01 Jan 1988

Executable Assertion Development for the Distributed Parallel Environment

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

L. M. Ni

Recommended Citation

B. M. McMillin and L. M. Ni, "Executable Assertion Development for the Distributed Parallel Environment," *Proceedings of the 12th International Computer Software and Applications Conference, 1988*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1988.
The definitive version is available at https://doi.org/10.1109/CMPSAC.1988.17187

EXECUTABLE ASSERTION DEVELOPMENT FOR THE DISTRIBUTED PARALLEL ENVIRONMENT†

Bruce M. McMillin[1]

Lionel M. Ni[2]

Computer Science Department[1]
University of Missouri-Rolla
Rolla, MO 65401-0249
C0556@UMRVMB.BITNET

Department of Computer Science[1,2]
Michigan State University
East Lansing, MI 48824-1027
ni@cps.msu.edu

Division of Mathematics and Computer Science[2]
Argonne National Laboratory
Argonne, IL 60439

ABSTRACT: The use of executable assertions is a powerful tool with which to perform program verification, to provide software fault-tolerance, and to provide hardware fault-tolerance via the application-oriented paradigm. In this paper we show that assertions commonly used in the sequential programming environment are inadequate for the distributed parallel environment. In particular it is shown that even design based assertions are myopic and provide inadequate error coverage. In their place, a triad of basis metrics are proposed for certain classes of problems which, when applied beginning with the specification phase of the life cycle, produce assertions that are better suited to the parallel environment. This method is applied to a well-known parallel computing problem in order to demonstrate the effectiveness of the method.

1. INTRODUCTION

The *Assertion* is the basic unit of program verification [Somm82], software fault-tolerance [Rand75], and the application oriented fault-tolerance paradigm [McMi88]. Development of these assertions is straightforward for the sequential program execution environment (See [YaCh75, Andr79, Somm82, HuAb87] for examples). Assertions are primarily generated from the coding and design process.

An assertion is a statement of the form:

if not ASSERTION **then** ERROR;

where ASSERTION is a boolean invariant on the program state. Assertions integrated into the program code are called *Executable Assertions*. If an assertion fails due to some abnormality, be it hardware or software, the ERROR condition is raised and appropriate action taken.

In the sequential programming environment, each result of a statement is deterministically a function of the current program state and the statement executed. Procedures and function calls may be handled in one of two ways. Either the function may be considered as a "meta-statement" which is a function of the calling program's state or the statements internal to the called function may be expanded inline to the calling program. In any case, assertions are made using the state of a single program and its statements. Careful choice of these assertions, particularly those involving loop invariance, lead to a high degree of confidence in both the program's termination and correctness.

† This work was supported in part by the DARPA ACMP project and in part by the GTE Graduate Research Fellowship program.

The target execution environment considered in this paper is the distributed memory multiprocessor (DMMP). The DMMP is characterized by autonomous MIMD (Multiple Instruction Multiple Data Stream) processors each with their own local memory. Interprocessor communication is accomplished by message passing over an interconnect. This environment limits the amount of information available to an executable assertion. Since a checking processor cannot examine another's internal memory, the message sent is the only form of communicated information available.

To develop and implement executable assertions on a DMMP requires additional consideration over the sequential environment. Since processors may only communicate via messages, detailed assertions may not be implementable. In general, the detailed assertions do not occur at *testable stages*, i.e. message interchange points. Thus, the entire program state is no longer deterministically obtainable by any single processor. This requires executable assertions be developed which function in the presence of partial information.

These DMMP issues foster consideration of granularity of test design and specification as a primary issue. Creation of assertions at the message interchange granularity is essential if true distributed diagnosis of hardware faults is to be achieved. Fortunately, as a problem is decomposed into its parallel components and partitioned/mapped onto the parallel processor, the message exchanges fall at the problem specific parallel component boundaries. Assertions then are a function of the local program state and any received messages from other parallel components.

The goal of this paper is to show how executable assertions that function at the appropriate granularity and function under incomplete information may be generated for the distributed parallel environment. These assertions must lead to the same high degree of confidence in termination and correctness as is attained for sequential programs.

This paper is organized into three remaining parts. Section 2 shows the shortcomings of sequential program assertions in the parallel environment. Section 3 presents a set of basis metrics which may be applied starting at the specification phase of the life cycle to generate assertions for the parallel environment. In Section 4, these metrics are applied to a parallel solution of a matrix problem. Error coverage is modeled probabilistically such that dominance of assertions may be quantified.

2. ASSERTIONS IN THE SEQUENTIAL ENVIRONMENT

As the model problem, we consider matrix iterative techniques which have been employed since the 1940's to solve large systems of equations numerically [Varg62, Ames77]. These systems of equations arise out of the numerical solution of physical problems utilizing finite difference and/or finite element methods. Techniques of particular interest are the *pointwise relaxation* tech-

niques, which, while not as elegant as some other methods, do contain the massive inherent parallelism necessary to take full advantage of a DMMP.

The problem is to solve the linear system $\mathbf{Au}=\mathbf{v}$, where, $\mathbf{A}=(a_{i,j})$ is a nonsingular $Q\times Q$ complex matrix, $\mathbf{v}=(v_i)$ is a complex vector, and $\mathbf{u}=(u_i)$ is the solution vector for $i,j\in\{1,2,...,Q\}$ and Q a perfect square. The method of Successive Over Relaxation (SSOR) is an iterative technique with which to obtain an approximate solution, $\mathbf{u}^{(K)}=(u_i^{(K)})$, where K is the final iteration, to this system. Keeping in mind that this algorithm will be parallelized, it is written in parallel even though in this section, execution will be sequential. A *consistent multicolor ordering* of the Q values of $\mathbf{u}^{(K)}$ (also called points) is made. Selection of points for computation is controlled by the iteration variable k as in the following predicate.

Definition 1:
$$\Lambda_{i,k}=\begin{cases}1 & \text{if } u_i^{(k)} \text{ is computed during iteration } k\\ 0 & \text{otherwise.}\end{cases}$$

The most common of these consistent multicolor orderings is the red/black or checkerboard multicolor ordering [Smit65] (implying, among other things, that the number of nonzero $a_{i,j}$'s per row i is no more than five). For this specific case $\Lambda_{i,k}$ becomes:

Definition 2: For $i\in\{1,...,N\}, k\in\{0,1,2,...\}$

$$\Lambda_{i,k}=\begin{cases}1 & \text{if } (i \textbf{ div } \sqrt{Q}+i \textbf{ mod } \sqrt{Q}) \text{ is even and } k \text{ is even}\\ 1 & \text{if } (i \textbf{ div } \sqrt{Q}+i \textbf{ mod } \sqrt{Q}) \text{ is odd and } k \text{ is odd}\\ 0 & \text{otherwise}\end{cases}$$

where **div** is the integer division operator and **mod** is the remainder operator for integer division. The u_i's are assigned to points of a $\sqrt{Q}\times\sqrt{Q}$ coordinate indexed grid such that u_j's with nonzero $a_{i,j}$'s are within distance one of u_i. Each u_i is updated on odd/even values of k according to $\Lambda_{i,k}$. An odd/even iteration cycle is sometimes called two half-iterations. The remainder of this paper considers (without loss of generality) consistent multicolor orderings of length two. The pointwise SSOR method is given by the following equation for fixed ω.

$$u_i^{(k)}=(1-\omega)u_i^{(k-2)}-\omega\frac{1}{a_{i,i}}\left[\sum_{\substack{j\neq i\\ j=1}} a_{i,j}u_j^{(k-1)}-v_i\right] \quad \text{if } \Lambda_{i,k}=1 \qquad (1)$$

The following theorem characterizes matrices which can be solved by (1).

