency components of the error efficiently, that the coarse grid correction d^k be a good approximation to the fine grid error in the low frequency components, and the interpolation operators (I^k_{k-1}) be accurate enough. These conditions can be verified for our problem.

Algorithm BASICMG($k, u^k, c, j, m, L^k, f^k$)

(Computes an approximation u_k to U^k,

where $L^kU^k = f^k$,

given an initial guess u^k,

Returns the improved approximate solution in u^k.

Reduces initial error in u^k by a constant factor.)

If $k = 1$ then

Solve the problem using a direct method. Return solution u^k.

else

(Smoothing step (j sweeps):)

$u^k \leftarrow$ SMOOTH(j, u^k, L^k, f^k).

(Compute data d^{k-1} for coarse grid correction equation:)

$d^{k-1} = L^{k-1}v^{k-1} = I_k^{k-1}f^k - I_k^{k-1}(L^ku^k)$

(Solve coarse grid problem approximately by c cycles of

BASICMG:)

$v^{k-1} \Leftarrow 0$. (Initial Approximation)

Repeat c times:

BASICMG($k-1, v^{k-1}, c, j, m, L^{k-1}, d_{k-1}$)

(Correction step:)

$u^k \Leftarrow u^k + I_{k-1}^k v^{k-1}$

(Smoothing step (m sweeps):)

$u^k \Leftarrow$ SMOOTH(m, u^k, L^k, f^k).

End if

End BASICMG

Table 7.2.1.

3. Design of the Computing Network

We will now design a parallel machine capable of performing the multigrid algo-

Algorithm FULLMG$(k, u^k, r, c, j, m, L^k, f^k)$

(Computes an approximation u^k to U^k

where $L^k U^k = f^k$,

using r iterations of BASICMG,

using initial guess u^k from interpolating the approximate

solution obtained on the next coarser grid.

Solution obtained can be proven to have truncation error accuracy.)

If $k = 1$ then

Solve the problem using a direct method to get u^1.

else

(Obtain solution on next coarser grid:)

FULLMG$(k-1, u^{k-1}, r, c, j, m, L^{k-1}, f_{k-1})$.

(Interpolate u^{k-1}:)

$u^k \Leftarrow I_{k-1}^k u^{k-1}$.

(Reduce the error by iterating BASICMG r times:)

Repeat r times:

BASICMG$(k, u^k, c, j, m, L^k, f^k)$.

End if

End FULLMG

Table 7.2.2.

rithm. We assume our problem is in d dimensions over a rectangular domain using a regular point grid of n^d points, and that we have a system of point grids $\{G_k\}_{k=1}^K$ where G_k has $(n_k)^d$ grid points, mesh lengths h_{kj}, $1 \le j \le d$, where the finest grid has $n_K = n$ and

$$n_{k+1} = a(n_k + 1) - 1, \quad k = 1, 2, ..., K-1, \tag{7.3.1}$$

for some integer $a \ge 2$.

We introduce a standard multi-index notation for grid points and processors.

Let $\mathbf{z}^+_{n,s}$ be the set of s tuples of non-negative integers less than n, e.g., $(i_1, i_2, ..., i_s)$, where, $0 \leq i_j \leq n$ and $i_j \in Z$. Define the projection operator $\pi^s_r : \mathbf{z}^+_{n,r} \rightarrow \mathbf{z}^+_{n,s}$ for $r \geq s$ by,

$$\pi^s_r((i_1, i_2, ..., i_r)) = (i_1, i_2, ..., i_s). \tag{7.3.2}$$

We also require $\mathbf{i} = (i_1, i_2, ..., i_s) \in \mathbf{z}^+_{n,s}$ to have the norm $|\mathbf{i}| = i_1 + i_2 + ... + i_s$.

All gridpoints in G_k are labeled as points in $\mathbf{z}^+_{n_k,d}$ in such as way that label $\mathbf{i}$ has spatial coordinants $(i_1 h_{k1}, i_2 h_{k2}, ..., i_d h_{kd})$. Similarly, processors in processor grid P_k have indices from $\mathbf{z}^+_{n_k,\gamma}$. Processors $\mathbf{i}$ and $\mathbf{k}$ are connected if $|\mathbf{i} - \mathbf{k}| = 1$.

The complexity per unit processor will be much greater in this design than in the standard systolic array. For evidently, if each processor has $O(n^{d-\gamma})$ memory cells, we can store the solution, f^k, and $O(1)$ temporary values belonging to the whole of grid G_k in the processors P_k. For example, the value at gridpoint $\mathbf{i}$ can be stored in processor $\pi^\gamma_d(\mathbf{i})$ for $\mathbf{i} \in \mathbf{z}^+_{n,d}$.

Even larger problems can be handled, at the expense of increasing processor complexity. Suppose we have restricted ourselves to using only n^γ processors, but we have $(m \cdot n)^d$ grid points. Now assume that each processor can store all information associated with the $m^d n^{d-\gamma}$ grid points. We wish to map all grid points in such a way that neighboring grid points reside in the same or neighboring processors. This will be a prerequisite for efficient smoothing. To this end, define

$$\phi_m : \mathbf{z}^+_{mn} \rightarrow \mathbf{z}^+_n, \tag{7.3.3}$$

where

$$\text{for all } i, j, \mathbf{z}^+_{mn}, \quad |\phi_m(i) - \phi_m(j)| \leq |i - j|.$$

Let $j = qn + r$ where q and r are integers and $0 \leq r \leq n - 1$. Now let

$$f_m(j) \in \begin{cases} r & \text{if } q \text{ is even} \\ n - 1 - r & \text{if } q \text{ is odd} \end{cases} \tag{7.3.4}$$

A multidimensional structure can now be mapped onto a processor grid. Suppose the point grid has $m_1 n \times m_2 n \times \cdots \times m_d n$ points. Then point $\mathbf{i}$ can be stored

in processor $F_d(m_1, m_2, ..., m_d, n; \mathbf{i})$ where

$$F_d(\mathbf{i}) = (f_{m_1}(i_1), f_{m_2}(i_2), ..., f_{m_d}(i_d)). \tag{7.3.5}$$

If we have only n^γ processors then we can map $\mathbf{i}$ into $F_d^\gamma(\mathbf{i})$ where

$$F_d^\gamma(\mathbf{i}) = F_\gamma(\pi_d^\gamma(\mathbf{i})).$$

Thus this mapping preserves the Principle of Locality that we clearly wish to exploit in our design, as relaxation, injection and interpolation require only local communication between neighboring processors. Consequently, our definition of $W(n)$ must be changed to

$$W(mn) = m^d W(n). \tag{7.3.6}$$

Thus we have a factor of m^d fewer processors, the time needed for smoothing and intergrid operations increases by the same factor of m^d, so there is no corresponding loss in efficiency. As stated, it can be shown that this mapping preserves locality of point grids $\{G_k\}$ and processor grids $\{P_k\}$.

As a simple example, suppose we have $m = 2, m = 2$, and $\gamma = d = 2$, and a processor grid of four processors labeled as

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \tag{7.3.7}$$

Then a processor grid assignment of a sixteen point grid will be as follows:

$$\begin{matrix} 1 & 2 & 2 & 1 \\ 3 & 4 & 4 & 3 \\ 3 & 4 & 4 & 3 \\ 1 & 2 & 2 & 1 \end{matrix} \tag{7.3.8}$$

Note that each point has access to all other points by *local* interprocessor communication. Of course, another labeling scheme could be, "naive" design given below:

$$\begin{matrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{matrix} \tag{7.3.9}$$

So clearly there are many possibilities of grid point to processor assignments.

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,

$$S = tn_k^{d-\gamma}. \tag{7.3.10}$$

Obviously, it is to our advantage to conduct as few smoothing sweeps as necessary and still assure sufficient accuracy. How could we determine in advance what this number should be? By being able to measure the smoothing capabilities of the relaxation scheme. A means of doing this will be described in the next chapter.

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

1. First, j smoothing 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. BASICMG 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 interpolation: $I_{k-1}^k v^{k-1}$.

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

$$W(n) = (j+m+s)tn^{d-\gamma}, \tag{7.3.11}$$

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

$$T(an) = cT(n) + W(an), \tag{7.3.12}$$

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 $(I_{k-1}^k v^{k-1})$. To derive formula (7.3.12), one must only see that in Table 7.3.1, BASICMG makes use of self-reference, and this of course, corresponds to applying the algorithm to the next coarsest grid. Now imagine two grids, one with $(an)^d$ points and the other with n^d points. Now the time to perform step 1, 2, 4, 5, would be $W(an)$ and the time to perform step 3 would be $cT(n)$. The sum of these two would be the time to perform BASICMG as it is represented in the five steps above. This derives (7.3.12).

The architecture used to implement these operators determine the actual time needed; (more precisely, this means the dimensionality and the available number of processors on an n-grid will be the major factors). A generalization of (7.3.11) would be,

$$W(n) = (j + m + s)tg(n), \quad \text{where} \quad g(n) = n^p, \quad p = d - \gamma. \tag{7.3.13}$$

We can now state the following theorem, which can be readily proved by the reader.

Theorem 7.3.1: (C & S, [3]). Let $T_p(\cdot)$ be a particular solution of (7.3.12), i.e.,

$$T_p(an) = cT_p(n) + W(an). \tag{7.3.14}$$

Then the general solution of (7.3.12) is:

$$T(n) = \alpha n^{\log_a c} + T_p(n), \tag{7.3.15}$$

where α is an arbitrary constant.

Now we can solve for this particular solution given the special case of $W(n) = \beta n^p$, in which case:

$$T_p(n) = \begin{cases} \beta(a^p/(a^p - c))n^p & \text{if } p \neq \log_a c, \\ \beta n^p \log_a n & \text{if } p = \log_a c. \end{cases} \tag{7.3.16}$$

Using this result we have the general solution to (7.3.12),

$$T(n) = \begin{cases} \beta(a^p/(a^p - c))n^p & \text{if } c < a^p, \\ \beta n^p \log_a n + O(n^p) & \text{if } c = a^p, \\ O(n^{\log_a c}) & \text{if } c > a^p. \end{cases} \tag{7.3.17}$$

Now as stated in (7.3.13), $g(n)$ is proportional to the time needed to perform smoothing steps on G_k, to set up the coarse-grid correction equation and inject it to the next lower grid, and interpolate the correction from the lower grid to G_k.

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 Basic Multigrid algorithm is of *optimal order* if $T(n) = O(g(n))$, a possibility that is sometimes precluded by some choices of c, a, γ and d, which in turn influence $T(n)$. Examination of (7.3.17) 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 $c < a^d$.

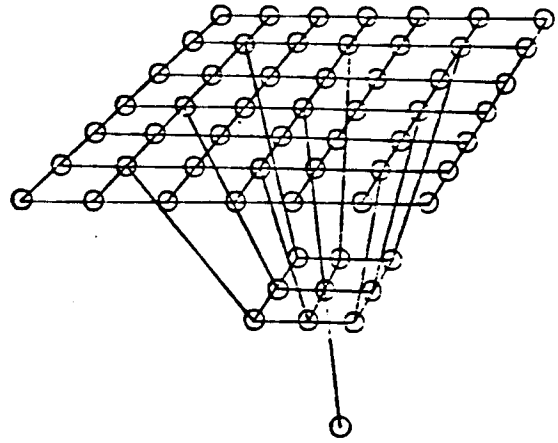
Thus the larger a or d is, the larger c can be and still keep the algorithm optimal. The trade-off is that a large value of a implies more relaxation sweeps are needed and thus c must be made larger so as to insure the previous level of accuracy. The constant in the $O(\cdot)$ term does not vary much with c or a so a reasonable strategy in

maintaining a balance between speed and accuracy is to choose c as large as possible so that the algorithm is still optimal, i.e., $c < a^d$. Example: if $d = 2$, $a = 2$, we should take $c = 2$ or 3.

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 $n_1 = 1, n_2 = 2(1+1) - 1 = 3, n_3 = 2(3+1) - 1 = 7$, (see equation (7.3.1)), 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.

Unfortunately, this design differs from the classical systolic array concept of Kung [8]-[11] 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 of this class of structures.

We might also remark that it is not necessary that the layers converge down to a single processor as in fig. (7.3.1). Instead, 3 or 4 levels of grids could be used and the multigrid method would still be highly efficient.



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

Figure 7.3.1 The Multigrid Architectural Model

4. Efficiency, Speedup, Accuracy, and Optimal Design

Now the four parameters c, a, γ, d are to be chosen with any implementation of BASICMG, 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 C & S [3], an important issue is *efficiency* , E vs. *speedup* S in a particular design. We define,

$$\begin{aligned} S(c, a, \gamma, d) &= T(c, a, 0, d)/T(c, a, \gamma, d) \\ E(c, a, \gamma, d) &= T(c, a, 0, d)/(P(\gamma)T(c, a, \gamma, d)) \end{aligned} \tag{7.4.1}$$

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 $E \to 0$ as $n \to +\infty$.

Theorem 7.4.1: (C & S, [3]) 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.* This is clear since c is a positive integer and we cannot have $c < 1 = a^0$.

3. "Halfway" between asymptotic efficiency and inefficiency is *logarithmic asymptotic efficiency*, with $E = O(\log n)$, as $n \to +\infty$. *A fully parallel design* ($\gamma = d$) *if logarithmically asymptotically efficient iff* $c = 1$. (See case 2) in Theorem 7.4.1).

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, (see last two cases in Case 3) of Theorem 7.4.1). 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.

C & S are then led to the following design problem.

Optimal Design Problem

For a given problem (i.e., given d), find the design that minimizes $T(n)$ and/or maximizes the accuracy of the computed solution, subject to the constraint that it is asymptotically efficient.

C & S go on the describe how the choice of any two of the parameters c, γ, a essentially predetermines the third according to the above design goal. We will also examine what for us is the relevant aspects of this relationship, but, as our objective is to find a computation time less than or equal to $O(n)$, even for $d > 2$, we proceed, for the sake of completeness, to an analysis of the time complexity of FULLMG, and will return with this knowledge to the design question.

Since FULLMG calls BASICMG, the computation time of the former depends heavily on that of the latter. Let $F(n)$ denote the time needed for one call of FULLMG. Inspection of the algorithm in Table (7.3.2) yields

$$F(an) = F(n) + rT(an) \tag{7.4.2}$$

Now $T(n)$ takes on the values of $O(n^p)$, or $O(n^p \log n)$, where $p = 0, 1, 2$ in the latter case. (See eqns. (7.3.16)-(7.3.17) for more details). Hence given a particular value of $T(n)$, we should be able to solve the recurrence in (7.4.2).

Theorem 7.4.2: (C & S, [3]) Recall that $a \geq 2$, $p = d - \gamma$. The following statements can be easily derived by using (7.4.2).

1). If $T(n) = \alpha n^p$, then

$$F_p(n) = (a^p/a^p - 1))r\alpha n^p + \ constant$$

2). If $T(n) = \alpha n^p \log_a n$, with $p > 0$, then

$$F_p(n) = (a^p/a^p - 1))r\alpha n^p \alpha \log_a n - (a^p/(a^p - 1)^2)r\alpha n^p + \ constant$$

3). If $T(n) = \alpha \log_a n$, then

$$F_p(n) = (r\alpha/2)(\log_a^2 n + \log_a n) + \ constant$$

Now from this information we are lead to a specific set of designs. Recall that if we are to at least equal the computing time of the systolic direct solver, we must

choose $T(n) = \alpha n$ or $T(n) = \alpha \log_a n$. Or, if we are to use the Full Multigrid algorithm, we must have $F_p(n) = O(n)$ or $O(\log^2 n)$. We will begin our discussion with the $O(\log n)$ time of BASICMG first.

A glance at eqns. (7.3.16)-(7.3.17) shows that the only time when $T(n) = O(\log n)$ is when $p = d - \gamma = 0$. Thus γ is chosen for us at the start. We see at once that we must relax the constraint on the optimal design problem of C & S since there is no way we can have the asymptotic efficiency requiring $c < a^{d-\gamma}$, where c is a positive integer. Now the next level of efficiency is logarithmic asymptotic efficiency, and this can be achieved only by setting $c = 1$, (see case 2) in Theorem 7.4.2). Now for a, the smaller it is, the more accurate the algorithm. Thus we set $a = 2$. Then our design will be

$$T(1, 2, d, d) = \beta \log_2 n + constant. \tag{7.4.3}$$

The constant depends on the time needed for a "direct solve" on the coarsest grid of a given multigrid design. The implication here is that if n were to increase, we would need more grid levels, but the size of the coarsest grid will remain constant and the computing time will grow only as in eq. (7.4.3).

If we were content with $T(n) = O(n)$, which presumably would only happen if efficiency were deemed important enough, then, keeping $a = 2$, we must have $\gamma = d - 1$ and $c = 1$. Thus

$$T(1, 2, d-1, d) = 2\beta n + constant.$$

The above time correspond to the length of *one* total cycle of BASICMG, which is not to be confused with the c inner cycles. We need to iterate the total cycle a number of times to reach convergence, and this number will depend on

1). the spectral radius of the Multigrid operator M_K,

2). and the quality of the initial approximation, which for the time-dependent Zakai equation is the value of the solution at the previous time-step.

At any rate, *the only way for* $T(n) = O(n)$ *is for* $c = 1$.

It may turn out that in some problems $c = 1$ is not enough, and that more inner iterations may be needed. This is because we would be increasing the number of numerical iterations and lowering the successive differences between the number of points of each point grid.

Another reason is that convergence theorems for the V-cycle (where $c = 1$), are either non-existent or not very promising, as they apply only to special cases such as symmetric and positive definite L^K, (see Hackbusch, [7]). However, Stüben and Trottenberg, [12], show that if $||M_2||$ is small enough, (say, $||M_2|| \sim .1$,) so that convergence is guaranteed for the general Multigrid case, then V-cycles may usually be used without any problems, although their convergence rates may not be as good as the $c = 2$ case.

Remark: This leaves open the question as to whether we have something to gain by using FULLMG instead of BASICMG. An increase in computing speed could not be a consideration, since

$$F_p(n) = O(n) \text{ if and only if } T(n) = O(n)$$

and the best it can do is $O(\log^2 n)$, which occurs when $T(n) = O(\log n)$. We naturally assume that we would wish to keep such speeds if we were to switch to FULLMG, but these occur only when $c = 1$.

Thus, $F_p(n) = O(n)$ *only when* $T(n) = O(n)$ *and only when* $c = 1$.

The *only* reason we would consider using FULLMG is in the hope that its automatically generated initial solution estimates could be better than the ones we would use for BASICMG, namely, the value of the solution at the previous time-step. This would imply that the "poor" estimate for BASICMG would take r_B total cycles, and that this would be much worse than the single cycle necessary for FULLMG and its r inner cycles of BASICMG.

From a purely theoretical point of view this may seem possible. To see why,

note that the FULLMG operator for the two-level case, when acting on the difference of its initial approximation and the true discretized solution, is

$$\mathrm{M}_2(I_1^2(L^1)^{-1}I_2^1 - (L^2)^{-1})U^2(t-\Delta t). \tag{7.4.4}$$

where $U^2(t-\Delta t)$ represents the solution at the previous time-step. In words, the approximation that FULLMG generates, which is $I_1^2(L^1)^{-1}I_2^1U^2(t-\Delta t)$, with $U^2(t)$ subtracted from it must be iterated to (nearly) zero by M_2 for convergence to occur.

The BASICMG operator on the other hand is

$$\mathrm{M}_2(I_2 - (L^2)^{-1})U^2(t-\Delta t). \tag{7.4.5}$$

Now we recognize the term in parentheses in (7.4.4) as being related to the "approximation property" discussed in the preceding chapter, where

$$||I_1^2(L^1)^{-1}I_2^1 - (L^2)^{-1}|| \leq Ch^2. \tag{7.4.6}$$

Now how many inner cycles of BASICMG will FULLMG need? This number is denoted by r, (see Table (7.2.2)), and we see that it must be large enough so that

$$\text{error tolerance } \sim ||\mathrm{M}_2||^r||I_1^2(L^1)^{-1}I_2^1 - (L^2)^{-1}||\,||U_{max}||,$$

where $||U_{max}|| = \max_{0\leq t\leq T}||U(t,x)||$, which is obtainable from estimates described in chapters 3-4.

On the hand using equation (from previous chapter) we have for using BASICMG alone,

$$\text{error tolerance } \sim ||\mathrm{M}_2||^{r_B}||L^2||\Delta t||U_{max}||,$$

where $||L^2|| = O(h^{-2})$. Presumably, this term would be offset by Δt. At any rate, these equations are unclear as to whether $r_BT(n) >> F_p(n)$, as we are using worst case estimates. When we conduct a more straightforward analysis with $\gamma = d, c = 1$ we obtain,

$$T(n) = \beta[\log_a n + constant],$$

and

$$F_0(n) = (r\beta/2)[\log_a^2 n + \log_a n + constant]$$

where we assume the constants are the same. Then if $r_B T(n) >> F_0(n)$, we would have,

$$\frac{r_b}{r} > \frac{1}{2}\log_a n,$$

which seems very unlikely for large n. Converting to the $T(n) = O(n)$ will not help as the convergence here is the same as the parallel case. The difference in the architectures is only in efficiency (or work distribution among the processors.) The Multigrid operator is the same in both cases and so convergence rates are the same.

Claim: We conclude that it is unlikely that FULLMG would be advantageously used. Also, that BASICMG can run in $O(n)$ efficiently and $O(\log n)$ with acceptable efficiency.

Observation: We turn now to an estimate of the *actual* time needed to perform BASICMG, in its most parallel form. The times given above are just the number of operational counts.

Recall that if $W(n) = \beta n$, then,

$$\beta = (j + m + s)t,$$

where

j = the number of smoothing sweeps on G_k prior to injection; (about 3 or 4).

m = number of smoothing sweeps on G_k after correction; (about 3 or 4).

s = ratio of time needed for injection and interpolation to the time needed for one smoothing sweep; (about $O(1)$).

t = time needed to perform one smoothing sweep; ($\sim$ 20 operation counts depending on the method).

Now if BASICMG takes

$$\beta \log_a n + constant$$

for *one* cycle, it will take r_B total cycles to reach convergence, which is probably of $O(1)$, according to the MG practitioners, (see Brandt, [2]). Since the constant is the time needed for a direct solve on a small, coarse grid, we will ignore it. Thus, $T(n) \sim O(10^2) \log_a n$ time units. Now the time units in VLSI are of the order of 10^{-2} μsec., so this implies that, if we are to have $T(n) \leq 1$ msec., then,

$$\log_a n \leq 10^3,$$

which for $a = 2$ should be quite reasonable for most applications.

5. Concurrent Iteration

Gannon and Van Rosendale have introduced an improvement on the original MG algorithm, which is designed to overcome the problem of inefficiency, or processor unemployment, [4].

Suppose that on the finest grid, one has the number of processors roughly comparable to the number of grid points. Then during the relaxation scheme, or during interpolation, the processor grids will be well employed. But as we move to the coarser girds, processor utilization will be poor. To overcome this problem, if indeed it is viewed as a problem, G & R proposed a way whereby iteration is performed concurrently on the grids.

