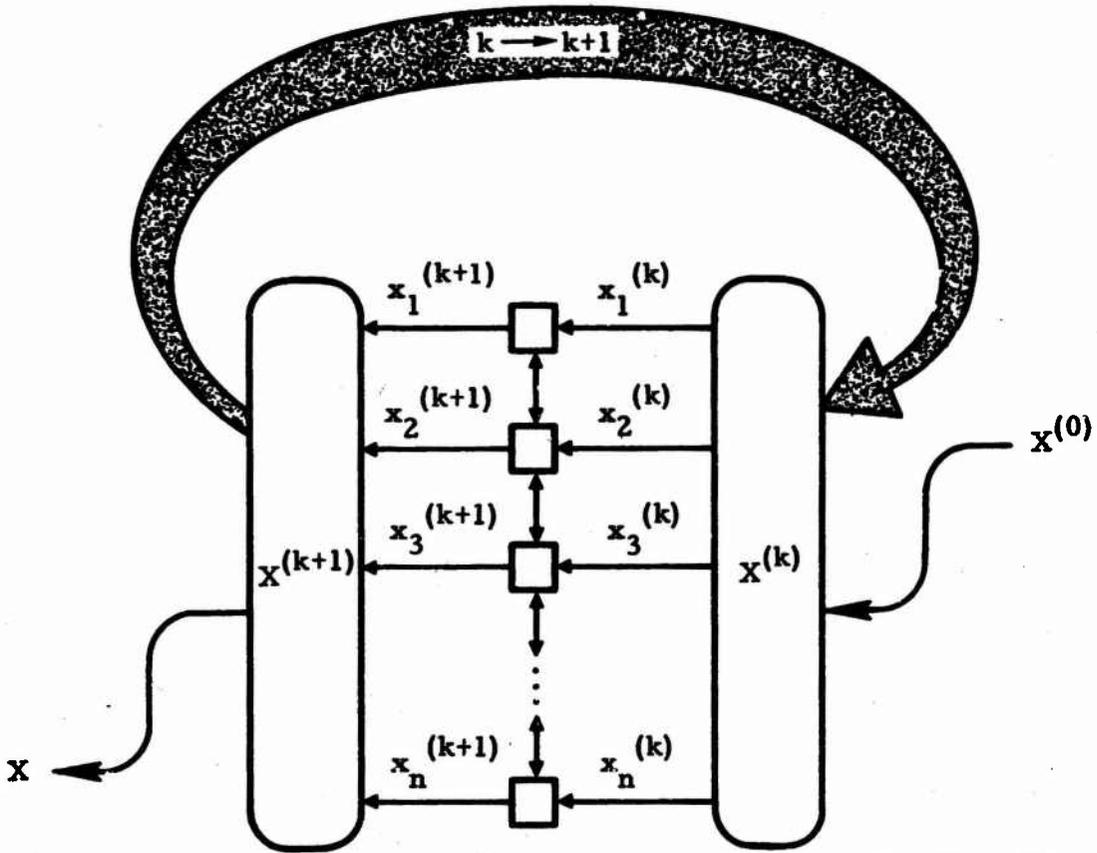


PARALLEL RELAXATION

by P. A. Gilmore

AD 727218



Prepared by

**Goodyear Aerospace Corporation
Akron, Ohio**

for

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH • JULY 1971

DDC
RECEIVED
 JUL 27 1971
RECEIVED
 B

Reproduced by
**NATIONAL TECHNICAL
 INFORMATION SERVICE**
 Springfield, Va. 22151

Approved for public release;
 distribution unlimited.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Goodyear Aerospace Corporation Akron, Ohio 44315		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
		2b. GROUP	
3. REPORT TITLE PARALLEL RELAXATION			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Scientific Final			
5. AUTHOR(S) (First name, middle initial, last name) P. A. Gilmore			
6. REPORT DATE July 1971		7a. TOTAL NO. OF PAGES 33	7b. NO. OF REFS 9
8a. CONTRACT OR GRANT NO. F44620-70-C-0100		9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO. 9749			
c. 61102F		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d. 681304		AFOSR-TR-71-1987	
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES TECH, OTHER		12. SPONSORING MILITARY ACTIVITY Air Force Office of Scientific Research (M) 1400 Wilson Boulevard Arlington, Virginia 22209	
13. ABSTRACT <p>Stationary iterative techniques for solving systems of linear equations are reviewed, and an accelerated form of the Point-Jacobi method (referred to as parallel relaxation) is developed. Associative processors are introduced, and operational characteristics are described. The parallel relaxation method is structured for parallel execution on an associative processor, and estimated parallel and sequential execution times are compared. Timing estimates show an advantage for parallel execution, which increases with the size of the system of equations. The results are extended to arbitrary stationary iterative techniques.</p>			

**GOODYEAR AEROSPACE
CORPORATION**

AKRON, OHIO 44315

PARALLEL RELAXATION

July 1971

P. A. Gilmore

Approved for public release;
distribution unlimited.

FOREWORD

This report is submitted by Goodyear Aerospace Corporation (GAC) to the Air Force Office of Scientific Research (AFOSR) as the final technical report of research performed under Contract F44620-70-C-0100 during the period of 1 July 1970 through 30 June 1971. Project monitor for this program is Lt. Col. Nicholas P. Callas, AFOSR, Arlington, Virginia. This report was issued by the originator as GER-15262. A version of this report has been submitted for publication.