Theorem 1: [Varg62] Let **A** be a strictly or irreducibly diagonally dominant Hermitian $Q\times Q$ complex matrix. Then the pointwise SSOR method with $0<\omega<2$ is convergent for any initial vector $\mathbf{u}^{(0)}$.

In the case of a two-dimensional grid, we can think of the values of $\mathbf{u}^{(K)}$ on the right hand side of (1) as neighbors of the point u_i at the compass directions N, S, E, and W. Each new value of a point is iteratively a function of the current values of these four neighbors, the locations of which are N: $(i \textbf{ div } \sqrt{Q}+1, i \textbf{ mod } \sqrt{Q})$, S: $(i \textbf{ div } \sqrt{Q}-1, i \textbf{ mod } \sqrt{Q})$, E: $(i \textbf{ div } \sqrt{Q}, i \textbf{ mod } \sqrt{Q}+1)$, and W: $(i \textbf{ div } \sqrt{Q}, i \textbf{ mod } \sqrt{Q}-1)$.

Ames [Ames77] gives a bound on the rate of error reduction in a matrix iterative solution as a function of the spectral radius ρ_G of the iteration matrix G. The first step in this analysis is to determine the spectral radius ρ_G of the iteration matrix **G**. While in general it is not feasible to calculate ρ_G directly from **G**, for analytical purposes this calculation is sufficient. For the red-black ordering used in this paper, **G** cannot be expressed in a convenient matrix form. Instead we use the **G** from the Gauss-Seidel consistent ordering with Successive Over-Relaxation applied. The matrix **A** is decomposed into **A=B+D+C** with **B** lower triangular, **C** upper triangular, and **D** diagonal $Q\times Q$ matrices. Thus **G** becomes,

$$\mathbf{G}=(\mathbf{D}+\omega\mathbf{B})^{-1}[(1-\omega)\mathbf{D}-\omega\mathbf{C}]$$

If the complex eigenvalues of **G** are given by λ_i for $i\in\{1,...,Q\}$, then

$$\rho_G=\max_{i\in\{1,...,Q\}}|\lambda_i|$$

Let the error at step k of the computation be denoted by the vector $\boldsymbol{\varepsilon}^{(k)}=|\mathbf{u}^k-\mathbf{u}|$ for $k\in\{0,1,2,...\}$. For a stationary linear iteration matrix **G** of the form considered in Theorem 1, $\|\boldsymbol{\varepsilon}^{(k)}\|=\|\mathbf{G}^k\boldsymbol{\varepsilon}^{(0)}\|$ where $\|\boldsymbol{\varepsilon}^{(k)}\|$ is the spectral norm of $\boldsymbol{\varepsilon}^{(k)}$. If $\rho_G<1$, for k sufficiently large, $\|\mathbf{G}^k\|\simeq[\rho_G]^k$ where $\|\mathbf{G}\|$ is the spectral norm of **G**. Combining these results yields [Ames77]

$$\|\boldsymbol{\varepsilon}^{(k)}\|\leq\|\mathbf{G}^k\|\,\|\boldsymbol{\varepsilon}^{(0)}\|$$

Thus, for large k the ratio $\|\boldsymbol{\varepsilon}^{(k+1)}\|/\|\boldsymbol{\varepsilon}^{(k)}\|$ averages to ρ_G. Therefore, on the average, the error decreases by a factor of ρ_G at each step in the iteration.

The following is a sequential algorithm implementing (1).

Procedure Relax$(u_i^{(0)})$

$k\leftarrow 0$ /* Iteration Count */
/* u_i^0 is the initial value */

while $(\|\mathbf{u}^{(k)}-\mathbf{u}^{(k-2)}\|>\varepsilon)$
  for(i=1;i<N;i++)
    if $(\Lambda_{i,k}=1)$
      $$u_i^{(k+1)}\leftarrow(1-\omega)u_i^{(k-1)}-\omega\frac{1}{a_{i,i}}\left[\sum_{\substack{j\neq i\\ j\in\{N,S,E,W\}}} a_{i,j}u_j^{(k)}-v_i\right];$$
  $k\leftarrow k+1;$

Since the scheme is convergent, as $k\to\infty$ $\|\mathbf{u}^{(k+2)}-\mathbf{u}^{(k)}\|\to 0$ and indeed $\|\mathbf{u}^{(k)}-u\|\to 0$. Additionally, since the error decreases by ρ_G, we can assert that the relative solution makes ever decreasing progress as it approaches the final solution. Clearly, the necessary assertions are (1) loop termination, and (2) correctness of the calculation.

Loop termination is guaranteed by the following assertion:

Assertion (1)
  if $k\geq 2$
    if $\|\mathbf{u}^{(k+2)}-\mathbf{u}^{(k)}\|<\|\mathbf{u}^{(k)}-\mathbf{u}^{(k-2)}\|$ **then** ERROR;

Correctness of the calculation is verified by making sure that at every step the intermediate result is locally a possible solution.

Assertion (2)
  if $a_{i,i}u_i^{(k+2)}+\sum_{j\neq i}^{N}a_{i,j}u_j^{(k)}\neq v_i$ **then** ERROR;

The transformed program (with some small additional assertions on loop variables) is:

Procedure Relax$(u_i^{(0)})$
$k\leftarrow 0$
$k_{last}\leftarrow 0$
while $(\|\mathbf{u}^{(k)}-\mathbf{u}^{(k-2)}\|>\varepsilon)$
  [Assertion (1)]
  for(i=1;i<N;i++)
    [**if** (i > N) **then** ERROR]

if ($\Lambda_{i,k}$=1)

$$u_i^{(k+1)} \leftarrow (1-\omega)u_i^{(k-1)} - \omega\frac{1}{a_{i,i}}\left[\sum_{\substack{j\neq i \\ j\in\{N,S,E,W\}}} a_{i,j}u_j^{(k)} - v_i\right];$$

[Assertion (2)]

$k \leftarrow k+1$

[**if** $k_{last} \neq$ k-1] **then** ERROR]