To understand the technique, one must take an appropriate view of the MG algorithm. This begins by noting that all relaxation schemes are effective at reducing Fourier error components having wavelengths comparable the mesh spacing, but do poorly on error components having much longer wavelengths. This is compensated for by performing iterations on the coarse grids, where the mesh spacing *is* comparable to the error's longer wavelength. It is from this behavior that the multigrid method derives much of its effectiveness.

One could therefore view the MG algorithm as using projection operators to decompose residuals into their short and long Fourier wavelength components. A sequence of projections could therefore decompose a given function defined on the

finest grid into components on every grid. This might be thought of as a Fourier decomposition of a fine grid function, which G & R call an *approximate spectral decomposition.*

Suppose we are given u^K corresponding to the finest finite element space M_K corresponding to the grid G_K. There are K grids in all. We also let I_j^{j+1} be the interpolation operator to the finer grid G_{j+1}, and let I_{j-1}^j be the projection operator to the coarser grid G_{j-1}. Define a sequence of functions $p^i, 1 \leq i \leq n$ for each grid to be constructed by the following program:

$$p^K := u^K$$

for $i := n$ downto 1 do begin

1 $\quad p^{i-1} := I_{i-1}^i p^i$

2 $\quad p^i := p^i - I_{i-1}^i p^{i-1}$

end

The result will allow us to write,

$$u^K = \sum_{i=1}^{K} p^i,$$

with p^1 being the smoothest component of u^K and p^K the most oscillatory.

The concurrent iteration version of the MG method can now be described as follows. Given the problem $L^K u^K = f^K$, perform steps 1–3:

1). Form an approximate spectral decomposition of f^K:

$$f^K = \sum_{i=1}^{K} g^i,$$

where g^i is located on the i^{th} grid.

2). Approximately solve the problems,

$$L^i v^i = g^i, \qquad 1 \leq i \leq K,$$

by point iteration methods. Do so *concurrently* on each grid.

3). Sum the solutions on each grid to get,

$$\bar{u}^K = \sum_{i=1}^{K} v^i.$$

The central idea of this algorithm is that the iteration in step 2 can be done concurrently on all grid levels. One might suppose that interpolation in steps 1 and 3 must be done level by level, but this too can be avoided. The resulting program can be designed to obey,

for i := itmx do

begin

1. perform j smoothing inner iterations on each grid level
2. interpolate solutions and residuals between adjacent levels

end

The interpolation in step 2 can be done in parallel on all grid levels.

While an increase in speed is possible, the complexity of the program increases. One first needs a procedure to shift the data in a family of functions $\{p^i\}_{i=1}^K$ one level, by performing a sequence of injections. Then a similar procedure would be needed to perform projections between grid levels. G & R provide subroutines capable of conducting these operations in their ICASE report, where other details can be be found, [4]. They point out that no convergence proofs have yet been proven for their technique, but that its effectiveness is demonstrated by numerical experimentation.

We mention their technique as a demonstration of the versatility of the MG method, and also because it was included in the empirical results we will review. However, we repeat that the trade-off for increased speed is in the increased complexity of the programming, which in turn increases the complexity of the processors. Also, not all of the most natural VLSI architectures can accommodate concurrent iteration.

Finally we remark that the complexity analysis of Concurrent Iteration is very difficult as expressed to this author during phone conversations with Dr. Gannon. The reason is simple enough: the recursive structure of the classical MG algorithm is destroyed, so the elegant use of finite difference equations that we have seen in this chapter is not possible. However, Gannon and Van Rosendale have recently (March, '86) published a article outlining the complexity of this method. Unfortunately, it was not available to this author at the time of publication. In this report, they apparently explain why the spectral norm of the MG operator can undergo changes in size due to modifications of the domain. We will see examples of this in chapter 9.

In a private conversation, Gannon approximated the complexity of Concurrent Iteration as being $O(\log(\log(n)))$, but that the constant was apparently too difficult to compute.

Our won conclusion is that this program is still in its embryonic stages, and while we should not lose sight of its progress, we cannot advocate its use without reservation at this point in time.

6. Conclusion

The complexity of various Multigrid schemes were discussed, and we found that the Basic Multigrid algorithm has a computation time that is dependent on the architecture incorporating it. Improvements in computing speed are obtained at the expense of increased area as well as an increased inefficiency, which is roughly measured as the number of processors likely to be idle at a given point in the computing cycle. While trade-off between time and area is inevitable, inefficiency can be dealt with by the use of a program design known as concurrent iteration, which makes use of all processor grids simultaneously–unlike the prototype Multigrid model. Of course, an increase in both program and processor complexity must also be met.

The fundamental Multigrid Program parameters were identified. They are:

c, the number of coarse grid iterations per fine grid iterations

a, the mesh refinement ratio

d, the dimension of the point grids

γ, the dimension of the processor grids

A fully parallel design is possible with $\gamma = d$, which implies that we would need as many processor as points, obviously an area-intensive architecture. This can still be circumvented by employing more complex processors that sequentially handle more than one point.

However, the system with $\gamma = d$ cannot be asymptotically efficient, in that as the number of grid points n, increases, more and more processors are likely to become idle at any given time. This can be quantitatively measured as the following ratio:

$$T(c, a, 0, d)/(p(\gamma)T(c, a, \gamma, d))$$

where $T(\cdot)$ is the time needed for computation with the given parameters and $P(\gamma)$ is the number of processors given as a function of n^γ. If this ratio tends to a positive constant, as $n \to \infty$, we have efficiency. If it tends to zero, we do not.

It turns out that a combination of algorithm and machine is asymptotically

efficient if and only if $c < a^{d-\gamma}$. However, the fully parallel machine suffers only from an inefficiency that tends to zero as $1/\log n$. We also have, since $d = \gamma$, $c = 1$, which yields the V-shape MG cycle described in the last chapter. It basically means that we proceed straight from the fine to coarse grids, all the way down to the coarsest grids, and interpolate directly back up to the finest grid. There are no inner cycles, i.e., no intermediate stages with returns to fine grids and then back to coarse grids again.

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. A more detailed look at architectural effects on computing speeds will be given in chapter 9. However, under reasonable conditions, such as small values of d, we also showed that the computation time for our Multigrid algorithm is compatible with the 1 msec time bound placed upon us by the real-time constraints of our problem.

We also introduced a variation of the MG algorithm, known as Concurrent Iteration, that was developed by Gannon and Van Rosendale. While it appears to have a promising structure, and even appears to be somewhat faster than the conventional algorithm, it suffers from certain technical difficulties, such as an unstable spectral norm, and so, too little is known about it at this writing, to recommend it with confidence. A more detailed look at it will be given in chapter 9.

Also, in chapter 9, we will discuss some other designs and their architectural complexity.

8. Relaxation Schemes for the Multigrid Method

1. Introduction

Throughout the preceding chapters the theory of relaxation or iterative schemes has only been touched upon, even though they clearly play a major role in the Multigrid algorithm. This chapter is a discussion of of such schemes, and the way they can be incorporated into a parallel processing array designed for Multigrid operations to be performed on our PDE.

We follow four main lines of development.

The first is a brief statement of the basic relaxation schemes and their abstract representation. We follow the classic account by Young, [17].

The second is a discussion of the role of relaxation schemes within the Multigrid framework. We will learn that while such methods can be used by themselves to solve PDEs, the question of their convergence, although needed as a prerequisite, is not the most important issue for us. We wish only to smooth out high-frequency error between the true solution and its approximation, prior to transferral to a coarser grid. For this purpose, we need good smoothing properties of the scheme only for the highly oscillatory components, which by itself does not necessarily imply good convergence behavior. It is therefore conceivable that a MG practitioner might use a poorly converging scheme as a smoother. Methods for measuring smoothing behavior will be discussed. The work of Brandt, [2]-[3], and Stüben and Trottenberg, [16], will be cited.

The parallel implementation of the relaxation scheme is the third topic, and we will find that not all schemes are equally suitable for such computing systems, but that dramatic increases in processing speed are possible when the execution is in parallel. The most likely candidates for such processing will be identified. This section follows Sameh, [15] as well as Brandt, [2]-[3].

Relaxations schemes turn out to be very suitable for asynchronous operation,

and the desirability for exploiting this feature will be addressed as our fourth topic. The reasons are two-fold: whenever a numerical method lends itself naturally to a certain kind of implementation, then, in the interests of high-speed processing, it is often to our advantage to design the system accordingly. The second reason is that global synchronization of large arrays is a troublesome problem at best, especially at VLSI signal speeds. Asynchronous data-flow is the reasonable alternative. We will be influenced in our discussion by S. Y. Kung, [7], and others.

Main results: The identification of the relaxation method most compatible with the properties of our matrix $I + \Delta t A$; and an elucidation of an appropriate architecture for parallel asynchronous implementation.

2. Six Basic Relaxation Schemes

For a survey of the theory of relaxation processes, we refer the reader to Young, [17].

We present in this section the six most commonly used iterative methods. First we establish some notation. Starting with $Ax = b$, with $a_{ii} \neq 0$, define the $n \times n$ matrix B to have components,

$$b_{ij} = \begin{cases} -a_{ij}/a_{ii}, & i \neq j \\ 0, & i = j \end{cases} \tag{8.2.1}$$

and define vector c in R^n to have components,

$$c_i = b_i/a_{ii} \quad 1 \leq i \leq n. \tag{8.2.2}$$

It follows that $Ax = b$ is equivalent to

$$x = Bx + c. \tag{8.2.3}$$

We could also write, $D =$ the diagonal of A, i.e.,

$$d_{ij} = \delta_{ij} a_{ij}, \tag{8.2.4}$$

and

$$C = D - A, \tag{8.2.5}$$

Thus,

$$\begin{aligned} B &= D^{-1}C \\ c &= D^{-1}b \end{aligned} \tag{8.2.6}$$

1). The *Jacobi method* J method) is defined as,

$$\begin{aligned} x_{n+1} &= Bx_n + c \\ &= (I - D^{-1}A) - D^{-1}b \end{aligned} \tag{8.2.7}$$

Since $I - B = I - D^{-1}C = D^{-1}(D - C) = D^{-1}A$, we have $(I - B)A^{-1}b = c$. Thus the existence of A^{-1} forces $(I - B)$ to be invertible and so the method is completely consistent, as per our earlier argument.

2). The *simultaneous overrelaxation method* (JOR method) is formed by choosing $\omega \in R$ and writing,

$$x_{n+1} = (\omega B + (1 - \omega)I)x_n + \omega c. \tag{8.2.8}$$

If $\omega \neq 0$, the method is completely consistent, and if $\omega = 1$, we have the Jacobi method.

If $\omega > 1$, we are in a sense, "overcorrecting," since

$$x_{n+1} = x_n + \omega(x_{n+1} - x_n),$$

where x_{n+1} is obtained from the J method. And if $\omega < 1$, we are "undercorrecting." The choice of ω can be optimized for a given problem for maximal rate of convergence. See Young, [17], for details.

3). The *Gauss-Seidel method* (GS method) is obtained by forming,

$$B = L + U, \tag{8.2.9}$$

where L and U are strictly lower and upper triangular matrices respectively, i.e., $l_{ii} = u_{ii} = 0$, for $i = 1, 2, ..., n$. Thus

$$x_{n+1} = Lx_{n+1} + Ux_n + c, \tag{8.2.10}$$

where we use the i^{th} component $(x_{n+1})_i$ when available. For example,

$$\begin{aligned}(x_{n+1})_1 &= b_{12}(x_n)_2 + b_{13}(x_n)_3 + c_1 \\ (x_{n+1})_2 &= b_{21}(x_{n+1})_1 + b_{23}(x_n)_3 + c_2 \\ (x_{n+1})_3 &= b_{31}(x_{n+1})_1 + b_{32}(x_{n+1})_2 + c_3\end{aligned} \tag{8.2.11}$$

Note that $det(I - L) = 1$. Therefore, with

$$\mathcal{L} = (I - L)^{-1}U,$$

we can write,

$$x_{n+1} = \mathcal{L}x_n + (I - L)c, \tag{8.2.12}$$

which is in standard form. If A^{-1} exists, the method is completely consistent.

4). The *successive overrelaxation method* (SOR method) operates on the same principle as the GS method:

$$x_{n+1} = \omega(Lx_{n+1} + Ux_n + c) + (1 - \omega)x_n, \tag{8.2.13}$$

which can be written,

$$x_{n+1} = \mathcal{L}_\omega x_n + (I - \omega L)^{-1}\omega c, \tag{8.2.14}$$

where,

$$\mathcal{L}_\omega = (I - \omega L)^{-1}(\omega U + (1 - \omega)I).$$

And so, if A^{-1} exists, the method is completely consistent. If $\omega = 1$, the SOR method reduces to the GS method, with $\omega > 1$ implying overcorrecting, and $\omega < 1$ implying undercorrecting.

5). The *stationary generalized Richardson's method* (GRF method) is designed for the case when the diagonal elements of A may vanish. We define,

$$x_{n+1} = x_n + P(Ax_n - b), \tag{8.2.15}$$

where P is a nonsingular, diagonal matrix. We can write (8.2.15) as

$$x_{n+1} = R_P x_n - Pb, \tag{8.2.16}$$

where,

$$R_P = I + PA.$$

If A^{-1} exists, the method is completely consistent, since $I - R_P = -PA$ and $(I - R_P)A^{-1}b = -Pb$. Note that if $P = -D^{-1}$, we have the J method.

6). The *stationary Richardson method* (RF method) is a special case of the GRF method, for we simply set $P = pI$, where p is a nonzero real number. As before, the method is also completely consistent if A^{-1} exists.

For the equivalent notation of (6.5.3), with $Q^{-1} = M$, we have the results of Table (8.2.1), where $A = D - C_L - C_U$. These terms are defined as, D = the diagonal of A, and C_L and C_U are strictly lower and upper triangular matrices, with zeroes in their diagonals.

Q	Method
D	J
$\omega^{-1}D$	JOR
$D - C_L$	GS
$\omega^{-1}D - C_L$	SOR
$-p^{-1}I$	RF
$-P^{-1}$ (P diagonal)	GRF

Q Values of Relaxation Methods

Table 8.2.1

As we have already seen, if A^{-1} exists, then an iterative method can be found to solve $Ax = b$. But much of the theory of these processes centers around other special properties that A could have, and which could be more efficiently exploited. Examples include, weak diagonal dominance, irreducibility, symmetry, and positive

definiteness, but all of these conditions still imply that A^{-1} exists. A survey of all such cases would not be suitable for our own, more narrow, interests. The reader is directed to Young, [17], for more information.

Finally, we do not mean to imply that the above six methods are all that is available. Far from it, as many techniques exist and are described in Young's classic work. But the other methods are mostly for special, and often difficult cases. For now, they will not concern us.

3. Measurement of Smoothing Factors

Much of the literature on iterative methods centers around their convergence properties, the value of one method over another for a given problem, and the means of optimizing the convergence rate by manipulation of certain parameters, such as those in examples 2 and 4 in section 8.2.

In Multigrid theory, this concern over convergence is of much less importance since iterative methods are never allowed to converge anyway. They are only designed to reduce the high-frequency error of the approximation relative to the true discretized solution. Thus denoting U^h as the true solution to

$$L^h U^h = f^h,$$

and u^h as a given approximation to U^h, the difference $U^h - u^h$ would have Fourier components $e^{i\theta \cdot x/h}$. Then a relaxation scheme would multiply each Fourier component by $\mu(\theta)$, which is defined as the factor by which the amplitude of that component is multiplied as the result of one relaxation sweep.

We remind the reader that what is being transferred to the coarser grid is the defect $f^h - L^h u^h$, which can be written as $L^h(U^h - u^h)$. Since L^h is linear and bounded as $O(h^2)$, then smoothing the superposition of terms,

$$U^h - u^h = (U^h - u^h)_{low frequencies} + (U^h - u^h)_{high frequencies},$$

implies that the high frequency terms will remain damped. after transformation by L^h.

A simple example was already encounted in the introductory chapter on Multigrid theory in connection with Poisson's equation. In fact, the only examples that are analytically tractable are when L^h has non-variable components (i.e., independent of $x \in G_h$,) which is indeed the case when L^h corresponds to a five or nine point star discretization of the Laplacian.

The smoothing factor $\bar{\mu}$ is defined as

$$\bar{\mu} = \max_{\pi/2 \leq |\theta| \leq \pi} |\mu(\theta)|.$$

Clearly, this is the *worst* factor with which to multiply the high-frequency components. It follows that we wish $\bar{\mu}$ to be as small as possible. As stated, this term is not easy to calculate analytically in all but the simplest cases. Fortunately, there exists a computer program to calculate it for us known as SMORATE, [3], which is part of a package of Multigrid related programs and recommended by Brandt, [3].

The above remark about the size of $\bar{\mu}$ is not meant to imply that it is unimportant that the spectral norm of the smoother not be as close to zero as possible, as $||S^{j_1+j_2}||$ factors into the bound of $||M_k||$, (see the chapter on Multigrid theory.) For example, suppose smoother $\tilde{S}$ has better smoothing properties than smoother S, but that $||S|| < ||\tilde{S}|| < 1$. Because of better smoothing we have $\tilde{j}_1 \leq j_1, \tilde{j}_2 \leq j_2$. Thus

$$||S||^{j_1+j_2} < ||\tilde{S}||^{\tilde{j}_1+\tilde{j}_2}.$$

As desirable as this may look from the point of view of the convergence of the Multigrid method, we may pay for this by increasing the computing time of the algorithm, due to the increased number of smoothing sweeps. Of course, this can be compensated for by fewer Multigrid cycles with smoother S. The issue would be settled by an investigation with estimates on $\bar{\mu}_S, \bar{\mu}_{\tilde{S}}, ||S||, ||\tilde{S}||, j_i$ and $\tilde{j}_i$.

For the more general case, when L^h is dependent on $x \in G_h$, as it will be in our investigations, one makes use of what Brandt calls *local Fourier analysis*, [3]. This is based on the principle that relaxation schemes operate locally, near or at, a

point. That is, all the information required for "improving" the approximation at a point depends only on the local, or nearby, values. (This fact will be expounded upon later when we investigate the parallel implementation of relaxation schemes.) The idea is to examine a single point, $x \in G_h$, and evaluate L^h at this point, i.e., all the coefficients contained in L^h that were part of the PDE are evaluated at x. Using this new matrix L^h, one obtains $\bar{\mu}(x)$ just as before, only now we say

$$\bar{\mu} = \max_{x \in G_h} \bar{\mu}(x).$$

To understand the concept of smoothing in the context of Fourier of harmonic analysis, let the eigenvectors of $(I_h - ML^h)$ be $\{\phi_k(x)\}$, where the smoother is

$$(I_h - ML^h)(\cdot) + Mf.$$

Now we assume that

$$U^h - u^h = \sum \alpha_k \phi_k(x) \sim \sum \gamma_k e^{i\theta_k \cdot x/h} \tag{8.3.1}$$

where the Fourier expansion is obtained by,

$$\phi_k(x) \sim \sum \beta_j(k) e^{i\theta_i \cdot x/h}. \tag{8.3.2}$$

Now the new approximation given by the smoother is

$$\bar{u}^h = (I_h - ML^h)u^h + Mf^h, \tag{8.3.3}$$

and, by consistency,

$$U^h = (I_h - ML^h)U^h + Mf, \tag{8.3.4}$$

and so

$$U^h - \bar{u}^h = (I_h - ML^h)(U^h - u^h) = \sum \alpha_k \lambda_k \phi_k(x). \tag{8.3.5}$$

With the help of eq. (8.3.1)-(8.3.2), we can write,

$$U^h - \bar{u}^h = \sum \mu(\theta_k) \gamma_k e^{i\theta_k \cdot x/h}. \tag{8.3.6}$$

Now, in the especially transparent cases when L^h has the same eigenvalues as $(I_h - M^h)$, which was the case in Poisson's equation, the defect, $d^H = f^H - L^H u^H = L^H(U^H - u^H)$ will not amplify the higher frequency components.

Brandt, [2]-[3], points out that $\bar{\mu}$ is the first and simplest quantitative predictor of the *obtainable* Multigrid efficiency. That is $\bar{\mu}^{j_1+j_2}$ is an approximation to the asymptotic convergence factor obtainable per multigrid cycle, $\rho(\mathrm{M}_k)$. This term is often very accurate, especially when the grid equations do not change much with a few mesh sizes for the finest grid, and it sets an ideal figure against which the performance of the full algorithm can later be judged. However, the most important aspect of $\bar{\mu}$ is that it separates the design and choice of the relaxation scheme from all other algorithmic considerations.

As pointed out earlier, when we compare various relaxations schemes, the smoothing factor should not be our only criterion of choice. One aim is to have the best high-frequency convergence rate per operation, i.e., the largest $w_0^{-1}\log(1/\bar{\mu})$, where w_0 is the number of operations per gridpoint per sweep. This measure was introduced by Brandt, [3], with the logarithm being used to detect small changes in $\bar{\mu}$. With two schemes of similar values of $w_0^{-1}\log(\bar{\mu})$, Brandt argues that the simpler scheme corresponding to the smaller w_0, should be used, because in practice, very small factors of $\bar{\mu}$ are essentially unattainable: the coarse-grid operator is usually unable to sustain them. Troublesome interactions with boundaries are another reason.

Another important consideration, is that the relaxation scheme be stable. Of course, all the standard schemes already mentioned are already stable, but smoothers being designed for nonelliptic and singular perturbation equations have shown instabilities. Stability analysis for such schemes can usually be done in an analogous fashion with von Neumann stability analysis for time-dependent problems, taking the main relaxation marching direction as the time-like direction.

It also appears that relaxation can a certain effect on the smooth or low-

frequency components as well. Usually, this effect is nil, with $\mu(\theta) \approx 1$ for small $|\theta|$. But there are cases according to Brandt, [3], where we have an excellent $\bar{\mu}$, while for small $|\theta|$, $|\mu(\theta)| >> 1$, which implies divergence for low frequencies. Apparently, these cases have occurred mostly in hyperbolic equations. At any rate, such schemes clearly must be discarded, and it is therefore in our interests to incorporate into the program for calculating $\bar{\mu}$, a check on the stability of the scheme. For example, the value $\max_{|\theta| \leq \pi} |\mu(\theta)|$, would certainly be worth examining. This is one of several measures that are included on SMORATE.

4. On The Choice of Relaxation Schemes

One would hope that, out of all the available relaxation methods, only one need be chosen and used for all conceivable problems. This, unfortunately, is out of the question even for some restricted classes of problems. It certainly would not be true in the case of the Zakai equation, owing to the variety of implicit schemes and mesh designs we can choose from for the higher dimensions; (this is still an open area of research.) The range in variability in the coefficients might also raise doubts about the wisdom of using one and only one relaxation scheme for cases. It is here that the MG practitioner must be prepared for a combination of theoretical *and* experimental work.

In addition to the smoothing factor, another criterion is the structure and nature of the matrix A which correspond to L^h, the discretization operator of the PDE.

We remind the reader that:

Defn: A matrix $A = a_{ij}$ of order N has *weak diagonal dominance* if

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq 1}}^{N} |a_{ij}| \quad i = 1, 2, ..., N \tag{8.4.1}$$

and for at least one i,

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq 1}}^{N} |a_{ij}| \tag{8.4.2}$$

If (8.4.2) holds for all i, then A is said to have strong diagonal dominance.

We already showed that the matrix $I + \Delta t A$ enjoys the property of strong diagonal dominance.

A very important pair of theorems is now relevant to our needs:

Theorem8.4.1: (Young, [17]) If A has strong diagonal dominance, then $\det A \neq 0$.

Theorem8.4.2: (Young, [17]) If A has real entries with nonnegative diagonal elements and is nonsingular, then A is positive definite.

Clearly, our own matrix has these properties.

In addition, with respect to our purposes, the most important class of matrices we should consider is that of L-matrices.

Defn: A real matrix A of order N is an L-matrix if

$$a_{ii} > 0, \quad i = 1, 2, ..., N \tag{8.4.3}$$

and

$$a_{ij} \leq 0 \quad i \neq j \quad i, j = 1, 2, ..., N \tag{8.4.4}$$

Defn:A Stieltjes matrix is a positive definite L-matrix.

Claim: The matrix $I + \Delta t$ is a Stieltjes matrix.

It is clearly an L-matrix, and the fact that it has strong diagonal dominance implies that it is positive definite.

We also have from Young,

Theorem8.4.3: (Young, [17]) If A is an L-matrix such that $\rho(B) < 1$ and if $0 < \omega_1 \leq \omega_2 \leq 1$, then

$$\rho(\mathcal{L}_{\omega_2}) \leq \rho(\mathcal{L}_{\omega_1}) < 1$$

It can also be shown that if all the eigenvalues of B are real eigenvalues, whose absolute values are less than unity, then the optimal choice of ω_b is

$$\omega_b = 2/(1 + (1 - \rho(B))^{1/2}) < 2. \tag{8.4.5}$$

Note that all eigenvalues of B will be real if A is a Stieltjes matrix.

An important concept in connection with the SOR method is the rather technical notion of *consistent ordering.* First, we have

Defn: Given a matrix $A = a_{ij}$, the integers i, j are associated with respect to A if $a_{ij} \neq 0$ or $a_{ji} \neq 0$.

With the above we can say,

Defn: The matrix A of order N is *consistently ordered* if for some t there exists disjoint, possibly empty subsets $S_1, S_2, ..., S_t$ of $W = \{1, 2, ..., N\}$ such that $\cup_{k=1}^{t} S_k = W$ and such that if i, j are associated, then $j \in S_{k+1}$ if $j > i$ and $j \in S_{k-1}$ if $j < i$, where S_k is the subset containing i.

Some matrices can, with the help of permutation matrices, be made consistently ordered. A matrix A, for which this is possible is said to have Property A, in which case $P^{-1}AP$ is consistently ordered. Young, [17], gives a description of a computer program designed to test for this property in a given matrix. An alternative definition is to say that A is similar to A' where

$$A' = P^{-1}AP = \begin{pmatrix} D_1 & H \\ K & D_2 \end{pmatrix} \tag{8.4.6}$$

and where D_i are square diagonal matrices.

Matrices that arise from five-point star finite difference schemes for elliptic PDE's frequently have Property A.

Yet another way of checking for consistent ordering is by a graphical examination of the labeling scheme used on the grid points. In our case we have a rectangular (or hyper-rectangular) grid. Let i, j be adjacent points on the grids, and their numerical labels be i and j. If $i < j$, draw an arrow emanating from i to j. Do this for all adjacent points in the grid. Now if we follow any circuit in the grid by passing through the grid points and arrows, and if the number of arrows pointing in the opposite direction of our path is equal to the number of arrows pointing in the same direction as our path, then the finite difference matrix formed

from such a grid will be consistently ordered. It can be shown that all naturally ordered rectangular grids will form consistently ordered finite difference matrices.

Claim: The matrix $I + \Delta t A$ is consistently ordered, as it is formed form a natural ordering on a rectangular grid.

We can now make the following conclusions. The relaxation method of choice for the MG practitioners is the SOR method, for, when it is applicable, it is usually superior in both smoothing and convergence to the other classic method. Indeed, if we have that

1). A is consistently ordered and positive definite or

2). A is positive definite and an L-matrix,

then the SOR method is an excellent choice. In fact, for case 2), the J method will converge while the GS method will be at least as fast. And if B is the matrix of the J method, we can have a substantial improvement over the GS method by using $\mathcal{L}_{\omega_b}$ where

$$\omega_b = 2(1 + [1 - \rho(B)]^{1/2})^{-1}. \tag{8.4.7}$$

We add that if the matrix A only has the above properties, then the SOR method is still the ideal method with respect to smoothing ability. In fact, Brandt, [3], together with Grosch, [5], argue that this is *the* method to use for linear PDEs with finite difference matrices having one or both of the properties above. Their work is based on a combination of theoretical and experimental work, for we must be content with the analytic intractability of the the variable coefficient case. However, the SOR-method appears to be superior both smoothing and computational complexity. We therefore have the weight of numerical evidence of the mainstream Multigrid community in our favor. This should not be dismissed too lightly, as there is a considerable research effort going on to develop new relaxation methods for more difficult PDE problems. We can avoid this entirely.

Claim: We can now effectively state that if the domain if a hyper-rectangle, in R^n,

and if a natural ordering of points is used, then we will have a consistently ordered matrix for our linear system, as the matrix is of the form,

$$(I + \Delta t A) \tag{8.4.7}$$

where A was defined in chap. 5, with the absolute value of the its diagonals being the expected value of the time the Markov chain to remain at that point, and all other points being related to transitional densities. It was shown in chap. 5 that $(I + \Delta t A)$ had strong diagonal dominance, and we know from this section that such matrices are positive definite. Thus this assures us that the SOR-method is the best choice, with (8.4.7) being the optimal selection of ω with respect to convergence. However, numerical experimentation, in combination with the program SMORATE, will be needed to choose the best choice of ω with respect to good smoothing.

5. The Parallel Implementation of Relaxation Schemes

It is not hard to imagine how such schemes can be implemented on a processor array. Sameh, [15], was among the first to discuss how naturally these schemes lend themselves to parallel implementation. We consider first only the two-dimensional case for the sake of simplicity, but generalizations will be possible.

Now suppose we have an approximation u^h to U^h, the true solution to

$$L^h U^h = f^h.$$

A processor array manages the computation to be done on the grid. For simplicity, let this be the case of maximal parallelism: we have as many processors as grid points. Generalizations to less parallel schemes will be clear. Thus, if the approximation were fed into the array, we would have,

$$\begin{matrix} u_{k-1,j-1} & u_{k-1,j} & u_{k-1,j+1} \\ u_{k,j-1} & u_{k,j} & u_{k,j+1} \\ u_{k+1,j-1} & u_{k+1,j} & u_{k+1,j+1} \end{matrix} \tag{8.5.1}$$

where the location of each $u_{(\cdot,\cdot)}$ corresponds to *both* a processor and a grid point, as well as the approximation to $U^h_{(\cdot,\cdot)}$. Note that in the mapping from the grid to

the matrix operations, in which the solution to the discretized PDE is a solution to a linear system, we have,

$$u_{k,j} \rightarrow u_{(k-1)n+j} \tag{8.5.2}$$

where u_p is in $R^{m \cdot n}$, with n denoting the largest width of the rectangular domain. This is the vector we speak of when we write $L^h u^h$, where L^h is an $n \cdot m \times n \cdot m$ matrix.

Now suppose we use the Jacobi method, which has an implementation that is especially transparent. Here the iterative method is of the form:

$$\tilde{u} = (I - (diag(L^h))^{-1} L^h)(u) + (diag(L^h))^{-1} f^h. \tag{8.5.3}$$

Note that f^h is a function of the solution at the previous time-step, and we assume that these functional values have been stored in the appropriate processors.

Then, denoting $M = (diag(L^h))^{-1}$,

$$\begin{aligned} \tilde{u}_{(k-1)n+j} = \sum_{p=1}^{n \cdot m} (\delta_{p,(k-1)n+j} - m_{(k-1)n+j,p} l_{(k-1)n+j,p} u_p \\ + m_{(k-1)n+j,p} f_p \end{aligned} \tag{8.5.4}$$

Now L^h is likely to be sparse, and this would not seem to bode well for parallel implementation. But the sparsity is a distortion of the mapping in (8.5.2). For the updated version of $u_{k,j}$ in (8.5.1), we will need the north-south-east-west values assuming a five-point star scheme. Thus only *local interprocessor communication* is needed. This is what is meant by the "local" nature of relaxation operations.

Unfortunately, the smoothing properties of the Jacobi method are not always adequate, and so we will probably have to turn to other techniques.

However, there is a very efficient way to implement the usually superior SOR method that is no more difficult than the J method. This is the "red-black" or "checkerboard" method. We consider a rectangular grid where all points have been "labeled" as if each designated a red-black square on a checkerboard. In (8.5.1), this corresponds to setting

$$u_{ij} = \begin{cases} \text{red if} & i+j \text{ even} \\ \text{black if} & i+j \text{ odd} \end{cases} \tag{8.5.5}$$

Now on all the red points, conduct a JOR sweep which is represented as

$$(I - \omega D^{-1}L^h)(\cdot) + D^{-1}L^h, \tag{8.5.6}$$

with $D = diag(L^h)$. Then, using the newly obtained values at these points, conduct a JOR sweep at the black points. The reader should convince himself that this two-part scheme can easily be done in parallel. Incidently, if $\omega = 1$ in the JOR method, this red-black operation would yield the Gauss-Seidel method.

The red-black approach works very well for the five-point difference scheme, it will fail for the more general case where mixed partials are present, (which requires a nine-point difference scheme,) and for general finite element grid designs. Sometimes the mixed partials can be removed by a linear transformation, but not always. In this case, an extension of the foregoing ideas is available, and is known as a multicolor ordering scheme. A four-color mapping seems to be appropriate for the nine-point star implicit scheme; a generalization of this concept can be found in Adams and Ortega, [1].

The idea of multicolor ordering is similar to the red-black concept. We wish to label (color) the grid points in such a way that there is local decoupling between the unknowns, thus making the scheme more suitable for parallel implementation. In general, p colors yields p Jacobi sweeps, with information coming from the eight nearest neighbors of a grid point. The multicoloring scheme is usually straightforward for a rectangular domain with the discretization pattern being repeated at each grid point, and in general, Adams and Ortega [1] argue that no more than six colors will suffice for most problems. However, obtaining the minimum number of colors for arbitrary discretizations and irregular regions is equivalent to the graph coloring problem * which is known to be $NP-$complete, (Ortega and Voight, 1985, [13].)

* This graph problem involves the assignment of colors to the vertices of a graphs in such a way that no two adjacent points have the same color. For planar graphs, no more than four colors are needed, (this is the celebrated Four Color Map The-

6. The Asynchronous Implementation of Relaxation Schemes

As pointed out in the section "The Synchronization of Large Processor Arrays" in chap. 2, the design of synchronous multiprocessor arrays is problematical at best, if not impossible for the kinds of VLSI signal speeds we are aiming for, since the clock skew would be at or near the same order of magnitude as the pulse rate. This implies that asynchronous operation may be at least desirable, if not actually necessary. Fortunately, the structure of relaxation schemes offer themselves quite naturally to asynchronous implementation.

Experimentation has shown that such asynchronous methods are faster than synchronized computations. To obtain an estimate as to how long it will take to perform one SOR sweep, we note that, if done synchronously, the two parts of the sweep, involving first red points and then black points, takes 10 time units. This comes from the five points schemes and the multiplications that occur at each point. We assume that all the multiplications are done simultaneously in one time-step, then there are four additions. This does all the red points if all of them take their data from the north, south, east and west processors in unison. Then the black processors do the same. Thus the sweep is done in 10 units.

Now if done asynchronously, we would have 5α as the expected time for all the red points to complete their work, where $\alpha < \Delta t =$ one synchronous time-step. Now a black point can begin its computations as soon as any of the four red points surrounding it are available. Using standard state notation we see that a black point can enter one of four states:

state (1,0,0,0), if the north point is first available,

state (0,1,0,0), if the east point is first available,

state (0,0,1,0), if the south point is first available,

and state (0,0,0,1), if the west point is first available.

---

orem,) but this is not the case for hyper-graphs to which the above problem is mathematically equivalent.

Now when made available, the red points conduct the necessary multiplication and transfers the result to the black point, which will not change its value until *all* the necessary calculations are done. Assuming the entire process has been synchronized at time $t = 0$, we see that a black point can be expected to be in one of the four above states at time $\beta 5\alpha$, where $\beta > 1$. Now suppose it is in state (0,1,0,0), then it can go to three other states, (1,1,0,0), (0,1,1,0), or (0,1,0,1). We assume this is done with an expected time of $\beta \cdot \beta 5\alpha$. Continuing in this way, it arrives at its final state with expected time $\beta^4 5\alpha$. Now if $\beta = (1 + 1/10)$ and $\alpha = 0.75\Delta t$, the expected time for completion for the first stage of the SOR sweep is,

$$asynchronous\ time = (1 + 1/10)^4 5(.75)\, time\ units$$

which roughly equals, using $(1 + 1/10)^4 \approx (1 + 2/5) = 7/5$, and the fact that $\frac{7}{5} \times \frac{3}{4} \approx 1$, about 5 time units, an increase of 50 percent.

This is perhaps the simplest analysis possible. An alternative is to use more probabilistic methods, similar to those used in the work of Gelembe, *et al*, []. This implies that, upon entering the "black" stage, the red processing times are governed by exponential distributions. This suggest that the black processor is waiting for four random times $t_i, i = 1,2,3,4$, governed by $\gamma_i e^{-\gamma_i t_i}, i = 1,2,3,4$. Thus the black processor would be ready at time $t = \max\{t_i\}$. If we assume that the four red processors have equal exponential parameters γ, where $1/\gamma \leq \Delta t$, then t has density $4\gamma e^{-4\gamma t}$. The expected time for the completion of the sweep is then $.25/4\gamma$, a considerable savings in time. All of this assumes that the processors have the appropriate buffers to store the interim values, and that this "local analysis" can give performance measures of the global process, which seems reasonable given the nature of relaxation schemes.

Of course, we must also consider the actual design of the asynchronous controls. This often involves a considerable cost in overhead, due to the nature of the protocols needed to manage interprocessor communication.

In the context of this specific application and the special needs of VLSI systems, a natural candidate for the processor array that implements the red-black SOR method is the Wavefront Array Processor, (or WAP,) that was described in chap. 2 as well as in the paper by S. Y. Kung, [7]. This machine has been offered as a compromise between the more powerful (and hence more expensive) capabilities of the general data-flow machine and the systolic array with its troublesome synchronization problems. The WAP still enjoys,

locality, for efficient interprocessor communications,

regularity, for a more facilitated fabrication process,

recursivity, for ease in programming design,

concurrency, for maximum parallelism.

This compromise is deemed necessary since a more general asynchronous machine architecture is simply not suitable for our size and complexity constraints, due to its more sophisticated, and hence more expensive protocol designs. The reader will agree that the four properties mentioned above are consistent with our VLSI design guidelines.

The WAP system comes complete with its own programming language known as the Matrix Data-Flow Language (MDFL). With this we can program the asynchronous relaxation, injection and interpolation operations. The relaxation scheme can be viewed as a sequence of computational wavefronts. Each sweep of the relaxation is actually in two parts, so one wavefront is over all the "red" processors first, and second wavefront is generated for the "black" processor. The language MFDL will allow us to program all the appropriate processors at once, by using commands such as

CASE KIND=

(I, J) WHERE $I + J$ EVEN

FETCH X, UP

FETCH Y, DOWN, etc. More details can be found in H. T. Kung's paper, [].

The use of this architecture for interpolation and injection is even more obvious, as this involves nothing more than taking averages among the four or eight nearest neighbors. This clearly can be done asynchronously.

Communication among the processors in the array is governed by a handshaking protocol in which they must each wait for a primary wavefront of data, then perform their computations and, finally act as a secondary source of new wavefronts. The waiting is implemented by data transfer buffers. Thus a FETCHing of data, involves an inherent WAITing for the buffer to be filled, and the DATA READY flag to be raised by the adjacent data sourcing processor. Thus the processing will not be initiated until the arrival of data wavefront, which is similar to the concept of data flow machine. The complexity of the protocols is entirely consistent with simple unit processor design.

The WAITS for wavefronts of data allow for globally asynchronous operation of processors; thus the synchronous timing issue is conveniently avoided.

We therefore recommend the WAP architecture for the implementation of relaxation, interpolation and injection operations.

7. Conclusion

This chapter discussed the theory, operation and parallel implementation relaxation schemes, and their role in Multigrid methods. The main results were a further investigation of the properties of our $I + \Delta tA$ matrix, which led to our choice of appropriate relaxation scheme, and a description of how the scheme could be executed asynchronously, as synchronization of large arrays may prove to be insurmountable problem.

We cited the standard relaxation schemes among which we would normally choose. The method of choice was found to be the successive over-relaxation (SOR) method, which is a combination of the Jacobi and Gauss-Seidel schemes. Most importantly we identified our problem as having a matrix, formed by the implicit scheme discussed in chap. 2, that is diagonally dominant, and hence positive definite. Being in addition an L-matrix, the SOR-method is not only an excellent choice, but the optimal parameter ω for convergence is known to be

$$\omega = 2(1 + [1 - \rho(B)]^{1/2})^{-1},$$

where $B = (I - diag(L)^{-1}L)$ and $\rho(B)$ is the absolute value of the maximum eigenvalue. However, for improved smoothing capabilities, a different choice of ω may be necessary, and will probably required numerical experimentation to obtain. Of course, such work is *precomputable* as far as our signal processing is concerned. The package program SMORATE, which measures the smoothing factor of of a given scheme and system may prove helpful in this regard. Grosch [5] also discovered the superiority of the SOR-method; see his excellent paper for a review of some simulations that he performed.

The basic role of the schemes in Multigrid theory is identified as the fulfilling the need for smoothing the error in the the defect: $f^k - L^k u^k$, where u^k is generated by the relaxation scheme. This allows for a transfer of the defect to the coarser grid, without the loss of too much information represented in the highly oscillatory

components of the error–as this would be invisible anyway on the coarse grids. Means of measuring this error were discussed.

A discussion of how the relaxation schemes could be implemented on an array computer was described, with the "local action" of the schemes being emphasized. This means that, in addition to the data in a given processor, the data in the processors to the north, south, east and west of it are all that is needed. We found that the SOR method could be implemented by using a "red-black" ordering. Here the processors in the array are labeled as if laid out on a checkerboard. The "red" processors conduct a modified Jacobi sweep, and the black processors conduct a similar sweep but exploit the newly obtained values of the "red" processors.

While the above can easily be envisioned as being performed synchronously, the problem of synchronization of large arrays may prove insurmountable, thus we argued that asynchronous operation is preferable. An analysis of the speed up attainable from asynchronous conversion was made. We also proposed the Wavefront Array Processor architecture as a natural way to implement the asynchronous controls. As far as we know, this is the first time the WAP has been proposed for such a design.

References for Chapter 8

[1] Adams, L. and Ortega, J., "A Multi-Color SOR Method of Parallel Computation," Dept. of Applied Math, Univ. of Virginia, 1982.

[2] Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation*, vol. 31, No. 138, 1977.

[3] ——, "Guide to Multigrid Development," in Hackbusch and Trottenberg.

[4] Fisher, A. and Kung, H. T., "Synchronizing Large VLSI Processing Arrays," *IEEE Trans. on Computers*, vol. C-34, No. 8, Aug. 1985.

[5] Grosch, C., "Performance Analysis of Poisson Solvers on Array Computers," Supercomputers: 2, Infotech International, Maidenhead, 1979.

[6] Heller, D., "A Survey of Parallel Algorithms in Numerical Linear Algebra," *SIAM Review*, vol. 20, No. 4, Oct. 1978.

[7] Kung, S. Y., Arun, K. S., Gal-Ezer, R. and Bhaskar, R., "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computers*, vol. C-31, No. 11, Nov. 1982.

[8] Kung, H. T. and Lam, M., "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *Journal of Parallel and Distributed Computing*, vol. 1, 1984.

[9] Kung, H. T., "Two-Level Pipelined Systolic Arrays for Matrix Multiplication, Polynomial Evaluation and Discrete Fourier Transform," Workshop on Dynamic Behavior of Automata, Luminy, France, 1983.

[10] ——,"Systolic Arrays," Dept. of Computer Sci., Carnegie-Mellon Univ. Pittsburgh, Penn. 1984.

[11] ——, "Why Systolic Architectures," *Computer*, Jan. 1982.

[12] ——, "Systolic Algorithms," in *Large Scale Scientific Computation*, Academic Pr. 1984.

[13] Ortega, J. and Voigt, R. "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Review*, vol. 27, No. 2, June, 1985.

[15] Sameh, A., "On Jacobi and Jacobi-Like Algorithms for a Parallel Computer," *Mathematics of Computation*, vol. 25, No. 115, July, 1971.

[16] Stüben, K. and Trottenberg, U., "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications," in Hackbusch and Trottenberg, (see Bibliographic entry).

[17] Young, D., *Iterative Solution of Large Linear Systems,* Academic Pr. 1971.

9. Empirical Results

1. Introduction

Multigrid techniques have already been investigated by a number of researchers, (see, for example, the papers in *Elliptical Problem Solvers I and II,* [26]). The numerical experimentations of Gannon and Van Rosendale at ICASE in 1982, [9] is of great interest to us, and will be given special attention in this chapter, as it demonstrates the sort of versatility we will demand from the Multigrid algorithm. The work of some other authors will also be cited.

The reader should keep in mind the following thoughts as he reads through these sections:

1). Does the theoretical computing speed predicted by Chan and Schreiber (designated C & S, [4] and described in chap. 7) hold during the experiments, and does it meet our expectations for real-time signal processing?

2). What is the relation between computing speed, circuit area or layout and the dimension of the problem?

3). How complicated are the model problems used by the numerical experimenters relative to our own class of problems?

4). What kind of flexibility do we have in choosing the appropriate architecture for the MG algorithm?

5). How easily can these architectures be implemented into VLSI, and what are the various trade-offs between area and complexity?

Our own results will be in providing answers to the above questions. We will integrate the work of Gannon and Van Rosendale with that of Chan and Schreiber, [4], whose work was discussed in chapter 7, on the complexity of the Multigrid method.

We will put forward arguments that the Configurable Highly Parallel design methodology is a natural, as well as accessible, technology from our point of view.

We will examine the effect of dimension of the Zakai equation on computing speed, thereby extending the complexity analysis of Chan and Schreiber, [4].

Alternative computing architectures will be explored, using the two-parameter approximation of computing systems pioneered by Hockney, [15], and applying it for the first time to the Multigrid algorithm.

Also, we describe our own stability results involving wordlength and round-off noise considerations. And our own design for a direct solver, to be incorporated into the Multigrid architecture, will be described.