$k_{last} \leftarrow k$

Notice that the algorithm can also easily be parallelized. Assume $Q=N$ for simplicity and let processor $P_i, i=1,...,N$ compute value u_i. The case for $Q>N$ differs only in performance and the corresponding analysis is performed in the full paper [McMi88]. Since each new value $u_i^{(k+2)}$ is a function only of the previous values $\mathbf{u}^{(k)}$, each point can be calculated in parallel by the following algorithm:

Procedure Relax($u_i^{(0)}$)
/* For Processor at location P_i */
/* u_i is the data held by P_i */

$k \leftarrow 0$ /* Iteration Count */
/* u_i^0 is the initial value */

while ¬ converged
foreach (*direction* **in** {N, S, E, W})
if ($\Lambda_{i,k}$=1)
receive $u_{direction}^{(k)}$ from $P_{direction}$;
else
send $u_i^{(k)}$ to $P_{direction}$;
if ($\Lambda_{i,k}$=1)

$$u_i^{(k+1)} \leftarrow (1-\omega)u_i^{(k-1)} - \omega\frac{1}{a_{i,i}}\left[\sum_{\substack{j\neq i \\ j\in\{N,S,E,W\}}} a_{i,j}u_j^{(k)} - v_i\right];$$

$k \leftarrow k+1$;

Any attempt to apply assertions (1) and (2) fails in this case. Assertion (1) requires that all values of the calculation be available to all processors at every iteration. This is extremely expensive in terms of run time since obtaining the necessary data in a distributed system requires a large number of message exchanges. A more fundamental problem exists in the unreliable distributed multiprocessing environment as there is no reason to expect that the values received as messages bear any relation to a correct value. A faulty processor can send any value it wants during an iteration and prevent termination or force termination to an incorrect result. Since we have this imperfect information, assertion (1) is useless and assertion (2) cannot be reliably computed.

This motivates the consideration of assertions that are not so "myopic" in that the development is not limited to the program design/coding phases. Global perspective is certainly achieved by assertion (1) but is not implementable. What is needed is a way to extract assertions that can be computed during execution which preserve the global perspective of the problem. To accomplish this, the specification phase is included in assertion development. This is discussed in the next section.

3. ASSERTION DEVELOPMENT

Hua and Abraham [HuAb87] noticed that assertions developed during the design process are superior to those developed at the coding level since the former can also be used to detect coding faults during the verification phase of the life cycle. This leads to speculation that the earlier the assertions enter the life cycle, the more comprehensive the error detection. However, what is necessary is a systematic approach for extraction of assertions from these higher levels.

Guidance in the selection of executable assertions may be provided by "Natural Constraints" of the problem. Working from these natural constraints starting at the specification phase provides a global perspective on assertion development. The parallel decomposition of the problem ties in closely with the design of the program to solve the problem. Thus the executable assertions are closely tied to the individual components of the life cycle. This relationship is shown in Figure 1.

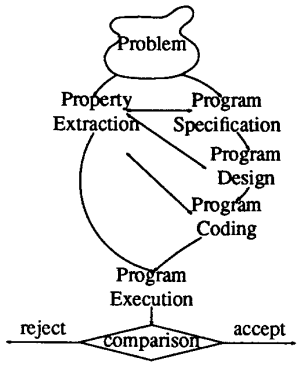

Figure 1. Parallel Executable Assertion Development

We now present a set of high-level basis metrics used to extract assertion generating properties from the problem specification. In a solution, each testable intermediate result should satisfy one or more of the subclasses of *progress, feasibility*, and *consistency*. The three subclasses form the class of *constraint predicates* in which each subclass contributes to form a cohesive unit that provides error coverage. The constraint predicate satisifies both the "liveliness" and "safety" properties of [Hail82]. Liveliness ensures that the solution makes progress - i.e. does not stall or deadlock. Safety ensures that the solution obtained will be the correct solution.

(1) *Progress*

We require progress to be made at each testable step of the solution. Progress means that the state of the solution advances to the goal or final solution of the problem. Each testable step of the solution is defined as a message interchange, sub-message interchange, or multi-message interchange dependent upon the type of solution. If this progress is not made, then faulty behavior may indefinitely postpone the solution - any solution, even an incorrect solution.

Problems of an iterative nature must be convergent. We shall assume that the problems attempted are convergent, for if not, then a nonconvergent result is indistinguishable from a hardware failure. Most, if not all, problem solutions contain implicit information necessary to form a *convergence envelope*. This places bounds on the reduction of error between the current and final solution. Let the error at step k of the solution be

$$\boldsymbol{\varepsilon}^{(k)} = |\mathbf{u}^{(k)} - \mathbf{u}|$$

where k is the current step count of the solution, $\mathbf{u}^{(k)}$ is a single or vector-valued state of the solution at the k'th step, and $\mathbf{u}$ is the actual correct solution. Monotonic reduction of error is then defined as

$$\|\varepsilon^{(k+1)}\| < \|\varepsilon^{(k)}\|$$

$$\|\mathbf{u}^{(k+1)}-\mathbf{u}\| < \|\mathbf{u}^{(k)}-\mathbf{u}\| \quad (2)$$

However, any test based on this relation requires all the values held by all the processors involved in the calculation. For a mesh of N processors, this takes $O(\sqrt{N})$ time. The second difficulty is that it involves the unknown $\mathbf{u}$. What is more desirable is a constraint predicate that can test locally in $O(1)$ time or even unit time and furthermore use only obtainable data for its tests. Locality and obtainability cooperate to form a test for monotonic convergence based on a single message interchange.

Consider as a special case of the convergence envelope, localized monotonic reduction of error. This is typically seen in relaxation solutions such as PDE relaxation. Without loss of generality assume that the sequence of local errors $\varepsilon_i^{(k)}$ ($\varepsilon^{(k)} = (\varepsilon_i^{(k)})$, $i \in \{1,2,...,N\}$) is monotone decreasing. Let $u_i^{(k)}$ be the value of the i'th component of the k'th step of vector $\mathbf{u}$.

$$\varepsilon_i^{(k+1)} \le \varepsilon_i^{(k)}$$

$$u_i^{(k+1)}-u_i \le u_i^{(k)}-u_i \quad (3)$$

It then follows immediately that the sequence of u_i's are also monotonically decreasing. A suitable constraint predicate $D_{M'}$ for this relation is:

Predicate $D_{M'}$ ($u_i^{(k)}, u_i^{(k+1)}$, **real**, i,k) **returns Boolean**;
  if ($u_i^{(k+1)} \ge u_i^{(k)}$)
    then return ERROR;
  else
    return correct;
end.

Convergence envelopes, while ensuring a guarantee of nonegative progress, cannot check for *sufficient progress*. By this we mean that a solution satisfying the above predicate may proceed arbitrarily slowly. Consider *bounds* on the convergence rate $\gamma_{min}^{(k)}$ and $\gamma_{max}^{(k)}$ such that each step satisfies

$$\gamma_{min}^{(k)} \le \frac{\|\varepsilon^{(k)}\|}{\|\varepsilon^{(0)}\|} \le \gamma_{max}^{(k)} \quad (4)$$

A corresponding local rate bound may be obtained and implemented similarly to that of (2).