Main Results: presentation of arguments demonstrating the viability of the Multigrid algorithm for the Zakai equation; analysis of the effect of problem dimension on the Multigrid architecture, due to constraints imposed by intergrid connections; analysis of alternative architectures; a stability analysis for wordlength effects and round-off error propagation; and a design of a direct solver.

2. Three Variations of Multigrid Algorithms

Using a prototype model developed at ICASE, Gannon and Van Rosendale, [9], (now to be referred to as G & R), experimented with several variations of the Multigrid method. Beginning at the algorithmic level, they used a program essentially the same as described in chapter 7. All such variations were distinguished by different choices of parameters in FULLMG. Using the notation of Chan and Schreiber, whose work was discussed in chapter 7, (and who hereafter will be denoted C & S), three possibilities are, but were not limited to:

1). $c = 2, r = 1$

2). $c = 1, r > 2$

3). $c = 1, r = 1$

where, c controls the number of recursive calls procedure BASICMG makes to itself in order to solve the related coarse grid problems and $r = m/j$ controls the relative number of inner iterations performed in these recursive calls.

Thus when the smoothing sweeps are repeated m times, G & R are repeating them $r \cdot j$ times. The idea is that it may be more desirable to have a greater number of smoothing sweeps on the coarse grids. Since their approximating power is less than that on a finer grid. Therefore, if either $c > 1$, or $r > 1$, then more iterations by BASICMG will be performed on the course grid. Choice number 1) is typical of the type of multigrid algorithm considered in most of the theoretical literature. Van Rosendale considered the second choice on his own in an earlier paper, [11], while Brandt proposed the third example (among others) in [2].

While each of these three choices have about the same performance, i.e., the cost of reducing the error by a fixed amount for each iteration is about the same, their parallel implementation is quite different. On choice number 1), with $c = 2, r = 1$, we would expect poor performance since the recursion involved amounts to a binary tree traversal. The second choice, $c = 1, r > 1$, would be slightly better and the third choice, $c = 1, r = 1$ would be the fastest. Using reasoning less rigorous than C & S, G & R derived much the same results for the complexity analysis. The same is true for the subject of efficiency or processor unemployment.

3. Architectural Considerations

At the architectural level, we assume we are dealing with systems with relatively simple processors, each having local memory large enough to hold its own short program and the data associated with at least one node of the grid. Hence the processing elements are somewhat more complex than the systolic units discussed before. The basic distinction between the architectures discussed here in this chapter will be in terms of the structure of the inter-processor communication networks.

For a two-dimensional elliptic PDE, the ideal parallel architecture for the multigrid algorithm would be a layered network of processor grids with the finest being $n \times n$ where $n = 2^m + 1$ and the i^{th} level is a square grid of width $2^i + 1$. Iteration, interpolation and projection all involve local or nearest-neighbor communication.

If we consider PDE's in two space variables, one possible grid architecture could be that of fig. (7.3.1). We see that each grid is homogeneous while connected to a finer and/or coarser grid. Of course, the layout of this network on a chip would appear quite different, as we will see later. Also, other designs and layouts are possible.

Four basic classes of architecture can be discussed. We let R_j be the parallel time required to make a Jacobi (or any other suitable) relaxation sweep on grid j. Let I_j and P_j be the complexity of the injection and projection mapping Ψ and Π on the j^{th} grid. In general, the quantity R_j will be a sum of three terms: the time required to complete the numerical computation at each processor, the overhead time required by a processor to send and receive messages from each of its neighbors, and the actual transit time required for message passing.

The primary difference between both I_j and P_j and R_j is that the former require intergrid data movements. As for R_j itself, the relative time spent for arithmetic versus communication will vary from system to system. For example, the communication software in asynchronous MIMD systems usually involves complex buffering and handshaking protocols, and may dominate the arithmetic computa-

tion. Such overhead is lower in synchronous VLSI systems (like the systolic array), but intergrid data transmission lines are still long enough to make an appreciable difference.

The simplest model we can use is that all operations have the same complexity:

$$I_j = P_j = R_j = R_1, \qquad j = 1, ..., K \text{ grid levels.} \tag{9.3.1}$$

This model will hold for the architecture in fig. (7.3.1), but it will not hold for more flexible intergrid data switching networks to be discussed later. Call it the *constant cost model.*

How are such multigrid to configured electronically? We can take our cue from the Finite Element Machine (Storaasli, [28], Jordan, [17]), or the Massively Parallel Processor (MPP)). These systems are configured as planar ν by m nearest-neighbor connected grids of processors. Three ways of doing this are shown in figs. (9.3.1a)–(9.3.1c). Only figs. (9.3.1a)–(9.3.1b) are suitable for concurrent iteration while fig. (9.3.1c) represents the most efficient packing $\nu = 3(m-1)/2+2$ versus $\nu = 3m-1$. The most natural embedding, although not allowing for concurrent iteration, is clearly fig. (9.3.1c).

All three layouts have similar problems with intergrid data transfers. To explain, consider a linear sequence of m processors, each capable of nearest-neighbor read/write communication. Assume that processors 1 through $m/2$ each holds an item of data and we wish processor $2i$ to obtain the data from processor i where $1 \leq i \leq m/2$. This problem is the one-dimensional analog to intergrid data transfer. An *expand* operation is the sequence of steps needed to solve this problem. The inverse operation of transferring data from all even numbered processors to those labeled 1 to $m/2$ is called the *compress* operation. It is clear that the processors in such a nearest-neighbor network can perform an expand or compress operation in time $m/2$ by a set of parallel "bucket brigade" write/read steps.

A similar technique can be used for intergrid transfers. By applying the expand along each row and then along each column in parallel the data item in position

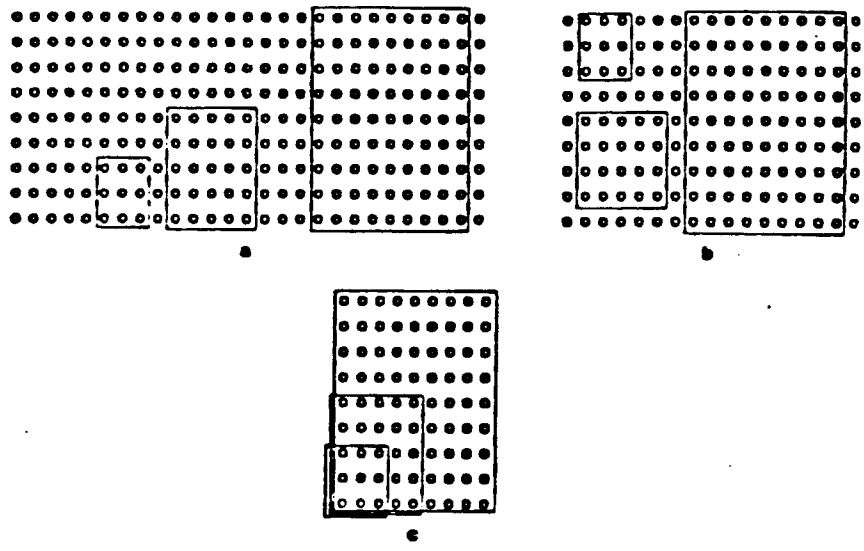


Figure (9.3.1) Three Multigrid Layout Schemes

(a). $\nu = 3m - 1$. (b). $\nu = 3(m-1)/2 + 2$. (c). $\nu = m$

(i, j) is moved first to $(2i - 1, j)$ and then to $(2i - 1, 2j - 1)$. When this technique is applied to the embedding in fig. (9.3.1a), all subgrids are expanded in parallel in $2m - 2$ steps. For the embedding in fig. (9.3.1c), the i^{th} subgrid is expanded in $2^i - 2$ step. The embedding in fig. (9.3.1b) requires a more elaborate algorithm, but can be done in $3m/2 - 2$ steps.

Since parallel relaxation is independent of the size of the grid, one has $R_j = R_1$ for $j = 1, ..., K$. Taking the remarks concerning communication above into account, one obtains,

$$I_j = P_j = R_1 + \kappa_1 2^j, \qquad R_j = R_1 \qquad \text{for embedding in fig. (9.3.1c),} \qquad (9.3.2)$$

$$I_j = P_j = R_1 + \kappa_1 n \qquad R_j = R_1 \qquad \text{for figs. (9.3.1a)–(9.3.1b),} \qquad (9.3.3)$$

where κ_1 is a small constant that refers to the time required for a read/write operation in R_1. Call the model in eqn. (9.3.2) the *linear cost model* since it grows

essentially linearly with the grid size, (which equals $2^j + 1$). The second eqn. may be called the n^{th} *size cost model.*

The expand and compress operation described above does not seem particularly efficient. An alternative has been proposed by Grosch, [12], and Brandt, [3], that involves the use of a shuffle-exchange network.[1] It turns out that the shuffle-exchange provides the exact connections that are simulated by the expand or compress operations. For the case of fig. (9.3.1a), the expansion operation can be performed by the shuffle network as depicted in fig. (9.3.2), which is illustrated for a row of processors. Note that processor i is directly connected to processor $2i - 1$ for $i \leq n/2$. Hence, for the embedding in fig. (9.3.1a) and (9.3.1c), we will have level j being mapped directly to $j + 1$.

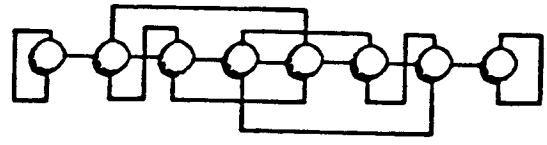


Fig. 9.3.2 Shuffle Connection on Eight Nodes

Now the complexity of using the shuffle-exchange depends upon the effect of signal propagation delay due to increased wire length. Inspection of figs. (9.3.1a) and (9.3.1c) indicates that the longest wire between level i and level $i + 1$ will grow as 2^i. The linear cost model in (9.3.2) therefore applies. We are of course assuming that propagation time grows as $O(l)$, where l is the length of the wire,[2] but we still

[1] See the section on this network in chap. 2.

[2] This is a nontrivial assumption, since the possibility of $O(l^2)$ delay cannot be ruled out. See the section, "The Physical Basis of Computation Time," in chap. 2.

might have a negligible contribution of signal delay, and cost model (9.3.2) could still be applicable. We will discuss this possibility shortly.

An extension of the foregoing ideas can be found in the use of a general permutation network for interprocessor communication. Such an approach is well-developed in the Ultracomputer or the Texas Reconfigurable Array Computer or TRAC system. The interconnection network for m processors will involve passing data through $\log(m)$ stages. Such data can be rearranged in a number of ways, although the full number of operations is a subset of the full permutation group S_N. (See "Shuffle-Exchange Networks" in chap. 2). The Ω^{-1}-network shown in fig. (2.4.2) can perform uniform shifts and the compress operation described earlier, while its inverse, the Ω-network, can execute uniform shifts and the expand operation. The resulting electronic network could appear as in fig. (9.3.3).

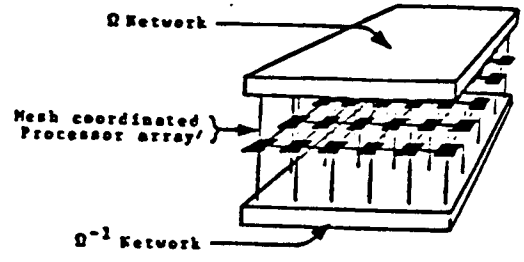


Figure (9.3.3) Mesh Ω System

Now Gannon , [11], argues that the Ω-network connections are not only preferable in general but are especially suitable for problems in which local refinement may be desired. These are cases when the solution of the PDE is expected to occupy only small portions of the fine grid, thus making most of the computations on this grid useless. One way to counteract this is to "track" the solution by first direct solving on the coarse in order to get a fix on the location of the solution, and then, by local refinement concentrate the finer grids only on that portion of the domain,

shutting down all the other processors. We might remark that Grosch, [12] arrived at the same conclusions in his elegant numerical simulations.

However, in our opinion, the Ω-network, as elegant and efficient as it is, may not suitable for our purposes, as it is too area-intensive, and one of our goals is to presumably limit the size of our computing system em. We would have to therefore choose from among the less rapid but more economical designs described so far. We should have known that eventually such a decision was inevitable, as the trade-off between speed and area, would put us over the limit we have set for ourselves in achieving a VLSI chip design. The Ω-network is simply not suitable for such a technology.

This is not to denigrate the views of Gannon, however, since his design really would be superior for the very large problems in structural mechanics and the other areas he had in mind.

The appropriate model for both standard multigrid and concurrent iteration, which was introduced in chapter 7, "The Complexity of Multigrid Methods," is

$$P_i = I_i = R_1 + \kappa \log n, \qquad R_j = R_1. \tag{9.3.4}$$

We will call this the *log-cost model.*

A direct VLSI embedding into silicon is possible for all of the architectures heretofore discussed. However, they each vary considerably in terms of cost due essentially to area requirements. The shuffle-exchange graph discussed earlier needs an area of order $O(n^2/\log^2(n))$ for n nodes (see the section on shuffle-exchange networks in chap. 2). Since the need for local refinement is meaningful only for relatively large arrays, (or large n), the corresponding area needed to support the permutation networks may grow beyond the bounds of conventional chip size fairly quickly.

On the other hand, the mesh embedding in fig. (9.3.4) which does not make use of permutation networks, requires area that grows $O(n)$ with the nodes, but the intergrid data-transfer algorithm is very complex.

As an example of a linear area embedding of the complete multigrid network in fig. (9.3.1), consider the layout provided by Purdue University's Configurable Highly Parallel or CHiP (*sic*) architecture, (Snyder, [27]). This has the advantage that all connections are completely implemented, which implies that the communication algorithm is simple. However, the length of the data paths for the coarse grids grow exponentially with the maximum level of refinement. In other words, inspection of fig. (9.3.4) shows that the coarsest grid has processors labeled as one's, and the finest grid has processors labeled with three's. Note how the data paths are longer in grid G_1 compared with G_3. Assuming the propagation delay is linear over a data path, the appropriate cost model will be

$$P_i = I_i = R_i = R_n + \kappa(2^{n-i} - 1), \tag{9.3.5}$$

where κ may again be small enough to be negligible. We will call this the *linear-VLSI model.* Now how small can κ be expected to be? An estimate comes from the section "The Physical Basis Of Computation Time," which puts $\kappa = O(10^{-6})$. Certainly small enough to be ignored as far as VLSI systems are concerned. Of course, for general systems, κ may be large enough to make a difference.

Ideally one would hope that an embedding of fig. (9.3.4) into silicon could have an area that grows as $O(n)$ with n being the number of processors, but that the length of the longest path between two processors would have an upper bound independent of m. Unfortunately, this is not possible.

We come now to a description of the numerical experiments performed by G & R. We provide a summary of the four models discussed so far in Table (9.3.1). The n^{th} size model was excluded from numerical experimentation due to its clumsy intergrid data transfer mechanism. The shuffle-exchange approach was favored over it. In most cases one would not expect that neither the algorithm nor the architecture would obey only one of these four models, but rather would be a hybrid of all

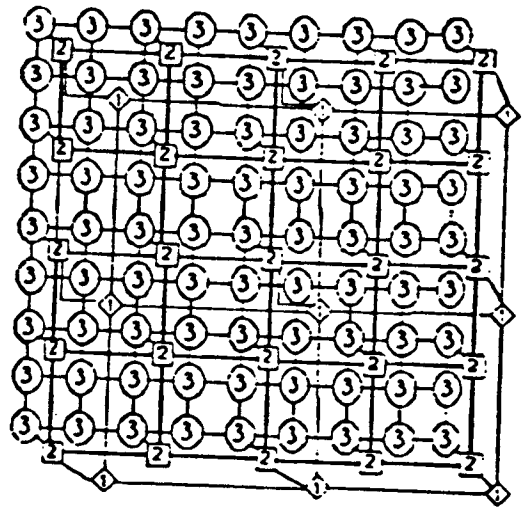


Figure (9.3.4) CHiP Processor Embedding

of them, yielding a relation of the form:

$$P_j = I_j = R_j = R_1 + \kappa_1 2^j + \kappa_2 \log n + \kappa_3 2^{n-j}, \tag{9.3.6}$$

for some constants κ_1, κ_2 and κ_3.

Model	Relations
Constant	$P_j = I_j = R_j = R_n; \quad j = 1, 2, ..., n$
Log	$P_j = I_j = R_n + \kappa n; \quad R_j = R_n$
Linear	$P_j = I_j = R_n + \kappa 2^j; \quad R_j = R_n$
Linear-VLSI	$P_j = I_j = R_j = R_n + \kappa 2^{(n-j)}$

Table (9.3.1) Communication Cost Models

The constant κ is hardware dependent.

G & R made no attempt to optimize the performance of the relaxation sweeps–

as they were only interested in a relative comparison between the models. All methods were tested using the same Jacobi inner iteration with the same suboptimal relaxation parameter. No consideration was given to whether Chebyshev acceleration or red-black S.O.R. might be a better choice. Instead, they argued that any such optimization was likely to improve the three methods in their study in about the same order of magnitude.

In making their relative comparisons G & R simply surveyed the operational counts each simulation program required, using the scale $R_j = 1$ count and $\kappa = 1$. The latter choice, which is orders of magnitude larger than a realistic value of κ, was made to reflect the differences one would expect to see on exceptionally large problems where signal speed variations due to wire length would be sure to show up.

Each PDE problem was solved on grids of maximal size n by n where $n = 2^m+1$ and m ranging from 3 to 6, i.e., n is of size 49, 225, 961, and 3969. In every case, the number of Jacobi relaxations in the inner iterations was kept at $j = 4$.

The three methods tested are:

1). Concurrent iteration, denoted CI

2). The $\log(n)$ time per outer iteration $(c = 1, r = 1)$ of Brandt denoted LO-MG.

3). The root(n) time per outer iteration method $(c = 1, r = 1.2)$ denoted RO-MG.

The three PDE model problems are all in two dimensions, and were drawn from an attempt to investigate three common characteristics of the numerical analysis of PDEs:

1). The rate of convergence to a point below the truncation error of the discretization.

2). The effect of domain shape.

3). And the effect of a non-self adjoint operator.

We provide only a qualitative description of the results of G & R, [9]. The reader is referred to their paper for more details. Comments on their work in the light of our own requirements will also be given.

Case 1: Convergence to Truncation Error

Let u be the exact solution of a PDE and let u_∞^n be the exact solution to the associated discrete system on an n by n mesh. Now if u_k^n is the approximation to u_∞^n at the k^{th} iteration of FULLMG, then given a positive $\Gamma << 1$, a reasonable measure of performance of the MG algorithm is the smallest value of k such that

$$||u_k^n - u_\infty^n||_2 \leq \Gamma ||u - u_\infty^n||_2. \tag{9.3.7}$$

In words, one says that the error of the discrete solution has been dominated by the truncation error.

As our model problem let O be the unit square in R^2 with one corner at $(0,0)$ and the opposite corner at $(1,1)$. We wish to solve,

$$\Delta u = -\left(\frac{1}{2\pi}\right) \sin\left(\frac{\pi}{2}x\right) \sin\left(\frac{\pi}{2}y\right), \tag{9.3.8}$$

with boundary conditions $u = 0$ for $x = 0$, or $y = 0$, and a vanishing normal derivative $\frac{\partial u}{\partial n} = 0$ for $x = 1$ or $y = 1$. This problem has a known exact solution which is

$$u = \sin\left(\frac{\pi}{2}x\right) \sin\left(\frac{\pi}{2}y\right). \tag{9.3.9}$$

Thus one could apply the convergence test in (9.3.7) to determine when to halt. (The computation time required to conduct the test was not included in the results). The value of Γ was chosen arbitrarily to be 10^{-4}.

G & R found that concurrent iteration, unlike the RO-MG and LO-MG algorithms, does *not* possess the property that the spectral radius of the MG operator is bounded independent of n. However, enough concurrency is allowed for in the Constant and Log-Cost communication models, that the concurrent iteration method

does very well in terms of computing time, and was generally the faster of all the models in this case study except of linear cost.

However, in the linear-cost model, fine grid interpolations exact a heavy price. Both the concurrent iteration and the LO-MG scheme do one course grid interpolation for each outer iteration. As the number of grid levels increases, the cost of fine grid interpolations begins to dominate the cycle time of each outer iteration. It follows that the method requiring fewer outer iterations, LO-MG, will be the fastest.

In the linear-VLSI model the burden is shifted to the course grid relaxations. As the RO-MG algorithm makes the largest number of coarse grid iterations, it is the slowest. In the linear-VLSI model, the cost of intergrid communication is trivial and concurrent iterations, which makes better use of parallelism than the LO-MG model, is moderately faster.

Remark: Concurrent iteration does very well but the sensitivity of the spectral radius of the MG operator with respect to grid size changes does not bode well for it. This phenomenon is apparently not well understood at present. See the section on Concurrent Iteration in chapter 7.

Case 2: Domain Shape

Many fast PDE solvers are designed to work best only for problem defined on the unit square. On more general domains, methods such as the fast Poisson Solver fare poorly. The ideal alternative would be a general iterative method that, while admittedly not as fast as the Poisson Solver, is nonetheless more adaptable to a wider range of domain shape.

Consider the problem,

$$\Delta u = 1.0, \tag{9.3.10}$$

on the domain formed by a square with one quarter section removed. We require that the normal derivative $\frac{\partial u}{\partial n} = 0$ on the boundary $\partial\Omega$. The solution near the re-entrant corner facing inward is constrained to vanish.

A different convergence scheme was used as the exact solution was unavailable. The method was allowed to run until machine precision was reached and the resulting solution $u_{\infty f}$ was recorded. The solution process was repeated until the L_2 error satisfied,

$$||u_k^n - u_\infty^n||_2 < 10^{-6}, \tag{9.3.11}$$

where the constant 10^{-6} was chosen arbitrarily. See G & R [9] for detailed results.

All three methods had roughly the same basic performance profile with respect to the four models. However, the spectral radius varied with grid size changes for *all* methods. While the RO-MG method has a radius that remains bounded (a property that can be proven to be the case), the LO-MG method reveals a substantial degeneration in convergence rate.

The Linear-VLSI architecture provided good performance for all three methods.

Remark: Multigrid methods appear to have technical difficulties for unusual grids and domains. We will hope to avoid such cases in our work as we will deal only with rectangular or hyper-rectangular domains. However, it is unknown how the Multigrid method will behave on a square of four or more dimensions. Work even

at three dimensions has been sparse, and will be discussed later.

Case 3: Non-Self Adjoint Operator

Time-dependent parabolic PDEs can also be solved by fast elliptic solvers. As an example, consider,

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x} + \alpha_m \Delta u. \tag{9.3.12}$$

Here, α_m is an artificial viscosity that was designed to obey the relation,

$$\alpha_m = O(h_m),$$

where h_m is the mesh-grid spacing for each of the experiments as m ranges from 3 to 6. It turns out that a well-known and standard method to avoid unstable discretization of problems of the form

$$\epsilon \Delta(\cdot) = f,$$

when ϵ is small in comparison with the mesh width h is the addition of artificial viscosity to ϵ. This is usually of the form $\beta = Ch$ yielding

$$\epsilon \leftarrow \epsilon + Ch.$$

Similarly, on the coarser grid we would have

$$\epsilon \leftarrow \epsilon + C2h,$$

and so on. Numerical experiments have shown this method to be very stable as well as being quite suitable for Multigrid algorithms.