Progress alone is not sufficient to guarantee solution correctness. The final vector, $\mathbf{u}^{(k)}$, particularly when the bounds γ are loose, may differ significantly from the actual solution $\mathbf{u}$. To restrict completely arbitrary behavior further, we consider feasibility as the next constraint predicate basis.

(2) *Feasibility*

Each testable result must remain within the defined solution space of the problem. Formally, consider a solution space H and any intermediate result $\mathbf{u}^{(k)}$. Then for any step k,

$$\mathbf{u}^{(k)} \in H_k$$

The feasibility constraints are often immediately apparent from the nature of the problem studied. Problems in physics and engineering, such as equilibrium problems, eigenvalue problems, and to a lesser extent propagation problems, contain feasibility constraints in the form of boundary conditions. Indeed, these boundary conditions are exactly the class of natural problem constraints. The boundary conditions are of course known a priori and do not vary as the solution progresses ($H_k = H$ is stationary), and thus can easily be used as feasibility constraints.

Branch and bound tree searching is a method which searches a state space (tree) for a minimum cost goal state. This search is guided by an accumulated and estimated cost function. Let $c^{(k)}(x)$ be the estimate of the cost function at step k and $c(x)$ be the actual cost of reaching the goal state from node x. If we enforce that $c^{(k)}(x) \le c(x)$ for $k \in \{1,2,...\}$, then $c^{(k)}(x)$ provides a lower bound in H_k on the cost of any solution obtainable from node x. Let L be an upper bound on the cost of a minimum cost solution (initially may be ∞). If at some step k, a possible goal node y is discovered, then all nodes x with $c(x) \ge c^{(k)}(x) > c(y)$ may be killed, and L is updated as $L=c(y)$. Thus branch and bound provides a way of dynamically narrowing the bounds on the feasible solutions H_k.

(3) *Consistency*

Many intermediate calculations contain additional properties that are indirectly obtainable from the problem's natural constraints. These are defined as *Consistency Conditions*. In a white box testing environment, all facets of each testable step can be checked. As noted previously, in the DMMP environment, due to the locality of information, black box testing must be utilized. A consistency predicate may be applied only to the received information and locally known information.

Consistency can compensate for limitations in the progress and feasibility bases. Consider a problem in which (4) has a globally specified bound γ but no locally specifiable γ_i. Furthermore, from the discussion in Section 2 we know that any global assertions are unimplementable. Knowledge that a processor has about its local state and the values that it sent to other processors in previous steps can provide bounds on the range of acceptable values for the current testable step.

Assume that the solution proceeds as in (3). Let each new value of $u_i^{(k)}$ be calculated as a linear function f of the neighboring values $u_l^{(k-1)}$, $l \in R$ with coefficient a_l. From the perspective of a processor P_i calculating u_i, a candidate intermediate result $u_l^{(k)}$, $l \in R$ must satisfy the following property for $l,i = 1,2,...,N$, $k \in \{2,3,4,...\}$

$$u_l^{(k)} \le u_l^{(k-1)} - a_l(u_i^{(k-2)} - u_i^{(k-1)}) \quad (5)$$

To prove this as a theorem it is necessary to consider two successive iterations $u_l^{(k)}$ and $u_l^{(k-1)}$. Subtracting the two yields an expression which is a function of known and unknown values (from processor P_i's point of view). If we let $W^{(k)}$ represent the unknown values at iteration k, the expression becomes

$$u_l^{(k)} - u_l^{(k-1)} = W^{(k)} - u^{(k-1)} + a_l(u_i^{(k-1)} - u_i^{(k-2)})$$

Since f is a monotonic linear function and $u_l^{(k)} \le u_l^{(k-1)}$ where $l,i = 1,2,...,N$, $k \in \{1,2,...\}$, then $W^{(k-1)} \ge W^{(k)}$. This provides a lower bound on the amount of movement that processor P_i can expect which results in the inequality given in (5). The lower bound is solely a function of the values that P_i sent P_l in the previous iterations and P_l's previous iteration.

4. PARALLEL ASSERTION DEVELOPMENT - A CASE STUDY

We now continue the example begun in Section 2.

Natural Problem/Solution Constraints

Constraint predicate development proceeds as outlined in Section 3. First the mathematical properties of the problem and its solution must be espoused. Theorem 1 allows the choice of the initial assignment vector $\mathbf{u}^{(0)}$ to be arbitrary. The next theorem shows that certain choices can facilitate constraint predicate development.

Theorem 2: Let **A** be a matrix satisfying Theorem 1 with positive diagonal and nonpositive off-diagonal entries. Then the solution given by (1) will converge monotonically to the final solution $\mathbf{u}^{(K)}$ for the following choices of $\mathbf{u}^{(0)} = (u_i^{(0)})$ for $\omega = 1$.

1) Let $u_{\min}$ be the smallest value such that $v_i - u_{\min} \sum_{j=1}^{Q} a_{i,j} \le 0$ for all $i \in \{1,2,...,Q\}$.
If $u_i^{(0)} = u_{\min}$ for all $i \in \{1,2,...,Q\}$ then $u_i^{(k)} \ge u_i^{(k+2)}$ for all $k \in \{0,1,...\}$.

2) Let $u_{\max}$ be the largest value such that $v_i - u_{\max} \sum_{j=1}^{Q} a_{i,j} \ge 0$ for all $i \in \{1,2,...,Q\}$.
If $u_i^{(0)} = u_{\max}$ for all $i \in \{1,2,...,Q\}$ then $u_i^{(k+2)} \ge u_i^{(k)}$ for all $k \in \{0,1,...\}$.

Proof: For case 1), by induction on k, the iteration count.