Now at each time-step, we must solve the elliptic PDE,

$$\alpha_m \Delta u + \frac{\partial u}{\partial x} = f, \tag{9.3.13}$$

where f is some function of u on the previous time-step. Letting $\alpha_m = 2^{-m}$ and $f = 1.0$ on the domain O n fig. (9.3.5), we set $u = 0$ on ∂O as boundary conditions.

This experiment corresponds to advancing the time dependent problem solution one step. The detailed results can be found in G & R [9]. The determination when to halt the program was based on the sup norm: as many iterations were used as needed to reduce the error to obey,

$$||u_k^n - u_\infty^n||_\infty \leq 10^{-12}. \tag{9.3.14}$$

As before, the constant 10^{-12} was chosen arbitrarily.

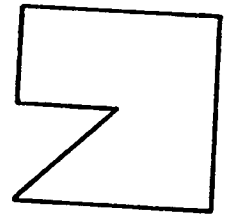


Fig. (9.3.5) Domain for Case 3

This time, as self-adjointness was not available, the inner iterative relaxations parameter was optimized to yield a reasonably good spectral radius when $m = 3$. Once again the spectral radius of the concurrent iteration algorithm degenerates as m increases, but the parallel time for the log and constant models continues to be better than for the other schemes.

Remark: Despite what would be an unusual domain from our point of view, the Multigrid algorithm fared rather well. Concurrent Iteration is seen to do well as far as time complexity is concerned, but its spectral norm sensitivity to grid size makes us hesitant in its use. It is an open question at present if similar phenomena would occur at higher dimensions even for hyper-rectangles. To see why this might be a problem, recall that our domain is largely a function of initial probability density, which we certainly want to be able to change at will. Changes in spectral

norm mean differences in computing times that may prove undesirable. This could be the ultimate trade-off for us with respect to Concurrent Iteration, speed vs. unpredictable convergence behavior.

Other Remarks

Remark 1: We recall from chap. 7 that C & S found the multigrid technique to take $O(\log^2 n)$ time units for computation for the fully parallel $c = 1$ version with as many processors as the number of grid points. How does this compare with the results of G & S? The number of grid levels for a problem of size n by n is roughly $\log_2 n$ (with $a = 2$ in C & S notation). If we consider the constant cost model, the computation time grows linearly with the number of levels. An upper bound of about $40 \log n$ for the computing time found by G & R multiplied by the time to complete one grid relaxation R yields an upper estimate of $40R \log_2 n$. Now the execution time for one relaxation R is about 20 time steps for arithmetic and communication overhead: so this yields $\sim 800 \log_2 n$ time units. In this case the C & S estimate was rather conservative. This is because C & S do not consider the finer points of architectural differences and the advantages in speed that can come from them. Only a worst case analysis was considered.

If we use the log cost model, and set one unit of communication equal to one arithmetic operation, then asymptotically the time is quadratic in the number of grid levels. Using the values in the table, a generous upper bound is $20 \log_2^2 n + 40 \log_2 n$. The C & S bound is of the form

$$(k\alpha/2)(\log_2^2 n + \log_2 n) \qquad (9.3.15)$$

where k is the number of FULLMG iterations and α corresponds to the smoothing sweeps, the computation of the coarse grid correction and the interpolation back to the fine grid, (see chap. 7). The factor in front of (9.3.15) is on the order of 10^2. Again, C & S have provided the more conservative estimate.

The central contribution of Chan and Schreiber is in their observation of the inherently self-referential quality of the MG algorithm, and the use of a recurrence relation to establish worst case complexity bounds.

Remark 2: G & S also point out the efficiency of the MG method over the systolic band solvers. In the latter case, computation times for problems of size n by n and in 2 dimensions are in the order of $8n^2$ time units, $O(w^2)$ processors, where the bandwidth is denoted by w.

While the multigrid method is time-efficient, especially as n grows larger, it certainly is not area-efficient, at least not in all the communication models while an $O(n)$ are growth rate is possible, as in the CHiP layout, shuffle-exchange graphs needed for some intergrid data transfer networks grow in area as $O(n^2/\log^2 n)$. This is the trade-off that must be considered if local refinement is desired.

For three dimensional problems, the computation time for systolic band solvers is of the order of $O(n^3)$ with n^4 processors based on the n by n by n cube. But multigrid techniques have computation times relatively impervious to changes in the dimension of the problem. Of course, wire length will be grow with dimension, and this will slow down the computing speed somewhat. This is a consequence of chip area growth. In fact, as we know that our computing time T is bounded from above, due to real-time processing constraints, we know from computation theory that

$$T \leq \textit{constant} \Rightarrow A = \Omega(f(d)), \tag{9.3.16}$$

where A is the chip area and d is the dimension of the problem. Now what is the design that yields the least area? Clearly it is the one in which the processor grids are embedded into each other, see fig. (9.3.1c). Now if we were dealing with a volume in R^d that had n^d points, this would be laid out on a $k \times k$ grid, where $k^2 = n^d$. But then the least area that the Multigrid design could have would be $\Omega(k^2) = \Omega(n^d)$. If we assume that the *density* of points remains constant, then n would remain fixed, and thus the area of the layout grows as $\Omega(e^{kd})$. Thus a chip

layout will only be possible for low dimensions.

4. On the CHiP Architecture

As already mentioned, the work at the Purdue University's Configurable Highly Parallel (CHiP) project, (Snyder, '82, [27],) is very appropriate for our needs. While we do not advocate this approach as the only circuit design methodology we should use, any prospective collection of techniques should at least have most if not all of the same features as the "Blue CHiP" architecture.

The central thrust behind Purdue's research effort is drive by the problem of how to configure or compose highly specialized unit processors into a larger system suitable for testing, layout and fabrication. In general, one could attach all the processor to a bus, but any benefits assumed by the algorithmic structure would be lost through wasteful interprocessor data movement. Alternatively, we could place the processors within the framework of a perfect shuffle or general interconnection networks, but this is highly wasteful of area. The CHiP approach allows the designer to retain locality ad regularity while still providing flexibility.

The CHiP architectures are characterized by a switch lattice connecting connecting a collection of homogeneous processors (denoted as PE's,) and in which the switching system is in turn controlled by a sequential computer (known as the controller.) The switch lattice is a regular structure formed from programmable switches connected by data paths. As of 1981, the lattice structure examined at Purdue could contain 2^8 to 2^{16} PEs, but large amplification in scale is expected, with at most "wafer level" fabrication remaining a possibility.

Broadcast commands from the controller drive the switches which in turn implement the configuration setting in the same location. With the entire lattice configured, the PEs begin synchronously executing the instructions stored in their local memory. The PEs do not keep track of the processors to which they are connected. Commands such as READ EAST and WRITE NORTHWEST determine

their actions.

If a new phase of the algorithm is then introduced, a new interconnection pattern is introduced. For our purposed, this would correspond to the phaseds of relaxation and intergrid data transfer.

The varieties of interconnection patterns yield a rich family of CHiP architectures. This diversity is further enhanced by the range of complexity allowed for by the PEs. Snyder , [27], argues that memory capacity is more important than functional capability, and hence argues that PE's used in an $n \times n$ array should have sufficient memory to store n data values.

The operation and design of the switches is very simple; they are circuit rather than packet switches, and they occupy area that is only a small factor larger than the minimum m^2, where m is the number of wires of the path pairs. Chip processing begins with the controller broadcasting to all switches to invoke a particular configuration setting. The number c of memory location for storing configuration settings will be small (< 16) and the degree d, which is the umber of incident data paths, will be either four or eight. The crossover number s varies between 1 (no crossover) to $d/2$, and we will generally have $s = 2$.

The total numbers of bits required at a switch is thus dcs, one for each direction for each crossover group for each configuration setting.

The design of the lattice is obviously important, and for the CHiP architecture, an important variable is the corridor width, w, which is the number of switches that intervene on the data paths between adjacent processors. Homogeneous arrays are distinguished as having uniform corridor widths but it is also possible to have a lattice with a variety of corridor widths.

To see the impact of a particular choice of corridor width in a design, we must first examine how a lattice accommodates an interconnection pattern graph. Two considerations come to mind, PE degree and edge density of complex interconnection paths. Snyder also points out that in the case of PE degree, if the pattern

graph vertices have degrees greater than 4 or 8 (the degree usually provided,) then larger PEs can be "constructed" by coupling the regular PEs to yield one PE with 6 degrees, (like the hexogonal design of S. Y. Kung.) Whether even larger vertex degrees can be accommodated is not clear from his system package description, (to be discussed below.)

Graph edge density is a second consideration, for it turns out that in order to host enough paths, some of the processors may have to remain disconnected, (and hence unused.) Increasing the corridor with obviously PE utilization. It also lowers the PE density. Since the number of PEs is linear in the are of high edge density of the lattice, this implies that PE utilization is inversely related to the are required to embed the pattern graph in the plane. Graphs requiring a nonlinear area will under-utilize PEs, since such layouts are composed mostly of wire–as in the case of shuffle-exchange networks.

Snyder's research team has concluded that up to constant multiplicative factors, the CHiP lattice with constant corridor width used the silicon area as efficiently as direct VLSI implementation for all pattern graphs; CHiP lattices with constant corridor width as well as such interconnection structures as shuffle-exchange networks cannot use the silicon area any more efficiently as direct VLSI implementation. More examples of such embedding can be found in Snyder, 1982, [27].

The CHiP architecture is also very fault tolerant. If a faulty PE, switch or data path is detected, one can simply configure around it. Consider wafer level fabrication. The wafer is viewed as an enormous chip, and it is accepted if, after testing, a regular $k \times k$ sublattice is found to be functional. On board mapping circuitry could map the addresses of the logical PEs and switches onto the functional elements of the wafer. Snyder, (1982, [27],) describes a scheme in which many processors are linked together to formed a redundant unit. The number of such processors needed could be determined by the yield rate of a wafer fabrication process, which is determined by what is known as the Price model. He found that

the area lost to redundancy is more than made up for by the increased recoverability of the processor blocks.

Finally, we mention that package programs are available that can simulate CHiP configurations. The package is called POKER, and is available from the Univ. of Washington, where Snyder now resides. It is an interactive graphics program that allows the user to experiment with a variety of switching patterns. Timing, fault tolerance and other issues can then be studied. The package is VAX compatible and appears to be quite user-friendly.

We conclude that the CHiP approach to system design should be seriously considered as a design tool for Multigrid computing systems.

5. Design of the Direct Solver

In much of Multigrid algorithmic design, the cost of direct solving on the coarsest grid is almost trivial in comparison with the rest of the computations. This is clearly true in the case of the classical design in which the coarse grid has been reduced to a single point. Of coarse, in reality we would not wish to do this, but instead would use three or four grids, so the coarsest grid would still be a sizeable grid of points, albeit much smaller than the others.

Now if we were to design a Multigrid program to be run on a regular mainframe, the direct solver would just be some library routine for solving linear systems directly, such as *LU* decomposition or Gaussian elimination. However, as we are designing for processor arrays, certain constraints will be placed upon us. For example, the cost of circuit fabrication can be reduced if we maintain regularity of the layout. This behooves us to attempt to design a direct solver for a n_1^d–dimensional linear system whose information has been stored in a certain format in an array of n_1^7 processors. (Here, n_1 refers to coarsest grid, just as a multigrid scheme of K levels has n_K^d points on its finest grid. We are using the notation of Chan and Schreiber; see [4] or chap. 7.)

The approach we are taking is not the usual one in the design of custom-made architectures. As can be seen in the work of Miranker, [24], one begins first with an algorithm, which then leads to a communication graph, (which is an acyclic digraph,) and from this to a processor array. Here we are given the processor array in advance, with the requisite information distributed throughout the array in a predetermined format. This format and architectural structure is a consequence of the coarse grid being merely a subsystem of a much larger design. Our desire to maintain regularity has required us to remain in this framework.

In what way is this information distributed? If a point is associated with a processor, then that processor also has all other numbers that will be multiplying the (temporary) value of the solution at that point. The processor also has all other data corresponding to that point. Thus if we are solving the linear system,

$$Ax = b, \tag{9.5.1}$$

where x is a vector associated with a grid of points, then the component x_i of x will be in processor P_j, and the i^{th} column vector of A along with b_i will be in this processor too. This is the case for *all* the grids, not just the coarsest grid.

We will now describe a simple direct solver that exploits the use of LU decomposition. We saw this technique used before in the scalar Zakai solver. It was especially appealing because our matrix is independent of time and hence the decomposition can be done off-line, i.e., it is precomputable.

To quickly review the technique, we have the system $Ax = b$ and we set $A = LU$, where L is a lower unit triangular matrix, and U is a upper triangular matrix.

Methods of obtaining such decompositions are worthy of review. We have,

$$a_{ij}^{(1)} = a_{ij}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} + l_{ik}(-u_{kj})$$

$$l_{ik} = \begin{cases} 0 \text{ if } i < k, \\ 1 \text{ if } i = k, \\ a_{ik}^{(k)} u_{kk}^{-1} \text{ if } i > k. \end{cases}$$

$$u_{kj} = \begin{cases} 0 \text{ if } k > j, \\ a_{kj}^{(k)} \text{ if } k \leq j. \end{cases}$$

The first observation we make is that if $a_{ij} = 0$ for $i \neq j$, then $l_{ij} = u_{ij} = 0$, which is verified by inspection of the above recurrence relations. This fact will prove useful to us. (Also note that our discretization ensures that $a_{ii} \neq 0$ for each i.)

Now one solves the decomposed linear system in two steps. If we begin with

$$LUx = b, \tag{9.5.2}$$

we first find

$$Lv = b, \tag{9.5.3}$$

and then solve for

$$Ux = v. \tag{9.5.4}$$

The two systems are solved similarly, as they are both triangular systems. We remark that we have already encountered this problem in the last section of chapter 2 for the scalar Zakai solver. There we were dealing with a tridiagonal system, and an elegant design with a data stream pipelined into three processors was described that could arrive at the solution. Here, for our problem the bandwidth will be much larger, but the sparsity of the matrix will be used in our favor. Also, the pipelined design will conflict with the array design already imposed upon us by the Multigrid architecture. In the interests of preserving regularity, we will use an array method for constructing the direct solver.

Now recall that the formula for solving

$$Lv = b \tag{9.5.5}$$

is

$$y_i^{(0)} = 0,$$
$$y_i^{(k+1)} = y_i^{(k)} + L_{ik}vk, \tag{9.5.6}$$
$$v_i = (b_i - y_i^{(i)})$$

To see what is happening more clearly, we will consider the case of two dimensions, but extensions to higher dimensions are possible. In this special case, we have a planar array, and, for additional simplicity, assume we are dealing with the case of one processor per point. Starting at the upper left hand corner of the array, we number the points left to right, so v_i corresponds to the i^{th} point/processor numbered in this way.

Now since $L_{ik} = 0$ when $(I + \Delta tA)_{ik} = 0$ for $i < k$, the non-zero entries of L_{ik} correspond to the north and west nearest neighbors. (If mixed partial are present in our PDE, the Northwest point is included as well.) Therefore, for each i, the new value of v_i is calculated as above by only communicating with nearest neighbors. Thus the Principle of Locality is preserved. Recall that $(I + \Delta tA)$ is an $n^d \times n^d$ matrix, but it is highly sparse, and of course, we will exploit this fact with respect to memory storage. The non-zero entries of the column vectors of $(I + \Delta tA)$ of L are precisely those values that correspond to the nearest neighbors of a given point. For example, the i^{th} component of u_{n+1} in the equation,

$$(I + \Delta tA)u_{n+1} = Fu_n \tag{9.5.7}$$

is used in calculation with all the nearest neighbors at the i^{th} labeled point. Now in the case of no mixed partials in the PDE, there are at most $2d + 1$ non-zero entries in the column vectors of $(I + \Delta tA)$ or half that number in L. If mixed partials are present, we have at most 3^d non-zero entries. It is the non-zero entries associated with the nearest neighbors of point i that can be stored in processor i. And since the matrix U will also be employed, we will need enough memory cells to store these values. Once again we see that not having mixed partials leads to a linear

growth with respect to dimension for our memory cells. An exponential growth rate is required for the case of mixed partials.

With the non-zero entries of the matrix stored in the appropriate processors, we are now in a position to solve (9.5.5). We state that a diagonal "wavefront of computations" can be constructed, a term borrowed from Kung, and the Wavefront Array Processor (or WAP) that he designed, (see the section on this topic in Chapter 2.) To begin, the upper left hand corner point of the array v_1, is simply given the value of b_1. Its neighbors immediately to the right and below (East and South), take on the values,

$$\begin{aligned} v_2 &\Leftarrow L_{21}v_1 + v_2 \\ v_{n+1} &\Leftarrow L_{n+1,1}v_1 + v_{n+1}. \end{aligned} \tag{9.5.8}$$

Thus, in general, we have the following situation, as seen in this cross section:

$$\begin{matrix} * & * & * \\ * & * & 1 \\ * & 1 & 2 \end{matrix} \tag{9.5.9}$$

Here, the "*" corresponds to points whose v_i values have already been calculated. The diagonal row of 1's are points that will be next in line to be computed, having received the computational wavefront of the * points. This in turn leads to the pattern

$$\begin{matrix} * & * & 1 \\ * & 1 & 2 \\ 1 & 2 & 3 \end{matrix} \tag{9.5.10}$$

where the 1's have been computed values, and the 2's are ready for processing.

The diagonal wavefront continues until it reaches the opposite end of the array. Then the system

$$Ux = v \tag{9.5.11}$$

will be solved in an analogous way. Note that the v_i values must change places in the b_i memory cells.

The diagonal wavefront for system (9.5.11) moves in the opposite direction as the first, using nearest neighbor values from the south and east.

We might remark that when the second wavefront reaches the opposite end, that will be the signal for the coarsest grid to interpolate its data on up to the next finer grid. Hence a clock pulse will be needed to spur this action, but it need not be absolutely precise, i.e., no clock skew among the processors. For the relaxations are done asynchronously, so we can allow for some timing problems.

In fact, the term "computational wavefront" that was employed earlier suggests that we should implement our direct solver asynchronously. This is not necessary at least for low dimensions. Suppose that $d = 2$, and that our finest array is composed of 100×100 points. If we used four grid levels with half spacing we would have a 12×12 processor array for the coarsest grid. Kung's SPICE analysis of clock skew , [18], (see the section on "The Synchronization of Large Arrays" in chapter 2) yields virtually zero timing delay with this array size. Therefore, if desired, we could use a systolic array for the direct solver.

Of course, for larger dimensional problems equidistant nearest neighbor data paths will not be possible, so asynchronous implementation will probably be desirable, and the WAP protocols and language may be ideal.

To time the algorithm, we see that for an $n \times n$ array, we need $2n$ time units to calculate the first wave, (the factor of two comes from having to invoke data transfers from the two north and west nearest neighbors. We would have a factor of three if the northwest neighbor were included, as in the case of mixed partials.) Once the first wave reaches the opposite end, the second wave for (7.5.11) begins, which also takes $2n$ time units. Therefore, the total time for the direct solver of n_1^2 points is $4n_1$, which is competitive with the pipelined one-dimensional array discussed in chapter 2. And we know from the work of the complexity theorists (e.g., Miranker, [24]) that this is the minimal time we can have. Our trade-off here is in choosing a design which is less processor intensive (only $2d+1$ processors would be needed in a pipelined version) and in preserving regularity, which is always the better choice with respect to fabrication cost.

In higher dimensions, when we are dealing with hyper-rectangles of n^d points, we have a possibly non-homogeneous $k \times k$ processor array where $k = n^{d/2}$, (this is the one processor per point design. Thus the direct solver works in $O(n_1^{d/2})$ time. Here n_1 is small enough that the $O(\log n_K)$ time needed for the overall Multigrid algorithm is the predominant term.

6. Remarks on the Stability of the Multigrid Method

In our discussion of the robustness of MG techniques for this equation, we will concentrate on the 2-grid level case, since convergence here is all but sufficient for convergence in the general set-up.

We assume for the moment that we have found the maximal eigenvalues of S_h and hence $\mu(h,\omega), \mu^*$ have been found. Recall that the Multigrid operator of the 2-grid case is

$$\mathrm{M}_2 = S_h^{j_2}(\omega)(I_h - I_{2h}^h (L^{2h})^{-1} I_h^{2h} L^h) S_h^{j_1}(\omega) \tag{9.6.1}$$

An important issue for us is the question of how robust M_2 is relative to small departures in L^h from its true value. The following Lemma will be of help in answering this question.

Lemma 9.6.1: Let A be an $n \times n$ matrix where we assume,

$$\begin{aligned} \max_v \frac{||Av||}{||v||} &= ||A||_{max} \\ \min_v \frac{||Av||}{||v||} &= ||A||_{min} > 0 \end{aligned} \tag{9.6.2}$$

in some suitable norm. In particular assume $\rho(A)$ is known. Also, let B be an $n \times n$ matrix.

Then if λ is the maximal eigenvalue of A, with

$$\begin{aligned} Av &= \lambda v \\ (A + \delta B)w &= \mu w \end{aligned} \tag{9.6.3}$$

for $\delta \in R$, then $|\lambda - \mu| = O(\delta)$.

Proof: We have

$$\begin{aligned} wAv &= \lambda(w,v) \\ v(A+\delta B)w &= \mu(w,v) \end{aligned} \tag{9.6.4}$$

Thus

$$\delta(v, Bw) = (\mu - \lambda)(w,v),$$

which can be written

$$\begin{aligned} |\delta|||v||\,||w||\,||B||_{max} &\geq |\mu - \lambda|\,|(w,v)| \\ &\geq |\mu - \lambda|\,||w||\,||v||\,A_{min}|\lambda| \end{aligned} \tag{9.6.5}$$

where we have used

$$wAv = \lambda(w,v)$$

$$||A||_{min}||w||\,||v|| \leq \lambda|(w,v)|$$

Thus

$$\frac{|\delta|\,||B||_{max}}{|\lambda|\,||A||_{min}} \geq |\mu - \lambda|. \tag{9.7.6}$$

This proves the lemma. **Q.E.D.**

The lemma merely says that small changes in A produces small changes in its spectral radius, as expected. Hence, in the case of the Multigrid operator, if the spectral radius is already small, then changes in the operator are also small, and convergence behavior will be left unaffected. More specifically, the number of MG cycles that were adequate for the model problem, will not have to be changed for a small perturbation in the problem. For example, we might be dealing with equations that are dependent on some parameter. Choosing one value of the parameter would give us a model problem for a class of equations. Or we choose choose the average values of the coefficients of the PDE and examine the Multigrid behavior for the resulting equation with these average, and constant coefficients. We could then optimize the program for this one problem, and since the Multigrid method is robust enough, small parametric changes should make little difference in our design. This is of great importance to us since, in general, one would decide to halt a numerical scheme when some global behavior is obtained, such as successive differences

reaching a certain tolerance level. But such a test would take up too much if not most of the time we have allotted to ourselves for real-time computation. We must therefore be able to pre-set the number of MG cycles needed for a pre-determined convergence level, and still be assured of stability of our design with respect to small changes in the model problem.