Basis, k=0,

$$u_i^{(1)} = \frac{1}{a_{i,i}} \left[v_i - \sum_{j \ne i}^{Q} a_{i,j} u_i^{(0)} \right] \tag{6}$$

and since $u_{\min}$ is the initial value,

$$v_i - \sum_{j \ne i}^{Q} a_{i,j} u_i^{(0)} - a_{i,i} u_i^{(0)} \le 0 \tag{7}$$

Since $a_{i,i} > 0$, dividing (7) by $a_{i,i}$, simplifying and substituting in (6) yields

$$u_i^{(1)} = \frac{1}{a_{i,i}} \left[v_i - \sum_{j \ne i}^{Q} a_{i,j} u_i^{(0)} \right] \le u_i^{(0)} \tag{8}$$

Continuing to the completion of the half iteration pair, by (7,8) and since $a_{i,j} \le 0$, $i \ne j$; $i,j \in \{1,2,...,Q\}$

$$\le \frac{1}{a_{i,i}} \left[v_i - \sum_{j \ne i}^{Q} a_{i,j} u_j^{(0)} \right] \le u_i^{(0)}$$

Now assume $u_i^{(l)} \le u_i^{(l-2)}$ for $l = 2,3,...,k-1,k$. Then for $l = k+1$

$$u_i^{(k+1)} = \frac{1}{a_{i,i}} \left[v_i - \sum_{j \ne i} a_{i,j} u_j^{(k)} \right]$$

Since $a_{i,j} \le 0$, $i \ne j$,

$$\le \frac{1}{a_{i,i}} \left[v_i - \sum_{j \ne i} a_{i,j} u_j^{(k-2)} \right] \le u_i^{(k-1)}$$

The proof for case (2) is symmetric. □

Theorem 3: In the solution of (1), each $u_i^{(k)}$, $i \in \{1,2,...,Q\}$, $k \in \{0,1,...\}$ is bounded by $u_{\min} \ge u_i^{(k)} \ge u_{\max}$ for $u_i^{(k)}$, $u_i^{(k+2)}$ satisfying the conditions of Theorem 2.

Proof: W.l.o.g. let the initial guess be $u_i^{(0)} = u_{\min}$, $i \in \{1,...,Q\}$. One of two cases can occur.

1) By Theorem 2 $u_i^{(k)} \le u_i^{(k-2)}$, $i \in \{1,...,Q\}$, $k \in \{2,3,4,....\}$. Thus

$$u_i^{(k)} \le u_i^{(k-2)} \le \cdots \le u_i^{(0)} \quad \text{for } k \text{ even}$$
$$u_i^{(k)} \le u_i^{(k-2)} \le \cdots \le u_i^{(1)} \quad \text{for } k \text{ odd}$$
$$\le u_i^{(0)} \le U_{\min} \text{ by (8)}$$

2) Now assume that $u_i^{(k)} < u_{\max}$ for some $i \in \{1,2,...,Q\}$ and $k \in \{0,1,2,...\}$ and $u_i^{(l)} \ge u_{\max}$ for $l = 0,...,k-1$. By (1)

$$v_i - \sum_{j \ne i}^{Q} a_{i,j} u_j^{(k-1)} - a_{i,i} u_{\max} < 0$$

The choice of $u_{\max}$ satisfies

$$v_i - \sum_{j \ne i}^{Q} a_{i,j} u_{\max} - u_{\max} \ge 0$$

Rearranging each yields

$$\sum_{j \ne i}^{Q} a_{i,j} u_j^{(k-1)} > v_i - a_{\cdot,i} u_{\max}$$
$$\sum_{j \ne i}^{Q} a_{i,j} u_{\max} \le v_i - a_{i,i} u_{\max}$$

Substituting,

$$\sum_{j \ne i}^{Q} a_{i,j} u_j^{(k-1)} > \sum_{j \ne i}^{Q} a_{i,j} u_{\max}$$

Since $a_{i,j} \le 0$ for $i,j \in \{1,2,...,Q\}$, $u_j^{(k-1)} < u_{\max}$ for at least one $u_j^{(k-1)}$, a contradiction.

The case for the initial choice of $u_{\max}$ is symmetric. □

Theorem 4: Consider a sequence of $u_i^{(k)}$'s satisfying the conditions of Theorem 2. From the perspective of a processor P_i calculating u_i, a candidate intermediate result $u_l^{(k)}$ with a nonzero coefficient $a_{l,i}$, $l,i \in \{1,2,...,Q\}$ must satisfy the following properties for $k \in \{0,1,2,...\}$

If $u_i^{(0)} = u_{\min}$, $i \in \{1,2,\cdots,Q\}$, then

$$u_l^{(k+2)} \le u_l^{(k)} + (1-\omega)(u_l^{(k)} - u_l^{(k-2)}) - \omega \frac{1}{a_{l,l}} \left[a_{l,i} (u_i^{(k+1)} - u_i^{(k-1)}) \right]$$

Similarly if $u_i^{(0)} = u_{\max}$, $i \in \{1,2,\cdots,Q\}$, then

$$u_l^{(k+2)} \ge u_l^{(k)} + (1-\omega)(u_l^{(k)} - u_l^{(k-2)}) - \omega \frac{1}{a_{l,l}} \left[a_{l,i} (u_i^{(k+1)} - u_i^{(k-1)}) \right]$$

Proof: Consider two successive iterations at k and $k+2$.

Subtracting $u_l^{(k+2)} - u_l^{(k)}$

$$u_l^{(k+2)} - u_l^{(k)} = (1-\omega)(u_l^{(k)} - u_l^{(k-2)})$$
$$- \omega \frac{1}{a_{l,i}} \left[\sum_{j \ne l, j \ne i} a_{l,j} u_j^{(k+1)} + a_{l,l} u_i^{(k+1)} - \left(\sum_{j \ne l, j \ne i} a_{l,j} u_j^{(k-1)} + a_{l,i} u_i^{(k-1)} \right) \right]$$

Since $u_l^{(k-1)} \ge u_l^{(k+1)}, k \in \{1,2,...\}$

$$\le (1-\omega)(u_l^{(k)} - u_l^{(k-2)}) - \omega \frac{1}{a_{l,l}} (a_{l,i} u_i^{(k+1)} - a_{l,i} u_i^{(k-1)})$$

The case when $u_i^{(0)} = u_{\max}$, $i \in \{1,2,\cdots,Q\}$ is symmetric. □

Theorem 5: A candidate intermediate result $u_l^{(k)}$ with nonzero coefficient $a_{l,i}$, from the perspective of processor calculating $u_i^{(k)}$, $l,i \in \{1,2,...,Q\}$, must satisfy the following properties.

If $u_i^{(0)} = u_{\min}$, $i \in \{1,2,\cdots,Q\}$, then

$$u_l^{(k+2)} \ge (1-\omega) u_l^{(k)} - \omega \frac{1}{a_{l,l}} \left[\sum_{j \ne l, j \ne i} a_{l,j} u_{\max} + a_{l,i} u_i^{(k+1)} - v_l \right]$$

Similarly if $u_i^{(0)} = u_{\max}$, $i \in \{1,2,\cdots,Q\}$, then

$$u_l^{(k+2)} \le (1-\omega) u_l^{(k)} - \omega \frac{1}{a_{l,l}} \left[\sum_{j \ne l, j \ne i} a_{l,j} u_{\min} + a_{l,i} u_i^{(k+1)} - v_l \right]$$

Proof: Immediate from the conditions of Theorem 2 and the proof of Theorem 4 □.

Theorems 4 and 5 guarantee a minimal and maximal amount of progress of each individual component of the solution as a function of the current state of the solution.

The stopping point is selected to be the iteration K at which $|u_i^{(K+2)}-u_i^{(K)}|<\varepsilon$ for $i=1,2,3,...,Q$ for some small ε.