The same goes for other features in the program such as choice of grid-size, injection and interpolation schemes. It should be clear that as long as the changes are small, no problems will occur here, so we concentrate on the MG cycle number.

Now let us assume, that we already have good convergence behavior with our program, which we assume to be of the two-grid level design, as this gives us a good estimate for $||M_k||$. In particular, we either know or can estimate the spectral radius of M_k, and based on this have determined the maximum number of MG cycles needed for convergence for each time-step of some parameterized Fokker-Planck model problem. Now we will replace L^h by $L^h + \Delta B$ where B is the matrix with the appropriate entries so that the resulting matrix is a parameterized, with respect to Δ, version of the Fokker-Planck equation. equation.

Now we already know that

$$||M_2|| \leq ||S_2||^{j_1+j_2}||I_2 - I_1^2(L^1)^{-1}I_2^1L^2||. \tag{9.6.7}$$

So we first concentrate on S_2, which we assume, for the sake of simplicity, to be the Jacobi operator. Generalizations to other schemes will be obvious although somewhat tedious.

Now recall that the spectral radius of the Jacobi method is determined by

$$(I_2 - \omega D^{-1}L^2)(\cdot), \tag{9.6.8}$$

where $D_{ij} = \delta_{ij}L^2_{ij}$, i.e., D is the "diagonal portion" of L^2. The relaxation parameter ω is in $(0,1)$.

This implies that the diagonal matrix D associated with S_k and utilized in the

formula,

$$S_2(\cdot) = (I_2 - \omega D^{-1}L^2)(\cdot) + \omega D^{-1}f^2, \tag{9.6.9}$$

will be replaced by a another matrix of the form:

$$D_{ii} \to D_{ii} + \Delta B_{ii}. \tag{9.6.10}$$

Now we will concentrate only on that portion that determines the spectral radius, namely,

$$\begin{aligned}(S_2)_{ij} &= I_2 - \omega\Big(\frac{L^2_{ij} + \Delta B_{ij}}{D_{ii} + \Delta B_{ii}}\Big) \\ &= I_2 - \omega\Big(\frac{L^2_{ij}}{D_{ii}}\Big(\frac{1}{1 + \Delta B_{ii}/D_{ii}}\Big) \\ &\quad + \frac{\Delta B_{ij}/D_{ii}}{1 + \Delta B_{ii}/D_{ii}}\Big)\end{aligned} \tag{9.6.11}$$

Using the geometric series, this can be written,

$$\begin{aligned}S_h(\omega)_{ij} = I_h &- \omega\frac{L^h_{ij}}{D_{ii}}\Big[1 - \Delta B_{ii}/D_{ii} + ...\Big] \\ &- \omega\frac{\Delta B_{ij}}{D_{ii}}\Big[1 - \Delta B_{ii}/D_{ii} + ...\Big]\end{aligned} \tag{9.6.12}$$

Assuming Δt is small enough, we can neglect all higher order terms other than the first. Thus, the new relaxation sweep is

$$||S_h^{new}(\omega)|| \le ||S_h^{old}(\omega)|| + |\Delta|\,||K||, \tag{9.6.13}$$

where $||K|| = O(||B||\Delta^2 x)$, since the factor of $1/D_{ii} = O(\Delta^2 x)$. It follows that the new spectral radius of $S_h^{new}(\omega)$ will differ from the old one by $O(||B||\Delta\Delta^2 x)$. Since B represents the changes imposed on L^h, we can clearly choose Δx small enough to offset to maintain stability.

Continuing in this way, we turn our attention to

$$I_2 - I_1^2(L^1)^{-1}I_2^1L^2. \tag{9.6.14}$$

Replace L^2 by $L^2 + \Delta B^2$ and L^1 by $L^1 + \Delta B^1$. Then

$$\begin{aligned}(L^1)^{-1} &= (L^1)^{-1}(I_1 + \Delta B^1(L^1)^{-1})^{-1} \\ &\approx (L^1)^{-1}(I_1 - \Delta B^1(L^1)^{-1}).\end{aligned} \tag{9.6.15}$$

Note that $||B^1|| = O(1)$ and $||(L^1)^{-1}|| = O(h^2)$. Using (9.6.14) and (9.6.15), and working all the algebra out, we see that we still have $O(\Delta)$ changes in the spectral radius.

We conclude that if k cycles of the full Multigrid are needed to solve the Zakai equation with $\Delta = 0$, then k cycles are sufficient for small values of Δ.

We have therefore used a model problem in exactly the same way as MG practitioners would do. By working out the number of cycles needed for this problem, we could also design the program to be robust enough for small changes without having to change the design parameters. Of course, this would be further verified by numerical experimentation.

The resulting robust or stable behavior of Multigrid methods is well known to the MG practitioners with problems involving even more dramatic changes than those contemplated in this section. However, the faith in stability is based on a wide range of numerical experiments, rather than on an exhaustive theoretical analysis.

The heuristic reason as to why such stability is to be expected is due to the spectral properties of $\mathbf{M}_2$ whose spectral norm is insensitive to small changes in L^h or even in the domain of the problem. Also the influence of a given relaxation technique is nearly the same for neighboring problems. This is mainly due to the "local" nature of relaxation processes in general.

Wordlength Considerations

When we store the matrix $L^h = (I + \Delta t A)$, into our processors, we will have to truncate the entries in our matrix due to limitations on bit size imposed by our processor design. What effect will this have on stability. Suppose the *true* matrix is L^h but, because of wordlength truncation, we actually store $\bar{L^h}$ into the processors, where

$$\bar{L^h} + \Delta L^h = L^h. \tag{9.6.16}$$

We know from previous arguments that a change in L^h of the order of ΔL^h produces a change in the Multigrid operator of the order of ΔL^h.

This, of course, will incur a change in the spectral norm of the MG operator, and the question for us is whether its norm can become greater than one. Thus, suppose the true MG operator is $\mathbf{M}_k$ and that because of wordlength limitations, we really obtain $\bar{\mathbf{M}}_k$, where

$$\bar{\mathbf{M}}_k + \Delta \mathbf{M}_k = \mathbf{M}_k. \tag{9.6.17}$$

This induces a change in the spectral radius of the form,

$$\lambda_{max} + \Delta\lambda_{max}. \tag{9.6.18}$$

Now, because of wordlength limitations,

$$(\Delta M_k)_{ij} \leq q/2 \tag{9.6.19}$$

where q is the quantization bound. In our case, $q = 2^{-b}$, where b is the number of bits in a word.

Although the Multigrid operator can be written as a huge matrix on the order of $n^d \times n^d$, it is only the non-zero entries that will create any problems with regard to wordlength limitations. Now each row vector of $\mathbf{M}_k$ that is dotted into the vector u_k contains at most 3^d non-zero entries corresponding to

the nearest neighbors of each component of u_k, in the case where mixed partials are present. (This is the case of having no mixed partials in our PDE.) Thus

$$||\Delta \mathbf{M}_k||_\infty \leq \frac{(3^d)}{2} 2^{-b}. \tag{9.6.20}$$

Now we make use of a result from Faddeeva, [6]. If a matrix A is perturbed by ΔA, then the corresponding change in the maximal eigenvalue of A can be estimated by

$$\Delta\lambda = v^T \Delta A \Delta u, \tag{9.6.21}$$

where u is the eigenvector of the matrix A, and

$$\begin{aligned} Au - \lambda u = 0, \quad ||u|| = 1 \\ A^T v - \lambda v = 0, \quad (v,u) = 1. \end{aligned} \tag{9.6.22}$$

Using this, the change induced by ΔM_k on the maximum eigenvalue λ_{max} is

$$\lambda_{max} + \Delta\lambda_{max} = \lambda_{max} + v^T \Delta M_k u \tag{9.6.23}$$

where u and v obey eq. (9.6.22).

It follows that for stability we must have

$$|\lambda_{max} + v\Delta \mathbf{M}_k u| \leq |\lambda_{max}| + |v^T \Delta \mathbf{M}_k u| \leq 1 \tag{9.6.24}$$

and using equations (9.6.20) and (9.6.24) we have, upon taking logarithms,

$$b = -(1 + \log_2((1 - \lambda_{max})/3^d) \geq 0. \tag{9.6.25}$$

This formula gives the minimum number of bits that are needed in a work so as to ensure stability of the Multigrid operator as a function of dimension d. We see in fig. (9.6.1) that even for large d say, $d \approx 10$, and rather large λ_{max}, eight bits should be sufficient to ensure stability. We see that despite the size of the matrix in our algorithm, its sparsity allows for low wordlength. Of course, greater accuracy may require 16 bits, but recall that the one-dimensional Zakai solver, described in the last section of Chapter 2, also needed 8 bits.

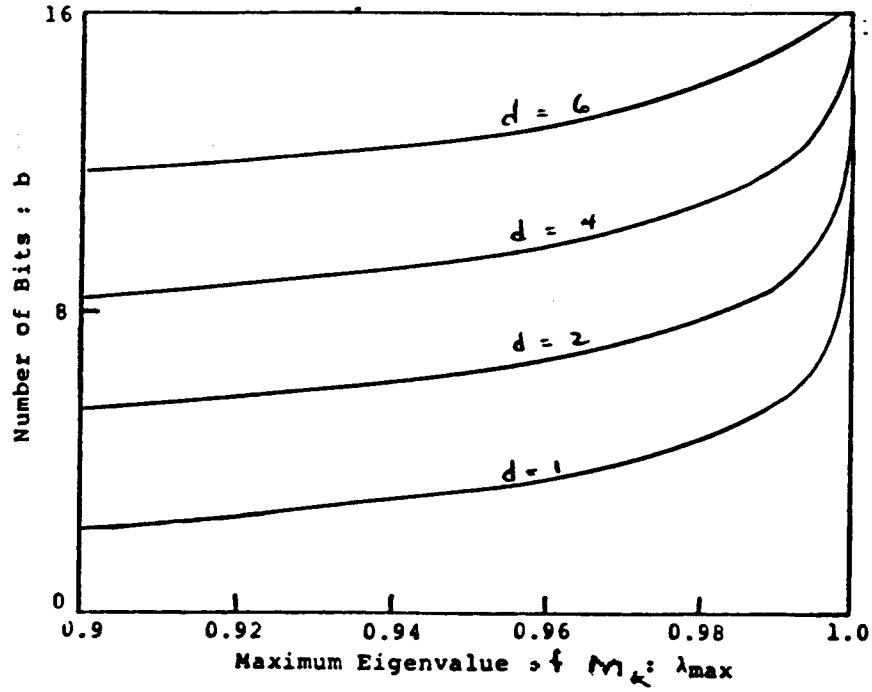


Figure (9.6.1) Minimum Wordlength as Function of Dimension

Preliminary Round-off Error Analysis

Because of the iterative nature of the Multigrid method, a preliminary analysis of round-off error should be fairly straightforward. Using the fact that $\mathbf{M}_k$ is constant while f^k is recalculated for each time-step, (as it depends on $\Delta y(t)$), we have,

$$u_{p+1} + \Delta u_{p+1} = \mathbf{M}_k(u_p + \Delta u_p) + I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1}(f^k + \Delta f^k). \qquad (9.6.27)$$

Here, the subscript p corresponds to the p^{th} iterate that has occurred within a single time-step. The error, due to round-off noise is

$$\Delta u_{p+1} = \mathbf{M}_k \Delta u_p + I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1} \Delta f^k. \qquad (9.6.28)$$

Now there are r cycles of the Multigrid algorithm for each time step, so if u_0 is our initial guess at the start of a complete cycle, then, at the end of r cycles we would

have,

$$\Delta u_r = \mathbf{M}_k^r \Delta u_0 + \sum_{p=0}^{r-1} \mathbf{M}_k^p (I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1} \Delta f^k. \qquad (9.6.29)$$

(Remember that Δf^k remains a constant during the complete cycle as it depends only on $\Delta y(t)$, and is calculated only at the start of the cycle.

Taking norms of (9.6.29), we have

$$||\Delta u_r|| \leq ||\mathbf{M}_k||^r \Delta u_0 + \left(\frac{1 - ||\mathbf{M}_k||^r}{1 - ||\mathbf{M}_k||}\right) ||I_{k-1}^k (L^{k-1})^{-1} I_k^{k-1}|| \, ||\Delta f^k|| \qquad (9.6.30)$$

But $||\mathbf{M}_k||^r \approx 0$, thus the round-off noise is $O(\Delta f^k)$ for each complete cycle. Fortunately, the calculated of f^k is trivial relative to all other calculations, and it is done completely in parallel. If we assume that the noise incurred by a calculation to be a random variable with a uniform distribution between $(-a, a)$, then we find that $\Delta f^k = O(a)$ since each component of f^k is calculated by a single multiplication independently of all others, and therefore incurs very little error.

7. The Effect of Dimension on Real-Time Processing

One of our goals to to determine whether the MG algorithm is suitable for real-time processing in higher dimensions, e.g., $d \geq 3$. We know from the results of Chan and Schreiber (hereafter denoted C & S) that the complexity results of Multigrid are affected by dimension in the choice of γ. This parameter in turn tells us how many processors are to be used, which is proportional to n^γ. In addition, the work of Gannon and Van Rosendale (or G & R) suggests that the actual implementation of the architecture will have a possibly dramatic effect on the speed and performance of the system, due to communication complexity and signal delays brought on by somewhat extravagant wire length. Consequently, we will combine the work of both research groups by demonstrating exactly where the results of G & R can be incorporated into the C & S framework. Upon doing this, we can study the effect upon computing time by an increase of dimension.

The most important observation we will make in this section is that processor array technology is limited by the number of interprocessor connections that can be

feasibly designed for a chip or any other electronic structure. This is true both for today as well as for the foreseeable future. The most reasonable maximal number of interconnections is about eight, although an even higher number might be possible. Eight interconnections correspond, in the notation of C & S, to $\gamma = 4$, (the rule being that each processor is connected to its 2γ nearest neighbors.) If we restrict ourselves to a particular processor array design, which means that γ is fixed, then we are automatically limited as to how high a dimension can be before real-time processing becomes impossible.

For purposes of illustration, suppose we intend to have one processor per point. Now in one dimension, a point has two nearest neighbors, in two dimensions a point has a minimum of four such neighbors, and possibly eight with which it will "communicate" or share data. Similarly, in three dimensions a point will have a minimum of six neighbors. Clearly the rule is that there are $2d$ nearest neighbors to a point in a d-dimensional space.

Now we can map a d-dimensional hyper-rectangle to a two-dimensional plane; this will not be the problem. Rather, it is the interprocessor connections that will limit the dimensional size of the problems we wish to consider if we want to keep the one point per processor design. For if $\gamma = 4$, we have 2γ interconnections per processors, i.e., each processor is connected to eight of its nearest neighbors. (This rule follows from the work of C & S.) This implies that $\gamma \geq d$, in this particular design, and so we cannot go any higher than $d = 4$. This of course, is the most parallel and therefore the fastest of all of the C & S designs. To choose another design is to incur a slower computing time.

These facts are made especially transparent if we think of each point on the grid as a vertex of a communication graph, which we know from our previous studies is an acyclic digraph. Then 2γ corresponds to the degree of the vertex and implies that each point must communicate only with each of its nearest neighbors during relaxation, interpolation and injection.

We reiterate that it is unlikely that we will see processor arrays with more that eight interconnections between processors. A greater number would probably imply costly communication over much greater distances than those incurred by nearest neighbors. Thus the principle of locality would be violated.

The solution to this is simply to use processors that accommodate more than one point. This would occur precisely when $d > \gamma$. We could then have a processor grid of n^γ processors, in which each processor is still connected to its 2γ nearest neighbors. If we set $n = n_K$, with the lower grids having n_k^γ, we see that each processor on processor grid P_k oversees $n_k^{d-\gamma}$ points. Now this means that a single point in a given processor has direct access to all the other points it shares with its host processor, plus those in the other processors. This yields a total of

$$n^{d-\gamma} - 1 + 2\gamma n^{d-\gamma} \text{ points.}$$

We also know that this same point has $2d$ nearest neighbors in the problem space. Suppose that γ and n were held fixed, but d was allowed to grow in size. Would our architecture still accommodate this growth and ensure that our original point had direct access to all those points with which it will communicate? For this *not* to be true we would have to have, for $\gamma < d$,

$$n^{d-\gamma}(2\gamma + 1) - 1 < 2d. \tag{9.7.1}$$

But this is impossible since

$$\frac{(2\gamma+1)}{n^\gamma} \geq \frac{(2d+1)}{n^d},$$

there being equality only when $d = \gamma$.

We now make an attempt to incorporate the work of C & S into this discussion. We know from their research that for BASICMG, the computing time obeys

$$T(n) = \begin{cases} \beta(a^p/(a^p - c))n^p + O(n^{\log_a c} \text{ if } c < a^p \\ \beta n^p \log_a n + O(n^p) \text{ if } c = a^p \\ O(n^{\log_a n} \text{ if } c > a^p \end{cases} \tag{9.7.2}$$

where $p = d - \gamma$. Let us make some estimates of these parameters. We begin first with β, which we recall obeys relation (7.3.16),

$$\beta = (j + m + s)t,$$

where

$j = m \approx 2$ is the number of relaxation sweeps used at different points in the BASICMG program, (consult chap. 7),

s turns out to be the parameter that is really being explored by the work of Gannon and Van Rosendale, as described in this chapter. It is the ratio of time devoted to injection, interpolation, and intergrid data transfers to the time required for one relaxation sweep.

Now the key to the work of G & R is that they relate the time required for injection, interpolation and data transfer in terms of the time needed for relaxations. For example, in the ideal case,

$$I_j = P_j = R_j = R_K \text{ for } j = 1, 2, ..., K. \tag{9.7.3}$$

Here the subscript j refers to the grid level, and I_j, P_j and R_j correspond to the time for injection, projection and relaxation. The thrust of the research of G & R was to show that when the MG algorithm was implemented on realistic designs, this idealized case did not hold true. Instead we obtained equations of the form:

$$I_j = P_j = R_j + f(j, n). \tag{9.7.4}$$

The reader can refer back to the previous sections to see examples of $f(j, n)$ such as $c2^{n-j}$ in the case of the VLSI implementation, or $c \log n$ in the case requiring Ω-network interconnections.

We note that for eq. (9.7.4) we have the ratio,

$$s = \frac{I_j + P_j}{R_j}.$$

It follows that in the idealized case, $s = 2$ and in all realistic cases we have $s \geq 2$, since

$$s = 2(1 + f(j,n)/R_j) \geq 2. \tag{9.7.5}$$

While we could give a more explicit determination of this ratio, for our purposes we merely note that $s = O(10)$, and usually $2 \leq s \leq 4$.

Now the other term is t which corresponds to the time required by one point in performing the necessary calculations for one relaxation sweep. Clearly this is on the order of $O(d)$ as this relates the number of nearest neighbor points that must be "averaged in" during the sweep. In particular we will say that we have $2d$ multiplications and $2d$ additions for each point during the sweep. This requires $4d$ elementary operations on the part of the processor. The time required for an elementary operation will obviously vary from one system (and one technology) to another, but in the case of VLSI we can assume that $t \approx 4d10^{-9}$ sec, or that one nanosecond is required for such operations as addition or multiplication, (see the section in chap. 2 entitled, "The Physical Basis of Computation.")

This takes care of $\beta(d)$. Now we will make the following assumptions. Assume n to be fixed. Recall that this is the width of a hyper-square, (it could easily be a rectangle). Thus, in two dimensions, we would have a $n \times n$ square, in three dimensions, a $n \times n \times n$ cube, and so on. We also assume that γ is fixed at $\gamma = 4$, which gives us 2γ or eight interprocessor connections. Finally, we assume that the time required for a direct solution on the coarsest grid is small enough to be ignored relative to the overall computational effort. This is tantamount to ignoring the constant terms in eq. (7.3.17) which gives the computing times for BASICMG.

We will also give ourselves a real-time constraint, T_{max}, which we arbitrarily set to 10^{-3} sec. This figure is chosen as being typical of the sort of computing speed we will likely need. The reader will recall that the one-dimensional Zakai solver described by LaVigna in his thesis also had this time bound.

The only parameter that we will allow to vary is the dimension d. We can then

ask: *at what value d does the computing time for r iterations of BASICMG exceed the real-time constraint?* First, we must choose an appropriate architecture. Since we will be looking at cases where $\gamma < d$, we must choose the most optimal architecture, (with respect to time), given this constraint on γ. A glance at (7.3.17) gives an optimal time when $c < a^{d-\gamma}$, which yields a computing time, for r iterations, of

$$rT(n) = r\beta a^p/(a^p - c)n^p. \tag{9.7.6}$$

When $p = d - \gamma$ is large, we can safely assume that $a^p/(a^p - c) \approx 1$. Setting T_{max} greater than eq. (9.7.6) gives

$$r\beta(d)n^{d-\gamma} < T_{max}, \tag{9.7.7}$$

or

$$n^d < \frac{T_{max}n^\gamma}{r\beta(d)},$$

which can be written,

$$d < \gamma + \frac{\log\left(\frac{T_{max}}{r\beta(d)}\right)}{\log n} \tag{9.7.8}$$

Now we can ask some quantitative questions. We set

d = dimension, which will vary from 1,2,...

$\gamma = 4$

$r = 10$ iterations

$T_{max} = 10^{-3}$ sec

$n = 100$

$t = 10^{-9}$ sec

$s = 2$ (see earlier discussion)

$j = m = 2$ relaxation sweeps

$\beta = (2+2+2)4\,dt$

We can use these values and find the maximal d for which equation (9.7.8) holds. Using a computer program, we find that *this maximal dimension is five.* Let

us change the above parameters slightly. If we let $n = 10$, keeping everything else constant, we would get *a maximum dimension of six.* It would seem that order of magnitude changes will not alter our results very much because we are incurring only logarithmic rates of change.

Let us now assume that $n = 10$ as before but that our estimate of t was too conservative. So let $t = 10^{-11}$ sec. Then we would have *a maximum dimension of eight.*

How might we correct for the $\gamma = 4$ restriction? Interconnection patterns do exist that could tie the processors together whose "nearest neighbors" requirements exceed the usual eight. For example, the Cube-Connected-Cycles interconnection, the Hypercube and the Butterfly organizations can all be thought of as networks that pass data from among the vertices of a k-dimensional cube. Each vertex can pass or receive data from every other vertex. The amount of time it takes to do so varies depending on the distance of one vertex to another, although the time is usually no more than k counts of the clock. These are also high-wire, area-intensive designs, and are more suitable for Supercomputers than for the special- purpose design we have in mind.

Snyder [27] argues that the CHiP project can allow for interconnection patterns of the kind that may be more useful to us. If the degree of the processor is the number of interconnections emanating from it, then Snyder argues that larger degrees are really not necessary as the same effect can be achieved either by multiplexing data paths or by logically coupling processor elements. For example, two degree-four PE's could be coupled together to form one degree-six PE. By the same token, two degree-six PEs could be coupled to form one degree-10 PE, and so on. However, the latter method will lead to a loss in processor utilization on the grid or wafer we would be using.