Theorem 6: At the end of the final iteration K, the final result $u_i^{(K)}$ satisfies the following relations

If $u_i^{(0)}=u_{\min}$, $i\in\{1,2,\cdots,Q\}$, then

$$u_i^{(K)} \geq \frac{1}{a_{i,i}}\left[\sum_{j\neq i}^{Q} a_{i,j}u_j^{(K)}-v_i\right] \geq u_i^{(K)}-\frac{\varepsilon}{\omega}$$

and if $u_i^{(0)}=u_{\max}$, $i\in\{1,2,\cdots,Q\}$, then

$$u_i^{(K)} \leq \frac{1}{a_{i,i}}\left[\sum_{j\neq i}^{Q} a_{i,j}u_j^{(K)}-v_i\right] \leq u_i^{(K)}+\frac{\varepsilon}{\omega}$$

Proof: Consider the final iteration at $K+2$. Assume that $u_i^{(0)}=u_{\min}$, $i\in\{1,2,\cdots,Q\}$.

$$u_i^{(K+2)}-(1-\omega)u_i^{(K)} = -\omega\frac{1}{a_{i,i}}\left[\sum_{j\neq i}^{Q} a_{i,j}u_j^{(K+1)}-v_i\right]$$

Since $u_i^{(K)}-u_i^{(K+2)}<\varepsilon$,

$$\frac{\varepsilon}{\omega} > u_i^{(K)}-\frac{1}{a_{i,i}}\left[\sum_{j\neq i}^{Q} a_{i,j}u_j^{(K+1)}-v_i\right]$$

The case when $u_i^{(0)}=u_{\max}$, $i\in\{1,2,\cdots,Q\}$ is symmetric □

Error Coverage Modeling for Matrix Iterative Analysis

Error coverage modeling is necessary not only to ensure adequate error coverage but to detect those assertions which may be completely *dominated*. An assertion is dominated if its error coverage is a subset of another assertion. Probabilistic measures of coverage are attractive as assertion dominance may not be apparent from the assertion text.

As in the development of the constraint predicate, the development of the modeling of each individual feature is treated separately.

Feasibility

Modeling of the error coverage of the feasibility constraints for the problem considered in this paper is straightforward. Since the feasibility constraints H_k of Theorem 3 are stationary over all iterations k, define:

${}^{j}F$: Event that $u_{\max} \leq u_j^{(k)} \leq u_{\min}$ for j and k.

The cumulative distribution function of the appropriate model yields the coverage probability for all steps of the problem:

$$Pr[{}^{j}F] = Pr[u_{\max} \leq u_j^{(k)} \leq u_{\min}]$$

Progress

Progress is modeled by considering the global convergence properties of the solution and extrapolating this behavior back to each individual case. For modeling purposes, the final result is known. Using the relation for ρ_G developed in Section 2, the rate of convergence to the final solution may be obtained. Assuming that each individual point behaves according to global convergence and using Theorem 2, treatment of an arbitrary point, i', yields the following relation

If $u_i^{(0)}=u_{\min}$,

$$\rho_G^k(u_{\min}-u_{i'}) \geq u_i^{(k)}-u_{i'},\ k\in\{0,1,2,...\} \qquad (9)$$

where $u_{i'}$ is the correct final result of the calculation. A similar result holds if $u_i^{(0)}=u_{\max}$. The events P_i that define progress are

If $u_i^{(0)}=u_{\min}$:

${}^{i'}P_1^{(k)}$: Event $u_i^{(k)} \leq \rho_G^k(u_{\min}-u_{i'})+u_{i'}$, $k\in\{0,1,2,...\}$

If $u_i^{(0)}=u_{\max}$:

${}^{i'}P_2^{(k)}$: Event $u_i^{(k)} \geq u_{i'}-\rho_G^k(u_{i'}-u_{\max})$, $k\in\{0,1,2,...\}$

Consistency

The consistency components of Theorems 4 and 5, like the progress component, can either be determined from a trace or by analytic means. For the analytic model, we apply Theorem 4 with the assumption that each $u_i^{(k)}$ proceeds toward the solution $\mathbf{u}$ at the same convergence rate. Again this is not the case in the actual solution but does effectively display the aggregate behavior of the solution. While Theorems 4 and 5 are directly implementable as a constraint predicate, to use them in the model requires additional simplification. The ratio $a_{l,i}/a_{l,l}$ is aggregated by replacing it with the ratio $R=u_i^{(k)}/Wu_i^{(k-1)}$ where W is the average number of nonzero $a_{l,i}$'s. Using relation (9) as an equality and substituting into Theorem 4 yields the events ${}^{i'}C_{1a}^{(k+2)}$ and ${}^{i'}C_{1b}^{(k+2)}$

If $u_i^{(0)}=u_{\min}$, ${}^{i'}C_{1a}^{(k+2)}$ is the event

$$u_i^{(k+2)} \leq \rho_G^k(u_{\min}-u_{i'})-u_{i'}(1-\omega)(\rho_G^k(u_{\min}-u_{i'})-\rho_G^{k-2}(u_{\min}-u_{i'}) -\omega R(\rho_G^{k+1}(u_{\min}-u_{i'})-\rho_G^{k-1}(u_{\min}-u_{i'})),\ k\in\{2,3,4,...\}$$

and if $u_i^{(0)}=u_{\max}$ ${}^{i'}C_{1b}^{(k+2)}$ is the event

$$u_i^{(k+2)} \geq \rho_G^k(u_{\min}-u_{i'})-u_{i'}(1-\omega)(\rho_G^k(u_{\min}-u_{i'})-\rho_G^{k-2}(u_{\min}-u_{i'}) -\omega R(\rho_G^{k+1}(u_{\min}-u_{i'})-\rho_G^{k-1}(u_{\min}-u_{i'})),\ k\in\{2,3,4,...\}$$

Letting the mean of the v_i's be $\bar{v}$, substitution into Theorem 5 yields the events ${}^{i'}C_{2a}^{(k+2)}$ and ${}^{i'}C_{2b}^{(k+2)}$.

If $u^{(0)}=u_{\min}$, ${}^{i'}C_{2a}^{(k+2)}$ is the event

$$u_i^{(k+2)} \geq (1-\omega)(\rho_G^k(u_{\min}-u_{i'})+u_{i'} -\omega R\left[\sum_{j\neq l, j\neq i}^{N} u_{\max}+\rho_G^{k+1}(u_{\min}-u_{i'})+u_{i'}-\bar{v}\right],\ k\in\{2,3,4,...\}$$

and if $u_i^{(0)}=u_{\max}$, ${}^{i'}C_{2a}^{(k+2)}$ is similar.

With these relationships defined, the modeling can proceed. Choose as an example problem the matrix shown in Figure 2 which satisfies the conditions of Theorem 1. The spectral radius ρ_G of the corresponding iteration matrix $\mathbf{G}$ is 0.4775 for $\omega=1$. It is easily verified that $u_{\min}=1.00$ (from rows 5 or 6 of $\mathbf{A}$) and $u_{\max}=.057$ (from row 4 of $\mathbf{A}$)†.

The normal distribution function is chosen as the conditional distribution of erroring values. If we center the mean of the distribution at the actual correct result of the calculation, then the normal with mean μ and standard deviation σ is an attractive choice since the erroring values tend to group near the correct value. While an overly pessimistic assumption, it does display the effectiveness of the generated constraint predicate. Figure 3 depicts the feasibility event F and its relation to the conditional distribution of erroring values. $\mu = .562667$ is equal to the norm of the final solution vector $\|\mathbf{u}^{(k)}\|/Q$. $\sigma = .65$ is assigned such that more than half of the error probability is undetectable by the feasibility conditions.

Let E be the event that an error has occurred. Figure 4 depicts the error coverage as a function of the iteration step k for three different levels of error coverage. The error coverage of the feasibility predicate is constant since the constraints are stationary. The progress component rapidly rises to it's limiting value of approxi-

†Note that due to truncation of the values shown these values do not correspond exactly with the matrix shown.

mately 25% additional coverage. This is because the average solution makes the greatest reduction of error in the early steps of the solution. The consistency component adds only a few more percent to the error coverage to bring the total error coverage to approximately 75%.

$$A = \begin{bmatrix} 4.87 & -1.28 & 0 & -1.37 & 0 & 0 & 0 & 0 & 0 \\ -1.28 & 5.62 & -1.78 & 0 & -1.37 & 0 & 0 & 0 & 0 \\ 0 & -1.78 & 6.87 & 0 & 0 & -1.37 & 0 & 0 & 0 \\ -1.37 & 0 & 0 & 5.37 & -1.28 & 0 & -1.62 & 0 & 0 \\ 0 & -1.37 & 0 & -1.28 & 6.12 & -1.78 & 0 & -1.62 & 0 \\ 0 & 0 & -1.37 & 0 & -1.78 & 7.37 & 0 & 0 & -1.62 \\ 0 & 0 & 0 & -1.62 & 0 & 0 & 5.87 & -1.28 & 0 \\ 0 & 0 & 0 & 0 & -1.62 & 0 & -1.28 & 6.62 & -1.78 \\ 0 & 0 & 0 & 0 & 0 & -1.62 & 0 & -1.78 & 7.87 \end{bmatrix}$$

$$v = \begin{bmatrix} 0.343750 \\ 0.625000 \\ 3.437500 \\ 0.062500 \\ 0.062500 \\ 2.593750 \\ 0.179688 \\ 0.531250 \\ 3.648438 \end{bmatrix}$$

Figure 2. Sample Matrix for Au=v

Matrix formed from the finite difference solution of the self-adjoint elliptic PDE $\frac{\partial}{\partial x}[(x+1)\frac{\partial u}{\partial x}]+\frac{\partial}{\partial y}[(y^2+1)\frac{\partial u}{\partial y}]-u=1$ on the unit square with Dirichlet Boundary conditions:

$u(0,y)=y$, $u(1,y)=y^2$, $u(x,0)=0$, $u(x,1)=1$.

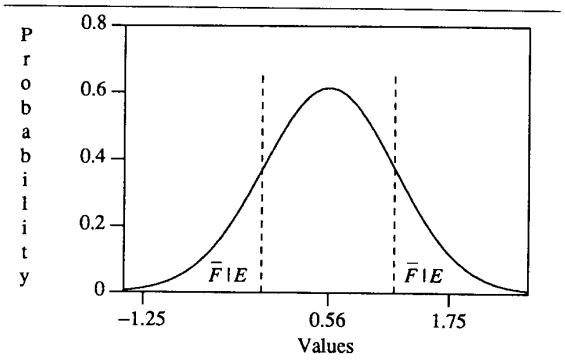

Figure 3. Feasibility Event and Conditional Distribution of Errors

Seventy-five percent of the errors generated by a faulty processor can be flagged with a fault latency of one using only predicates that test on a single message. It is interesting to note that while the addition of the consistency constraints only provides a small gain in error coverage, it is still a necessary component. The constraints provided by Theorems 4 and 5 guarantee that a faulty processor must make some non-negligible movement toward the solution. While not modeled here, empirical data suggest that a faulty processor can delay convergence almost indefinitely by chosing a movement just slightly large than ε. The consistency constraint, when viewed in this manner, is a necessary component of the constraint predicate.

The choice of the normal as the distribution of errors also limits the error coverage using these predicates. Any symmetric distribution centered at the final result will necessarily result in an error coverage measurement somewhat less that total coverage since the tests developed (except for the consistency test based on Theorem 5) are primarily one-sided.

Increasing Error Coverage

The predicate formed by Theorem 6 has not yet been used in the constraint predicate. Theorem 6 affords total test coverage if certain additional restrictions are imposed.

From the application view, a *Reliable Broadcast Environment* guarantees that all recipients of a message receive the same version of that message and the message received is the message sent. This can be achieved through the use of a reliable broadcast protocol as proposed by [ChMa84] or through the method proposed by [McNi87] which integrates with the application oriented environment.

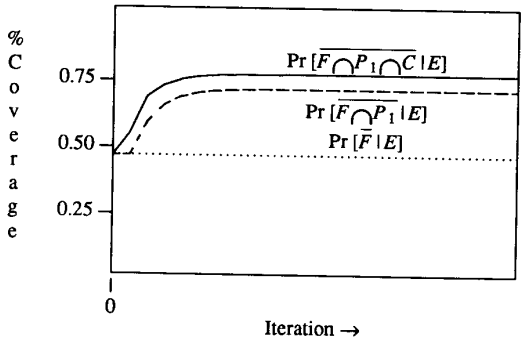

Figure 4. Error Coverage by Solution Step Count

The second requirement is to place a bound on the number of failures that may occur in the system. This is implicit in most fault-detection and fault-tolerance schemes but we have not treated it from the application level until now.

Definition 3: *Local Fault Group*: $\Omega_i = \{P_i | a_{i,j} \neq 0,\ j=1,\ldots,N\}$.

Thus a fault group $\Omega_{i'}$ for some i' is the set of processors involved directly in the solution of (2.1) for point $u_{i'}$.

The diagnosis of a fault group is stated as follows. If some processor in $\Omega_{i'}$ is faulty for some i', then $\Omega_{i'}^D \equiv 1$ and the entire fault group is flagged as faulty; otherwise, $\Omega_{i'}^D \equiv 0$ and the entire fault group is flagged as non-faulty.

Lemma 1: Let the processors in a local fault group $\Omega_{i'}$ be capable of a reliable broadcast. Then processors in $\Omega_{i'}$ reliably diagnose $\Omega_{i'}$ as either faulty or non-faulty if the maximum number of faulty units per fault group is 1.

Proof: The proof is constructive.