This then, is a major question that we can ask ourselves, once we obtain the software provided by the CHiP project: namely, how might the equivalent network of

processors with degrees > 8 be constructed, what is their time-area characteristics, and how might they compare with the more general interconnection networks based on the k-dimensional cube?

We can already identify some causes of time delay produced by using the CHiP design:

1). greater length of data paths.

2). a more complicated set of communication protocols–and hence a greater amount of time to calculate decisions governed by them.

3). buffering delays.

Without benefit of simulation studies, we can make some educated guesses as to the effect of using such a design for $\gamma > 4$ (or degree > 8). It would be reasonable to assume that an order of magnitude increase in time would be incurred at the expense of using such a design, and that this would depend linearly on d, or if γ is allowed to increase with d, on γ. Thus we modify (9.7.8) to read:

$$\beta(d)10dn^0 \leq T_{max} \tag{9.7.9}$$

where $d = \gamma$ and where $10d$ represents the cost in using the CHiP approach of logically coupling processors. Using this, the maximum dimension we can have, using the assumptions we first made in this section, is 9, for $n = 100$, and 13, for $n = 10$. Certainly, an improvement, but not a very dramatic one. And besides, our estimate that a cost of $10d$ is incurred for dimensions greater than 4 is probably less than what it is likely to be.

The reader is encouraged to vary the parameters on his own, but we have sufficiently demonstrated that a dimension of $d \geq 10$ is not likely, at least for most of the real-time constraints that are encountered in practice.

In the next section we ask an even more general question. Are there other computing structures, either real or hypothetical, that can outperform the computing times discussed above?

8. Alternative Architectures

Despite the elegance of the designs of Gannon and Van Rosendale, we cannot help but wonder whether other machines might do as well or even better. Especially now that we know there is a limit to what can be done with respect to the dimension d and keeping the real time computing bounds we have set for ourselves.

Hockney [15]-[16] has provided a two-parameter characterization of supercomputers, be they in the form of serial, pipelined or array-like architectures. The first parameter r_∞ is the traditional maximum performance in megaflops (i.e., millions of floating point operations per second) and the second parameter $n_{1/2}$, is a measure of the apparent parallelism of the system. The theory is designed to compare two different algorithms on the same or different systems. In [15], Hockney applies his method to a comparison of the direct methods for solving Poisson's equation known as the FACR(l) algorithm, which involves the optimal combination of Fourier analysis in the x−direction and block cyclic reduction of lines in the y-direction. Here, l is the number of stages of line cyclic reduction that is performed before Fourier analysis takes place. Determination of optimal l is then possible, given a parallel architecture.

Hockney is decidedly not suggesting that these two parameters characterize all the features that one would wish to know about a computing system. Other features might be cost and efficiency, to name only two. Rather, this two-parameter description is only a first order approximation of the system to be used in a relative comparison with other systems of entirely different structure, such as vector, or more generally pipelined systems, as well as serial and multiprocessor architectures. What we hope to do in this section is ask whether some existing machines can do any better with respect to computing time than the architectural prototypes described in this pages.

The two-parameter description of any computer is obtained by fitting the best straight line to the measured time, t, to perform a single vector operation on vector

of varying length, n. For example, the operations $A = B * C$, where A, B and C are vectors, or any other similar procedure will suffice. Hockney writes this best-fitting straight line as

$$t = r_\infty(n + n_{1/2}). \tag{9.8.1}$$

Again,

r_∞ = (maximum or asymptotic performance) or the maximum number of elementary arithmetic operations per second. This occurs for infinite vector length for the idealized computer.

$n_{1/2}$ = (half-performance length) the vector length required to achieve half the maximum performance.

Examples will make these terms clearer.

a). *serial computer:* The execution is simply proportional to

$$t = t_1 n, \tag{9.8.2}$$

where t_1 is the time for one elementary operation. Comparing with (9.8.1) we have

$$r_\infty = 1/t_1, \quad n_{1/2} = 0. \tag{9.8.3}$$

b). *pipelined computer:* the execution time is expressed as

$$t = (s + l + n - 1)\tau, \tag{9.8.4}$$

where

τ = clock period

s = set-up time in clock periods

l = number of segments in the arithmetic pipeline

Thus

$$r_\infty = 1/\tau, \quad n_{1/2} = s + l - 1. \tag{9.8.5}$$

c). *processor array:* Suppose we have a processor array of N processors which simultaneously perform the same arithmetic operation on n elements of each vector.

Let t_p be the time for one parallel arithmetic operation of all processors in the array. Then,

$$t = t_p[n/N], \tag{9.8.6}$$

and we see that

$$\begin{aligned} r_\infty &= N/t_p \\ n_{1/2} &= N/2. \end{aligned} \tag{9.8.7}$$

Note that $n_{1/2}$ is simply the "average parallelism. Hockney argues that for the CRAY-1, $n_{1/2} \approx 15$, and that for the CYBER 205, $n_{1/2} \approx 100$. The ICL DAP has $n_{1/2} \approx 2048$. We also note that the "infinite array" often used by complexity theorists has, appropriately enough, $n_{1/2} = \infty$.

Going back to the pipelined case, we see that we have large $n_{1/2}$ if l is large (large hardware parallelism) or if s is large (large overhead). Thus Hockney argues that in this case as well as in the others, $n_{1/2}$ is a measure of *apparent* parallelism.

The terms r_∞ and $n_{1/2}$ can be measured by a computer program, that is easily run on any system. A shorthand version of it appears below. The function *etime(tarray)* gives back a real number in the variable tarray, which is the user time since the last call to *etime*. The first call gives a value of zero.

```
      program time
      real tarray
      do 20 n=1,nmax
      call etime(tarray)
      do 10 i=1,n
10    a(i)=b(i)*c(i)
      call etime(tarray)
20    continue
```

Finding the best straight line through the data obtained by successive runs with different values of nmax gives a slope of r_∞ and an x-intercept of $-n_{1/2}$.

A variant of this program was run on the VAX here at the Univ. of Maryland. I found $r_\infty = 21240.1$ operations per second and $n_{1/2} = 9.44 \times 10^{-10}$, the latter value being expected on a serial machine. The value of r_∞ was corroborated by independent calculations by David Hsu of the computer staff here at Maryland, (and I would like to thank him for his help during this time.)

Hockney gives a general formula for the computing time of an algorithm:

$$T = r_\infty^{-1} \sum_{l=1}^{lmax} q_l(p_l + n_{1/2}) \tag{9.8.8}$$

where we define

$$q = \sum_{l=1}^{lmax} q_l, \tag{9.8.9}$$

is the total number of vector operations, the parallel operations' count, or the number of unit timesteps.

$$s = \sum_{l=1}^{lmax} q_l p_l, \tag{9.8.10}$$

the number of elementary operations, or the traditional serial (scalar) operations' count.

$$\bar{p} = s/q, \tag{9.8.11}$$

is the average vector length, or average parallelism of the algorithm. Thus

$$T = r_\infty^{-1} q(\bar{p} + n_{1/2}), \tag{9.8.12}$$

or

$$T = r_\infty^{-1}(s + n_{1/2}q). \tag{9.8.13}$$

We can also show that speedup is related to these quantities. We say that

$$\begin{aligned} \text{Speedup} &= \frac{\text{time of execution on uniprocessor}}{\text{time of execution on multiprocessor}} \\ &= \frac{\text{number of elementary operations}}{\text{number of parallel operations}} \\ &= \frac{s}{q} = \bar{p} \end{aligned} \tag{9.8.14}$$

The implicit assumption here is that the parallel and serial machines have the same speed for elementary arithmetic operations.

We now have a way of comparing the performance of algorithms on different systems. Define,

$$\frac{T^{(b,2)}}{T^{(b,1)}} = \frac{s^{(b)} + n_{1/2}^{(2)} q^{(b)}}{s^{(a)} + n_{1/2}^{(1)} q^{(a)}} \times \frac{r_\infty^{(1)}}{r_\infty^{(2)}} \tag{9.8.15}$$

where a, b correspond to different algorithms and 1, 2 correspond to different systems. Of course, we can compare the same algorithm ($a = b$) on different systems. In fact, if we divide (9.8.15) by $q^{(a)} = q^{(b)} = q$ we get

$$\frac{\tilde{p} + n_{1/2}^{(2)}}{\tilde{p} + n_{1/2}^{(1)}} \times \frac{r_\infty^{(1)}}{r_\infty^{(1)}} = \frac{T^{(2)}}{T^{(1)}}. \tag{9.8.16}$$

However, we might remark that we already have a satisfactory way of optimizing the design of the Multigrid algorithm on the C & S architecture. As stated above, the two-parameter method of Hockney will be used to compare performances on other systems.

But in the case of the Multigrid algorithm we can still exploit our knowledge of the speedup. This is simply

$$\left(\frac{a^p}{a^p - c}\right) \frac{n^\gamma}{\log_a n} \tag{9.8.17}$$

as seen by multiplying the efficiency E in Theorem 7.4.1 by $P(\gamma)$ (eq. (7.4.1)). This comes from the definition of speedup in eq. (7.4.1) as described in chap. 7.

However, let us use Hockney's notation and theory to if we arrive at the same result. We will use nominal values of the parameters throughout, in particular, we will let $c = 1$, which corresponds to the V-cycle. The results are easily generalized.

Now for BASICMG, we begin on the finest grid and do our relaxations. This involves $(2d + 1)$ operations and there are $2d$ nearest neighbors to a point. This is done to $n^d/2$ points. Thus we set

$$q_1 = (2d + 1), \quad n_K^d/2 = p_1. \tag{9.8.18}$$

Note that $s = (2d+1)n^d/2$ gives the number of serial operations that would have to be performed for the relaxation sweeps.

Injection is next. We can assume this involves averages of nearest neighbor points of $n^d/2$ of the total number of points. Thus

$$q_2 = (2d+1), \quad n_K^d/2 = p_2. \tag{9.8.19}$$

Interpolation can viewed as the "inverse" of injection. Values from a coarse grid is transmitted to the finer grid and then points not directly receiving a value take on the average of the nearest neighbor points that did receive values. This takes the same amount of time as injection. Again,

$$q_i = (2d+1), \quad n_k^d/2 = p_i. \tag{9.8.20}$$

It turns out that the factors in front of the terms n_k^d are all alike. Thus when we divide s by

$$q = \sum q_i,$$

where every q_i is a constant, we will have cancellation. We also note that the sum is from 1 to $2\log_a n_K$ since we have $\log_a n$ levels in the classical design, and so the speedup is

$$\frac{a^d}{a^d - c}\frac{n^d}{\log_a n}, \tag{9.8.21}$$

after having used eq. (7.4.1),

$$P(d) = \frac{a^d}{a^d - c}n^d,$$

which corresponds to the number of processors in a $\log_a n$ level system of $\gamma = d$.

We remark at this point that his speedup corresponds to our comparing the performance of the Multigrid algorithm on a multiprocessor system with an unlimited number of interconnections between processors and a processor for each point. Such a machine is not meant to be practical but is a measure of the *potential* speedup capable of the algorithm.

Thus the speedup predicted by Hockney agrees with that derived earlier by Chan and Schreiber. We note that in both cases, this speedup corresponds to systems, both serial and parallel, whose r_∞ remains constant.

Since we now know s, q, all we need to know regarding the performance of the Multigrid algorithm on a given system is its r_∞ and $n_{1/2}$. Some typical values of these two parameters are available from Hockney and Jesshope [16] and shown in Table (9.8.1).

Computer	$n_{1/2}$	r_∞ in megaflops/sec
64' CRAY-1	15	80
48' BSP	90	50
2-pipe 64' CDC CYBER 205	100	100
1-pipe 64' TIASC	30	12
64' CDC STAR 100	150	25
(64 x 64) ICL DAP	2048	16
HEP	820	1.7

Sample values of r_∞ and $n_{1/2}$

Table (9.8.1)

We can also compare the performance of another computing system relative to that of the VAX, whose r_∞ and $n_{1/2}$ were given previously, simply by using (9.8.16) and the r_∞ and $n_{1/2}$ for the new system.

Now if we ran r iterations of BASICMG on a given system, the computing time would be of the order of

$$T = r r_\infty^{-1} q(\bar{p} + n_{1/2}) \tag{9.8.22}$$

by eq. (9.8.12). We ask the question, given T_{max} = 1msec, what is the maximal dimension of the Zakai equation we can numerically solve by our BASICMG program and still be with this time constraint? Using the values from Table (9.8.1), we obtain the maximal dimensions found in Table (9.8.2). These values follow from

Computer	d_{max} for $n = 100$	d_{max} for $n = 10$
64' CRAY-1	1	3
48' BSP	1	3
2-pipe 64' CDC CYBER 205	1	3
1-pipe 64' TIASC	1	2
64' CDC STAR 100	0	1
(64 x 64) ICL DAP	0	0
HEP	0	0

Table 9.8.2 Maximum dimensions for $n = 100, 10$

the dependence of q and $\bar{p}$ on the dimension d.

The most startling observation we can make regarding Table (9.8.2) is that the performance of our parallel algorithm actually *fares worse* on the more parallel architectures, such as the DAP. How is this possible? We already know that a multiprocessor architecture is ideal for our purposes, as demonstrated by an analysis of the Gannon and Van Rosendale prototype. But this was a special purpose, custom-made machine. Those systems listed in Tables (9.8.1-9.8.2) are for general classes of algorithms. In fact, the more parallel machine will outperform their serial counterparts only for fairly large highly parallel programs.

To see this, let us, following Hockney, [15], define the performance (or speed) P, of an algorithm as the inverse of it time execution, i.e., T^{-1}, which denotes the number of executions of the algorithm that are possible per second. Then the relative performance of a parallel and a serial machine is given by

$$\frac{P_p}{P_s} = \frac{T_s}{T_p} = \frac{s_s \times t_s}{q_p \times t_p} \tag{9.8.23}$$

where the subscripts s and p refer to the serial uniprocessor and parallel multiprocessor respectively, and t_s and t_p are respectively the time for a serial and parallel

operation. The above equation can be written as

$$\frac{P_p}{P_s} = \frac{s_p}{q_r} \times \frac{s_s}{s_p} \times \frac{t_s}{t_p} \qquad (9.8.24)$$
$$= \text{Speedup} \times \text{algorithmic slowdown} \times \text{hardware slowdown}$$

The first factor is speedup, which is equal to $\bar{p}$ in eq. (9.8.14), but the second and third factors are slowdown factors and will be less than one. Hence, having $\bar{p} > 1$ will not be sufficient in order to have superior parallel performance.

Now algorithmic slowdown arises because the definition of speedup assumes that the parallel algorithm is executed on the serial uniprocessor with an elementary operations count of s_p. Now any algorithm chosen for a parallel computer will not be the best on a serial computer, and the number of elementary operation in the best serial algorithm s_s will almost certainly be less than t_p.

Hence the algorithmic slowdown factor is

$$\frac{s_s}{s_p} < 1 \quad (\text{typically } 1/5.) \qquad (9.8.25)$$

In effect, the elementary operations have increased in the parallel machine due to the distribution of work that is possible in that architecture.

The third slowdown factor is due to hardware differences, namely that the time needed to perform a serial operation on the uniprocessor t_s, will be much less than the the time needed to perform a parallel operation on a multiprocessor, t_p. Consider a machine of many thousands of processors. Each processor will be slower than a uniprocessor. Hockney argues that hardware slowdown is usually very small ($\approx 10^{-3}$ to 10^{-4}). An extreme example is provided by a comparison of the CRAY-1 and the ICL DAP. In the first case, the CRAY has an $n_{1/2} \approx 10$ and can produce an arithmetic result every 12.5 ns ($=t_s$). On the other had, the ICL DAP is a parallel array of 4096 processors and performs a parallel operation in about 250 μs ($= t_p$). For this case, the hardware slowdown is about 1/20000. Thus we would have to consider speedup $\bar{p}$ on the order of 100,000 or more before the DAP can outperform the CRAY. Unfortunately, it is precisely these speedups that put

the DAP computing time way beyond the real-time processing bound we have set for ourselves. This presumably may be a problem for all multiprocessor systems, including the Connection Machine, (see Hillis, [13]).

Therefore, once again we are led to believe that dimensions higher than six or seven will not allow the Multigrid algorithm to be performed in real-time. Even the Heterogeneous Processor or HEP Supercomputer, one of the latest MIMD systems to recently come out, appears to have difficulty. (The r_∞ and $n_{1/2}$ data for this machine is found in [14].) We might also remark that all of the above models were tested with $\gamma = d$, where d is the dimension. This corresponds to the performance of a multiprocessor system which allows only $2\gamma = 2d$ interprocessor connections, (using the notation of Chan and Schreiber). Of course, for large d, such a multiprocessor is either impossible or very difficult to build. What we are doing here is assuming the use of a hypothetical parallel machine similar in concept to the Paracomputer by Schwartz, []. The exploitation such abstract devices in the investigation of parallelism is now a standard procedure. What is really being measured in the speedup $\bar{p}$ is the *intrinsic parallelism of the algorithm.*

However, the above results follow only surveying today's commercially available systems. What about systems that are possible *in principle?* We can obtain some estimates of the performance of such machines by using theoretical results concerning the physical limits of computations. These results stem from investigations on the extreme limits attainable by a given technology with respect to computing times.

We assume a multi-processor network identical to the G & R network with three levels, the finest grid having n^7 processors, the middle grid with $n^7/2$ processors and the third with $n^7/4$ processors, which yields a total of $\frac{7}{4}n^7$ processors. Here, as we are dealing with real systems, we will assume that $\gamma = 4$ and that we have $2\gamma = 8$ interconnections between our processors.

According to Hockney, the parallelism is equal to half the total number of

processors, or in this case, $n_{1/2} = \frac{7}{8}n^7$. The speedup $\bar{p}$ and q would be the same as before. Only r_∞ would change, relative to the technology.

Now suppose we had a system with $\approx n^7$ processors and thus its $n_{1/2} = n^7/2$. Now according to Hockney, $r_\infty = 2n_{1/2}/t$, where t is the time to perform one parallel operation. If we set this time equal to that of performing one elementary operation, which is somewhat unrealistic, then we could get an upper bound on the dimensions we can expect to process in real-time. Using the silicon VLSI estimate of $t = 10^{-9}$ sec, we get *a maximum dimension of 9.*

Using superconducting technology, (which we are not likely to use in practice,) we get $t = 10^{-10}$ sec, and thus *a maximum dimension of 10.*

Other technologies may yield $t = 10^{-11}$ sec and $t = 10^{-12}$ sec, which respectively give *maximum dimensions of 10 and 11.*

9. Some Other Current Work in Multigrid Methods

The reader should be prepared for the fact that the standard model problem of the MG practitioners is Poisson's equation. Of course, this should still be an excellent benchmark for our own research. Thus the following survey of work, which makes extensive use of this model problem, is still of great relevance.

The problem of optimizing the parametrically dependent SOR method led Braess, '82, [], to consider using the parameter-free Gauss-Seidel method for Poisson's equation. His results show that, for rectangular domains, the GS method, used with red-black ordering, works rather well. This implies that we should first use the SOR method with $\omega = 1$, and then decide if any further optimization is really necessary.

We might remark at this point that any Multigrid analysis should be done in comparison with other linear problem solvers. A survey of available software packages up to 1982 can be found in Duff, [5], who offers this overview as a guide to the strengths and weaknesses of currently used codes. He examines

direct methods: such as those contained in LINPACK and SPARSPACK;

iterative methods: such as those in ITPACK, which include all of the standard techniques, some of which are amplified by Chebyshev acceleration or by the conjugate gradient method;

semi-direct methods: which are similar to to relaxation schemes but which converge in a predetermined number of iterations.

fast methods: which are specially designed for the very regular structure of the Laplace, Poisson and Helmholtz equations.

A software package for Multigrid problems on rectangular domains is described by Foerster and Witsch, [7], and is called MG00. This method easily meets the benchmark provided by fast direct methods for solving standard problems. MG)) is also *not* a black-box program which returns a meaningful solution for each problem given to it, but it does issue warning messages to indicate to the user when performance may be less than expected. MG00 is written in portable Fortran and is very user-oriented. It also includes a greater variety of relaxation methods than those discussed in this dissertation.

10. Conclusion

We examined in this chapter various architectures that are suitable for Multigrid algorithms. They each had different trade-offs: some were incapable of concurrent iteration, others were too area-intensive for our purposes. The latter case involved the use of the Ω-network, a generalization of the shuffle-exchange networks. We found that this design was actually superior with respect to computation speed, but the cost in area may make it unsuitable as far as VLSI chip-size implementation is concerned. The need to accept a design with a suboptimal computing speed was viewed as inevitable if we wanted to keep the area of the system from growing beyond our constraints. However, a CHiP layout was possible for one design, and this technology for multiprocessor network construction was briefly discussed.

Numerical results of Gannon's and van Rosendale's experiments, which formed a major portion of this chapter, were discussed, and we noted that the model problems they used were very relevant to our own research. We found that the parameters they identified as being crucial are likely to be negligible when the systems are put in VLSI, thus only for a very large number of grid levels and grid sizes would we expect to see a substantial difference in the designs discussed here.

The problems associated with Concurrent Iteration (CI), a very promising variant of the MG algorithm, were discussed. Specifically, the performance of CI is very good relative to other MG variations, but it is also very sensitive to changes in the domain size. This may pose a problem in our work as the ability to design the size of a domain would be an attractive feature in any Zakai Solver package.

We also discussed the area layout growth rate as a function of dimension, and found that it grew exponentially, making VLSI implementation in small chips impossible except for $d \leq 3$.

Then we examined our own design for a direct solver, which all MG designs incorporate as the responsibility of the coarsest grid. In keeping with our VLSI methodology, we designed a solver based on LU decomposition that we felt as con-

sistent with our overall design constraints.

Stability issues were also addressed which included wordlength considerations and round-off error propagation. We found that bit size requirements to be well within our expected problem size and round-off error to be controllable due to the structure of the MG algorithm.

The question of what dimensional range of our Zakai equation can be accommodated by our Multigrid design was explored by a careful integration of the work of Gannon and Van Rosendale and Chan and Schreiber. We found that, depending on reasonable estimates of computing speeds, Zakai equations in dimensions no higher than about six or seven can be solved in real-time, where the latter term is defined as being on the order of 1 msec.

The question of whether other architectures could fare any better was also addressed using the theories of Hockney. He provided a two-parameter characterization of all computing systems which we showed yields performance estimates of various high-speed architectures with respect to our algorithm. We found that general purpose systems fared worse than our custom-made model. Also, multiprocessor systems as a rule only begin to outperform their high-speed serial counterparts at dimensions beyond which real-time processing is possible, i.e., they are faster but still take too long with respect to reasonable real-time computing bounds.

Numerical work and software packages of other researchers was also surveyed.

References for Chapter 9

[1] Braess, D., "The Convergence Rate of a Multigrid Method with Gauss-Seidel Relaxation for the Poisson Equation," in Hackbusch and Trottenberg. *

[2] Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation,* vol. 31, No. 138, 1977.