Step 1: Each Processor $P_j \in \Omega_{i'}$ sends its last value of $u_j^{(K)}$ to the other members of $\Omega_{i'}$.

Step 2: If $|u_i^{(K-2)} - u_i^{(K)}| > \varepsilon$ then report that $\Omega_{i'}$ (and indeed $P_{i'}$) is faulty. Stop. Otherwise,

Step 3: Each processor applies Theorem 6 to the $u_j^{(K)}$'s it receives from Step 1.

Step 4: Each processor reports, in a distributed manner, whether Theorem 6 is satisfied or not.

Since a reliable broadcast is utilized, each processor receives the same version of a sent message. Each non-faulty processor then reaches the same conclusion concerning the result of the final calculation of the iteration sequence specified by (1). The test in Step 2 precludes $P_{i'}$ changing its value from an erroneous to correct state during the broadcast phase. If $P_{i'}$ were allowed to

change it's value, it could send a value that agreed with Theorem 6 thus making $\Omega_i^D = 0$ erroneously.

If $P_{i'}$ does not change it's value, then Theorem 6 is used to diagnose $\Omega_{i'}$. Note that Theorem 6 does not explicitly specify which member of $\Omega_{i'}$ is faulty. Since the maximum number of faults per fault group is 1, no two processors can cooperate to fool the test of Theorem 6. Furthermore, in the example under consideration, since the minimum cardinality fault group is 3 (equivalent to the row of matrix A with the minimum number of non-zero entries), the non-faulty processors will always outvote the faulty processors. A simple majority vote is taken to determine the status of the fault group. □

Definition 4: *Extended Fault Region* $K_i = \bigcup\{\Omega_j | \Omega_j \cap P_i \neq \emptyset\}$

The extended fault region of a point i' is simply the set of local fault groups of all neighboring points j such that u_j uses $u_{i'}$ in it's calculation.

Let $K_i^D = |\{P_j | P_j \text{ is faulty and } P_j \in K_i\}|$ and let $k_i = |\{j | \Omega_j \in K_i\}|$

Theorem 7: For each processor P_i, P_i is faulty iff $\Omega_j^D = 1$ for all $\Omega_j \in K_i$ and $K_i^D \leq k_i - 2$.

Proof:

Necessity: Assume P_i is non-faulty. If $\Omega_i^D = 0$ then we are done. If not, then there exists some $P_j \in \Omega_i$ and $P_j \in \Omega_j$ such that P_j is faulty. Since at most $k_i - 2$ processors may be faulty and at most one is allowed per fault group (by Lemma 1), and since there are $k_i - 2 - 1$ faulty processors remaining to be distributed over the $k_i - 2$ remaining fault groups, some $\Omega_i^D = 0$.

Sufficiency: Assume that some $\Omega_j^D = 0$. However, by definition of the extended fault group, each $\Omega_j \in K_i$ contains P_j. Thus P_j is non-faulty. Secondly assume that $K_i^D > k_i - 2$. Then it is possible to have $k_i - 2$ faulty processors none of which are in Ω_i and one faulty processor in Ω_i which is not P_i. Thus P_i is non-faulty. □

With the test provided by Theorem 7, we can obtain the error coverage shown in Figure 5.

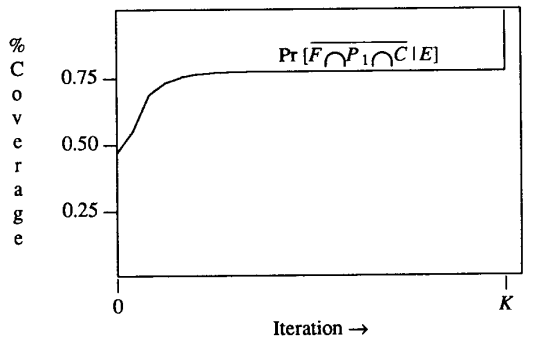

Figure 5. Error Coverage by Solution Step Count. Theorem 7 Applied.

5. COMMENTS, LIMITATIONS, and FUTURE DIRECTIONS

The use of executable assertions is a software engineering approach to providing both software and hardware fault-tolerance. This paper has presented a high-level set of basis metrics which may be applied in conjunction with the software development process to produce executable assertions for the parallel execution environment. These assertions embody both the "liveliness" and "safety" properties of parallel systems. The development methodology is a systematic one in which particular specified features are extracted from the problem to produce the executable assertions. The technique is applied to a large scale parallel problem solution and the effectiveness of the error coverage is modeled.

Employing executable assertions for hardware fault-tolerance, as in the application-oriented fault-tolerance paradigm, is effective not only for the problem of matrix iterative analysis, but for such diverse applications as computer vision [McNi88] and parallel sorting [McMi88]. In the unreliable environment, a reliable message delivery system and a distributed diagnostic basis is necessary [McNi87]. The programming environment for the executable assertions should be able to treat this as a "virtual machine" interface with a low degree of coupling. However, in the example presented, Theorem 7 was necessary to achieve a high degree of confidence in the error coverage. Theorem 7 is an Ad Hoc approach at best to fault diagnosis. Optimally, control of this test should be migrated to the virtual machine rather than the applications level. It is not understood at this point in time how to do this in a general way while still preserving the application oriented view of fault-tolerance. Clearly, this is an area for further study.

REFERENCES

[Ames77] Ames, William, *Numerical Methods for Partial Differential Equations*, Academic Press, New York, 1977.

[Andr79] Andrews, D., "Using Executable Assertions for Testing and Fault Tolerance," *9th Symposium on Fault Tolerant Computing*, 1979, pp. 102-105.

[ChMa84] Chang, J. and Maxemchuk, N., "Reliable Broadcast Protocols," *ACM Transactions on Computer Systems*, Vol 2., No. 3, August 1984, pp. 251-273.

[Hail82] Hailpern, B., *Verifying Concurrent Processes Using Temporal Logic*, Springer-Verlag, Berlin, 1982.

[HuAb87] Hua, K. and Abraham, J., "Design and Evaluation of Executable Assertions for Concurrent Error Detection," *Proceedings of the 11th International COMPSAC*, Tokyo, Japan, October 1987, pp. 324-330.

[McMi88] McMillin, B., *Reliable Parallel Processing - The Application Oriented Paradigm*, Ph.D. Dissertation, Department of Computer Science, Michigan State University, 1988.

[McNi87] McMillin, B. and Ni, L., "Byzantine Fault-Tolerance through Application Oriented Specification," *Proceedings of the 11th International COMPSAC*, Tokyo, Japan, October 1987, pp. 347-353.

[McNi88] McMillin, B. and Ni, L., "A Reliable Parallel Algorithm for Relaxation Labeling," *Proceedings of the 1988 International Conference on Parallel Processing for Computer Vision and Display*, Leeds, U.K., January, 1988, to appear.

[Rand75] Randall, B., "System Structure for Software Fault Tolerance," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 220-232.