[3] ——, "Guide to Multigrid Development," in Hackbusch and Trottenberg.*

[4] Chan, T. and Schreiber, R., "Parallel Networks for Multi-grid Algorithms: Architecture and Complexity," *SIAM J. Sci. Stat. Comput.*, vol. 6, No. 3, July, 1985.

[5] Duff, I., "Sparse Matrix Software for Elliptic PDE's," in Hackbusch and Trottenberg.*

[6] Fadeeva, P. K., *Computational Methods of Linear Algebra,* Freeman Pubs., p. 288, 1983.

[7] Foerster, H. and Witsch, K., "Multigrid Software for the Solution of Elliptic Problems on Rectangular Domains: *MG*00," in Hackbusch and Trottenberg.*

[8] Gannon, D. and van Rosendale, J., "On the Impact of Communication Complexity on the Design of Parallel Numerical Algorithms," *IEEE Trans. on Computers,* vol. C-33, No. 12, Dec. 1984.

[9] ——, "Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis," ICASE Report No. 82-36, Nov. 1982.

[10] ——, "On the Structure of Parallelism in a Highly Concurrent PDE Solver," *Jour. of Parallel and Distributed Computing,* vol. 3, 1986.

[11] Gannon, D. "On Mapping Non-uniform PDE Structures and Algorithms onto Uniform Array Architectures," from IEEE Conf. on Parallel Processing, 1981.

[12] Grosch, C., "Performance Analysis of Poisson Solvers on Array Computers," SuperComputers: 2, Infotech International, Maidenhead, 1979.

[13] Hillis, D., *The Connection Machine,* MIT Pr., 1985.

* See bibliographic entry.

[14] Hockney, R. W. and Snelling, D. F., "Characterizing MIMD Computers: e.g. the Denelcor HEP," Parallel Computing, '83, North Holland.

[15] Hockney, R. W., "Performance of Parallel Computers," in Paddon, D., ed., *Supercomputers and Parallel Computation,* Clarendon Pr., Oxford, 1984.

[16] Hockney, R. W. and Jessope, C. R., *Parallel Computers: Architecture, Programming and Algorithms,* Adam Hilger, Bristol, 1981.

[17] Jordan, H., "A Special Purpose Architecture for Finite Element Analysis," Proc. 1978 International Conf. on Parallel Processing, p. 263-66.

[18] Kung, S. Y., Arun, K. S., Gal-Ezer, R. and Bhaskar, R., "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computers,* vol. C-31, No. 11, Nov. 1982.

[19] Kung, H. T. and Lam, M., "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *Journal of Parallel and Distributed Computing,* vol. 1, 1984.

[20] Kung, H. T., "Two-Level Pipelined Systolic Arrays for Matrix Multiplication, Polynomial Evaluation and Discrete Fourier Transform," Workshop on Dynamic Behavior of Automata, Luminy, France, 1983.

[21] ——,"Systolic Arrays," Dept. of Computer Sci., Carnegie-Mellon Univ. Pittsburgh, Penn. 1984.

[22] ——, "Why Systolic Architectures," *Computer,* Jan. 1982.

[23] ——, "Systolic Algorithms," in *Large Scale Scientific Computation,* Academic Pr. 1984.

[24] Miranker, W. and Winkler, A., "Spacetime Representation of Computational Structure," *Computing,* vol. 32, 1984.

[25] Paddon, D., ed., *Supercomputers and Parallel Computation,* Clarendon Pr. Oxford, 1984.

[26] Schultz, A., *Elliptic Problem Solvers, I, II,* Academic Pr., vol I., 1981, vol. II, 1984.

[27] Snyder, L, "Introduction to the Configurable Highly Parallel Computer, *IEEE Computer*, Jan. 1982.

[28] Storassli, O., Peebles, F., Crockette, T., Knott, J., and Addams, L., *The Finite Element Machine: An Experiment in Parallel Processing,* NASA Technical Memorandum, No., 84514,in Parallel Processing, NASA Technical Memorandum, No., 84514, July 1982.

10. Conclusion

1. Review of Goals and Results

The immediate purpose of this research was to determine the high-speed signal processing properties of the Multigrid algorithm with respect to the Zakai equation. The more general purpose was to explore some parallel processing concepts that may eventually have an impact on the controls and systems area. In effect, the real-time solution of the Zakai equation can be viewed as a case study of a much larger set of issues involving the role of computation in decision and control systems, and we tried to delineate some of these issues throughout this dissertation. Also, the inter-disciplinary nature of this research area required a greater effort in supplying introductory material, as it is assumed that prospective readers may be coming with different backgrounds and perspectives.

We made an attempt to bridge three distinct fields of electrical engineering:

a). nonlinear filtering theory

b). numerical analysis

c). VLSI and parallel processing

Examples of where these bridges were laid include:

a review of the Zakai equation and means of obtaining estimates on how to bound its values on a compact set, clearly a prerequisite for any numerical work on this equation.

a study of the Multigrid algorithm and its appropriateness for the Zakai equation.

a demonstration of the suitability of parallel processing for the MG algorithm, especially when implemented in VLSI.

To give more details, we studied selected topics in VLSI theory in chapter 2 which:

delineated the nature of parallel processing, which identified those issues that would preoccupy us throughout this study.

introduced systolic arrays for matrix processing; these served as our prototype architecture.

discussed the physical basis of computation time–results which were useful in later chapters on time estimates.

described a methodology for the optimal design of systolic arrays.

explored the variations of shuffle-exchange a permutation networks–the results here were relevant in the analysis of communication cost models.

pointed out the difficulties in synchronizing large systolic arrays operating in nanosec clock times. This included a graph theoretic arguments that showed that clock skew must grow as $\Omega(n)$ for a $n \times n$ array.

offered a compromise between the systolic array and the general data-flow machine in the form of the Wavefront Array Processor. (WAP) This architecture can avoid the synchronization difficulties mentioned above. The WAP figured prominently in our later design work.

reported on current work in progress at the Univ. of Maryland on the scalar nonlinear filter and its systolic implementation. This serves as the path breaking work for our own research.

The topics discussed above were incorporated into our research effort.

The study of the Zakai equation and methods of finding a compact set for which the solution was less than an arbitrary ϵ on its complement were discussed in chapters 3 and 4. This included some technical extensions to the work of Baras *et al.* Even tighter bounds than those discussed here are probably possible by numerical simulation.

Theories of weak convergence followed in chapter 5. This included a finite difference scheme that converges weakly to the true solution as Δt and $\Delta x_1, \Delta x_2, ..., \Delta x_n$ go to zero, and what is more, space and time finite differences can go to zero *in-*

dependently of each other. We found this to be essential to our purposes, as the time step will be set by a sampling rate. This is in marked contrast to explicit schemes which place a tighter restriction on the discretization, especially in higher dimensions. Our scheme is implicit and yields a linear system whose matrix has an elegant structure: it is diagonally dominant, real and hence positive definite. We derived other properties of this matrix as well. We also remarked that this structure will lend itself very well to the relaxation techniques we will be considering.

Chapters on the Multigrid algorithm properly began with chapter 6. We studied its convergence properties, noting that it was an iterative procedure, and examined its usefulness in connection with the Zakai equation. This included the effect of using the value of the solution at the previous time step as an initial approximation, and an estimate for the number of iterates that will be needed for convergence to occur. This is in contrast to generating a first approximation, which can be done automatically by the Multigrid algorithm. We also discussed the use of the Fokker-Planck equation as a model problem–arguing that a two-grid analysis should be sufficient for empirically testing various designs. Using such arguments, we found that all the parameters of the program to be *precomputable*, which is essential as we do not wish to use a convergence test that would take up our entire allotted computing time of about 1 msec. However, this could lead to the problem of using more iterations than would be necessary to reach convergence, such as in those cases where the conditional density stabilizes, produces fewer and fewer changes with each new time-step. Studies of the heat equation using the Multigrid method have produced a technique that can compensate for this problem while not relying too heavily on global computations, and their expensive communication requirements.

Chapter 7 was a more detailed study of the complexity of the Multigrid algorithm, and the effect of various parameters on the duration of the computing time, accuracy and efficiency. We examined these trade-offs in the light of our problem.

The recursive structure of the Multigrid algorithm was elucidated, and this was contrasted with concurrent iteration, which allows relaxations to be performed on the grids simultaneously. The latter algorithm has a number of technical difficulties, and it also taxes the overall complexity of the program and hence the complexity of the individual processors. However, it appears to be somewhat faster that conventional Multigrid, according to numerical simulations, and certainly more efficient with respect to processor utilization, and so trade-off comparisons were made.

A look at relaxation sweeps was in order in chapter 8. After surveying what is available, we found that the successive over relaxation method to be best suited for our purposes, and we did this by analyzing the structure of the linear system we are solving, which we found to be a positive definite L-matrix, as explained in the text. Ways of measuring the smoothing factor of the relaxation method, as this determines the amount of error incurred by transferring to coarser grids, was discussed as well. Asynchronous implementation of relaxation methods were discussed, and we argued that Red-Black or, in general, multicolor, ordering was needed and that the WAP, introduced in chapter 2, would be ideal for our purposes.

The chapter on empirical results (9) contained a critique of some numerical simulations that were done by Gannon and Van Rosendale. We surveyed the performance of a variety of architectures and their communication cost structure. We found that optimal computing time was achieved by the most area intensive designs, which is to be expected. However, our real-time computing bound does not give us much maneuvering room for area-time trade-offs, but we did conclude that permutation networks, such as those introduced in chapter 2, would be needed for high speed performance. However, a VLSI implementation of an elegant Multigrid architecture that does not possess such sophisticated interconnection networks is possible by using a CHiP design. This is the Configurable Highly Parallel architecture, and a package program for designing in it is available.

We also examined the numerical stability of the Multigrid algorithm, with

respect to word length and round-off effects. We also examined the complexity requirements of the individual processors, using currently existing designs as a benchmark.

Also, we designed a direct solver, which is incorporated into the Multigrid framework, by requiring that the overall architecture dictate the design, as opposed to letting the algorithm determine the best architecture for this subsystem. We did this in the interests of preserving the regularity of the global multigrid design, as this allows for ease of fabrication.

Using the complexity bounds of chapter 7, and combining them with the work of Gannon and Van Rosendale, we were able to show that optimal performance within the real-time processing bound of 1 msec was not possible for problems with dimensions higher than six or seven, depending on the arithmetic computing speed obtainable. This followed from observations on the limits imposed by interprocessor connections in a mesh configuration, as we assumed to have no higher that eight interconnections to a processors nearest neighbors. According to claims the CHiP project, a higher number of interconnections is possible but this is obtained by much longer data paths. If we make the natural assumption that signal delays are increased by an order of magnitude, this the maximal problem dimension is increased to about 10.

We also discussed in this chapter the work of Hockney and his two-parameter characterization of computing machines. We found that other more general purpose systems could fare no better, and that general multiprocessor machines have superior performance over their serial counterparts only on problems whose size is so large as to already be beyond the computing time bound we have set for ourselves. Examinations of scaled down versions of these machines was also discussed. The special purpose system described in these pages appears to be the optimal machine, despite its limitations in reaching the higher dimensions.

2. Final Remarks

The results described here is merely a first step in a larger research effort. For example, more sophisticated filtering problems can be investigated that could lead to more complicated domains and boundary conditions. This in turn will lead to new matrix representations of the finite difference scheme which will probably require, among other things, a reappraisal of the relaxation methods we have suggested. Our survey of what is available in this area should be a good start in that direction.

Also, the various package programs that were mentioned throughout this paper should be acquired. Using the model problem in the form of the Fokker-Planck equation, we could obtain an expertise in finding the norms of the Multigrid operator, smoothing factors of relaxation techniques, and performance comparisons of various Multigrid parametric designs.

Investigations into different communication designs for intergrid data transfers in higher (¿2) dimensions must be conducted. Unfortunately, judging from the literature, little appears to be known, except that hypercube architectures are likely to dominate. This is almost certainly not suitable for our purposes, as they will probably only be efficient for problem sizes that require computing times well beyond that which we have allotted for ourselves in a real-time environment. However, the CHiP project does promise planar arrays with individual processors connected to more than eight neighbors, so experimenting with the interactive CHiP graphics design package, (available this July, '86), would certainly be in order.

Another thought provoking area of investigation would be in determining the use of comparing the information that is collected on each grid–a kind of multi-resolution approach to the problem. This notion appears to be important in the field of image processing. In fact a March 1986 article in the *IEEE Transactions on Pattern Analysis and Machine Intelligence* discussed the use of multigrid methods in analyzing images. By comparing results of each grid to that of its coarser and finer counterparts, decisions could be made on the shape of objects in the visual field.

Can this multi-resolution approach be generalized to a broader class of problems?

A related notion is the use of coarse grid tracking and local refinement. Here we are dealing with problems whose conditional densities are mostly confined to certain portions of the grid. The coarse grid can cheaply locate them and, by using local refinement on the finer grids, zero into an efficient computation of the solution. Determining the class of problems for which technique would be best suited would be helpful.

Clearly, there are many directions in which we can go, for this is a very active field of research, and as all research is open-ended by nature, more questions have been raised than answered. But for now, our present research has ended.

Bibliography

Adams, L. and Ortega, J., "A Multi-Color SOR Method of Parallel Computation," Dept. of Applied Math, Univ. of Virginia, 1982.

Archetti, F. and Cugiani, M., eds., *Numerical Techniques for Stochastic Systems,* North Holland, 1980.

Arnould, E., Kung, H. T., Menzilcioglu, O., and Sarocky, K., "A Systolic Array Computer," from Proc. of IEEE Conf. on Acoustics, Speech, and Signal Processing, Mar. 1985.

Bank, R. and Dupont, T., "An Optimal Order Process for Solving Finite Element Equations," *Mathematics of Computation,* vol. 36, No. 153, Jan. 1981.

Baras, J., Blankenship, G. and Hopkins, A., "Existence, Uniqueness, and Asymptotic Behavior of Solutions to a Class of Zakai Equations with Unbounded Coefficients", *IEEE Trans. Automatic Control,* vol. AC-28, No. 2, Feb. 1983.

Baudet, G., "Asynchronous Iterative Methods for Multiprocessors," *Journal of the Association for Computing Machinery,* vol. 25, No. 2, Apr. 1978.

Besala, P. "Fundamental Solution and Cauchy Problem for a Parabolic System with Unbounded Coefficients," *J. Diff. Eqns,* vol. 33, pp. 26-38, 1979.

Bilardi, G., Pracchi, M. and Preparata, F., "A Critique of Network Speed in VLSI Models of Computation," *IEEE Jour. of Solid State Circuits,* vol. SC-17, No. 4, Aug. 1982.

Braess, D., "The Convergence Rate of a Multigrid Method with Gauss-Seidel Relaxation for the Poisson Equation," in Hackbusch and Trottenberg.

Brand, K., "Multigrid Bibliography," in Hackbusch and Trottenberg.

Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation,* vol. 31, No. 138, 1977.

——, "Guide to Multigrid Development," in Hackbusch and Trottenberg.

Bucy, R. and Senne, K., "New Frontiers in Nonlinear Filtering," Lincoln Lab. Technical Note 1978-16, 1978.

Chan, T. and Schreiber, R., "Parallel Networks for Multi-grid Algorithms: Architecture and Complexity," *SIAM J. Sci. Stat. Comput.*, vol. 6, No. 3, July, 1985.

Davis, M. H. A. and Marcus, Steven, "An Introduction to Nonlinear Filtering," in *Stochastic Systems: The Mathematics of Filtering and Identification*, (NATO Advanced Study Institute Series). Dordrecht, The Netherlands: Reidel, 1981, pp. 53-75.

Davis, M. H. A., "A Pathwise Solution of the Equations of Zakai Filtering," Unpublished manuscript.

Davis, R. and Thomas, D., "Systolic Array Chip Matches the Pace of High-Speed Processing," *Electronic Design* vol. 32, no. 22, Oct. 1984.

Doob, J., *Stochastic Processes,* Wiley, 1953.

Duff, I., "Sparse Matrix Software for Elliptic PDE's", in Hackbusch and Trottenberg.

Fadeeva, P. K., *Computational Methods of Linear Algebra*, Freeman Pubs., p. 288, 1983.

Fisher, A. and Kung, H. T., "Synchronizing Large VLSI Processing Arrays," *IEEE Trans. on Computers*, vol. C-34, No. 8, Aug. 1985.

Foerster, H. and Witsch, K., "Multigrid Software for the Solution of Elliptic Problems on Rectangular Domains: *MG*00," in Hackbusch and Trottenberg.

Gannon, D. and van Rosendale, J., "On the Impact of Communication Complexity on the Design of Parallel Numerical Algorithms," *IEEE Trans. on Computers*, vol. C-33, No. 12, Dec. 1984.

——, "Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis," ICASE Report No. 82-36, Nov. 1982.

——, "On the Structure of Parallelism in a Highly Concurrent PDE Solver," *Jour. of Parallel and Distributed Computing*, vol. 3, 1986.

Gannon, D. "On Mapping Non-uniform PDE Structures and Algorithms onto Uniform Array Architectures," from IEEE Conf. on Parallel Processing, 1981.

Gelenbe, E. Lichnewsky, A., and Staphylopatis, A., "Experience with the Parallel Solution of Partial Differential Equations on a Distributed Computing System," *IEEE Trans. on Computers*, vol. C-31, no. 12, 1982.

Gray, J., ed., *VLSI '81*, Academic Press, 1981.

Grosch, C., "Performance Analysis of Poisson Solvers on Array Computers," Super-Computers: 2, Infotech International, Maidenhead, 1979.

Hackbusch, W. and Trottenberg, U., eds., *Multigrid Methods*, Lecture Notes in Mathematics, Springer-Verlag, No. 960, 1982.

Hackbusch, W., "Multigrid Convergence Theory," in Hackbusch and Trottenberg.

Hazelwinkle, M. and Willems, J. C., eds. "Stochastic Systems: The Mathematics of Filtering and Identification," (NATO Advanced Study Series), Dordrecht, The Netherlands: Reidel, 1981.

Hedstrom, G. and Rodrique, G., "Adaptive-grid Methods for Time-dependent Partial Differential Equations," in Hackbusch and Trottenberg.

Heller, D., "A Survey of Parallel Algorithms in Numerical Linear Algebra," *SIAM Review*, vol. 20, No. 4, Oct. 1978.

Hillis, D., *The Connection Machine*, MIT Pr., 1985.

Hockney, R. W. and Snelling, D. F., "Characterizing MIMD Computers: e.g. the Denelcor HEP," Parallel Computing, '83, North Holland.

Hockney, R. W., "Performance of Parallel Computers," in Paddon.

Hockney, R. W. and Jessope, C. R., *Parallel Computers: Architecture, Programming and algorithms,* Adam Hilger, Bristol, 1981.

Hopkins, W. E., *Nonlinear Filters of Nondegenerate Diffusions with Unbounded Coefficients,* Ph.D thesis, Univ. of Maryland, 1982.

Jordan, H., "A Special Purpose Architecture for Finite Element Analysis," Proc. 1978 International Conf. on Parallel Processing, pp. 263-266.

Kung, S. Y., Arun, K. S., Gal-Ezer, R. and Bhaskar, R., "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computers,* vol. C-31, No. 11, Nov. 1982.

Kung, H. T. and Lam, M., "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *Journal of Parallel and Distributed Computing,* vol. 1, 1984.

Kung, H. T., "Two-Level Pipelined Systolic Arrays for Matrix Multiplication, Polynomial Evaluation and Discrete Fourier Transform," Workshop on Dynamic Behavior of Automata, Luminy, France, 1983.

——,"Systolic Arrays," Dept. of Computer Sci., Carnegie-Mellon Univ. Pittsburgh, Penn. 1984.

——, "Why Systolic Architectures," *Computer,* Jan. 1982.

——, "Systolic Algorithms," in *Large Scale Scientific Computation,* Academic Pr. 1984.

——, "Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI," from the Proc. of the Soc. of Photo-Optical Instr. Engr., 1980.

Kushner, H. and Yu, C., "Probability Methods for the Convergence of Finite Difference Approximations to Partial Differential Equations," *Jour. of Math. Analysis and Appl.,* vol. 43, 1973.

LaVigna, A., "Real Time Sequential Detection for Diffusion Signals," Dept. of Electrical Eng., Univ. of Maryland, 1986.

Lawrie, D., "Access and Alignment of Data in an Array Processor," *IEEE Trans. on Computers*, vol. C-24, No. 12, Dec. 1975.

Legland, F., *Estimation de Parametres dans les Processes Stochastiques en Observations Incomplete*, L'Université de Paris, IX-Dauphiè, 1981.

Li, Guo-Jie, and Wah, B.,"The Design of Optimal Systolic Arrays," *IEEE Trans. on Computers*, vol. C-34, No. 1, Jan. 1985.

Mead, C. and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, 1980.

Miranker, W. and Winkler, A., "Spacetime Representation of Computational Structures, " *Computing* , vol. 32, 1984.

Novak, Z., "Use of Multigrid Method for Laplacian Problems in Three Dimensions," in Hackbusch and Trottenberg.

Ortega, J. and Voigt, R. "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Review*, vol. 27, No. 2, June, 1985.

Paddon, D., ed., *Supercomputers and Parallel Computation*, Clarendon Pr. Oxford, 1984.

Pardoux, E. "Stochastic Partial Differential Equations and Filtering of Diffusions Processes," *Stochastics*, vol. 3, pp. 127-167, 1979.

Parker, D., "Notes on Shuffle/Exchange-Type Switching Networks," *IEEE Trans. on Computers*, vol. C-29, No. 3, March, 1980.

Piccioni, M., "Convergence of Implicit Discretization Schemes for Linear Differential Equations with Applications to Filtering," Dept. of Electrical Engineering, Univ. of Maryland, 1985.

Sameh, A., "On Jacobi and Jacobi-Like Algorithms for a Parallel Computer," *Mathematics of Computation*, vol. 25, No. 115, July, 1971.

Schultz, A., *Elliptic Problem Solvers, I, II*, Academic Pr., vol. I, 1981, vol. II, 1984.

Seitz, Charles, "Concurrent VLSI Architectures," *IEEE Trans. on Computers*, vol. C-33, No. 12, Dec. 1984.

Snyder, L, "Introduction to the Configurable Highly Parallel Computer, *IEEE Computer*, Jan. 1982.

Stone, H. "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers*, vol. C-20, No. 2, Feb. 1971.

Storassli, O., Peebles, F., Crockette, T., Knott, J., and Addams, L., *The Finite Element Machine: An Experiment in Parallel Processing*, NASA Technical Memorandum, No. 84514, July 1982.

Stüben, K. and Trottenberg, U., "Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications," in Hackbusch and Trottenberg.

Travassos, R. H., *Application of Systolic Array Technology to Recursive Filtering*, Prentice Hall, 1983.

Ullman, J., "Computational Aspects of VLSI," Computer Science, Pr. 1984.

Wesseling, P., "A Robust and Efficient Multigrid Method," in Hackbusch and Trottenberg.

Wong, E., *Stochastic Processes in Engineering Systems*, Springer-Verlag, 1985.

Young, D., *Iterative Solution of Large Linear Systems*, Academic Pr. 1971.