ABSTRACT

Stationary iterative techniques for solving systems of linear equations are reviewed, and an accelerated form of the Point-Jacobi method (referred to as parallel relaxation) is developed. Associative processors are introduced, and operational characteristics are described. The parallel relaxation method is structured for parallel execution on an associative processor, and estimated parallel and sequential execution times are compared. Timing estimates show an advantage for parallel execution, which increases with the size of the system of equations. The results are extended to arbitrary stationary iterative techniques.

TABLE OF CONTENTS

		<u>Page</u>
	FOREWORD	ii
	LIST OF ILLUSTRATIONS	iv
	LIST OF TABLES	iv
<u>Section</u>	<u>Title</u>	
1	INTRODUCTION	1
2	BACKGROUND	2
3	STATIONARY ITERATIVE TECHNIQUES AND PARALLEL PROCESSING	5
	<u>a.</u> Basic Stationary Iteration	5
	<u>b.</u> Accelerated Point-Jacobi	6
	<u>c.</u> Comparison of Convergence Rates	11
	<u>d.</u> Variations of PR Iteration	14
	<u>e.</u> Commentary	17
4	PARRALLEL EXECUTION ON AN ASSOCIATIVE PROCESSOR	17
	<u>a.</u> Introduction	17
	<u>b.</u> AP Structure	18
	<u>c.</u> AP Operations	19
	<u>d.</u> Parallel Relaxation Structure	23
	<u>e.</u> Timing Estimates	30
	<u>f.</u> General Stationary Iteration	31
5	SUMMARY	33
6	REFERENCES	33

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1	Discrete Approximation to the Unit Square	3
2	ω -Dependent Eigenvalues for the PR Iteration Matrix $J = \{(1 - \omega) I + \omega D^{-1} (E + F)\}$ for the Given Matrix A .	11
3	Simplified AP Structure	20
4	AM Structure	21
5	AP Data Storage Scheme	24
6	AP Storage Configuration after Step 1	25
7	Tree-Sum Algorithm	26
8	AP Storage Configuration after Step 2	27
9	AP Storage Configuration after Step 3	28
10	AP Storage Configuration after Step 5	29

LIST OF TABLES

<u>Table</u>	<u>Title</u>	
I	Comparison of Convergence Rates	13
II	Execution Time in Milliseconds for One Sweep by the PR Method.	31

1. INTRODUCTION

In recent years several designs have been proposed for advanced computer organizations known as parallel processors. A distinguishing feature of the parallel processor is its capability to perform many - perhaps thousands - of arithmetic operations simultaneously or in parallel. The implementation of parallel processors will provide the scientific community with vastly increased computational capacity. The realization of the potential offered by the new computer organizations will require the development of new computational algorithms and procedures. The new computational procedures will no doubt derive both from the restructuring of existing methods and the development of new ones, and will certainly occasion considerable theoretical investigation of the properties of the new methods. In this report the subject of parallel execution of matrix-oriented computations is considered. A particular type of matrix computation is considered, the numerical solution of systems of linear equations by stationary iteration, and a particular type of parallel processor, the associative processor (AP).

In Section 2, a brief statement of the rationale behind the application of parallel processing to the solution of linear systems is offered. Section 3 opens with a review of stationary iteration. Subsequently, the theoretical basis for an accelerated form of the Point-Jacobi iteration is developed, the resulting method being called parallel relaxation. In Section 4, the parallel relaxation method is structured for parallel execution on an associative processor and estimates are made of relative execution speeds for parallel and sequential processing. The results of the structuring and timing analyses are extended to arbitrary stationary iterative techniques. In Section 5 the results of this report are summarized.

Throughout this report, a system of linear equations in matrix form is denoted by $AX = B$, where $A = (a_{ij})$ is an $n \times n$ matrix of coefficients; $X = (x_1, x_2, \dots, x_n)^T$ is an $n \times 1$ vector of unknowns; and $B = (b_1, b_2, \dots, b_n)^T$ is an $n \times 1$ vector of real constants. Unless otherwise stated, we shall assume that A is real and symmetric with positive diagonal elements. The stationary iterative techniques with which we are concerned are the Point-Jacobi (PJ) and Gauss-Seidell (GS) methods; the accelerated

successive overrelaxation (SOR) form of GS; and an accelerated form of PJ that we shall call parallel relaxation (PR).

2. BACKGROUND

System of linear equations often derive from discrete approximations to partial differential equations. An example of this is provided by the numerical solution of Laplace's equation over a rectangle, which we shall employ as an example and for elucidation of the rationale behind parallel processing for matrix computations. Let us consider the numerical solution of the Dirichlet problem for Laplace's equation on the unit square. We seek approximations to a function $u(x, y)$ that satisfies

$$\frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} = 0, \quad (1)$$

with $x \in [0, 1] \wedge y \in [0, 1]$ in the interior of the unit square and satisfies a boundary condition

$$u(x, y) = g(x, y). \quad (2)$$

where $g(x, y)$ is a given function defined in the boundary of the square. We select a uniform mesh with spacing $h = 1/3$ and number the interior and boundary mesh points according to the convention implied in Figure 1, where u_i denotes the approximation to $u(x, y)$ at interior point i , and g_j denotes the value of $g(x, y)$ at boundary point j .

If we employ the familiar 5-point stencil for the discrete approximation of Equation 1, we have for each interior mesh point (x, y) of the square the approximation

$$u(x_0, y_0) = \frac{1}{4} \left[u(x_0 + h, y_0) + u(x_0 - h, y_0) + u(x_0, y_0 + h) + u(x_0, y_0 - h) \right]. \quad (3)$$

From this "north, south, east, west" neighbor approximation, there results the system of linear equations

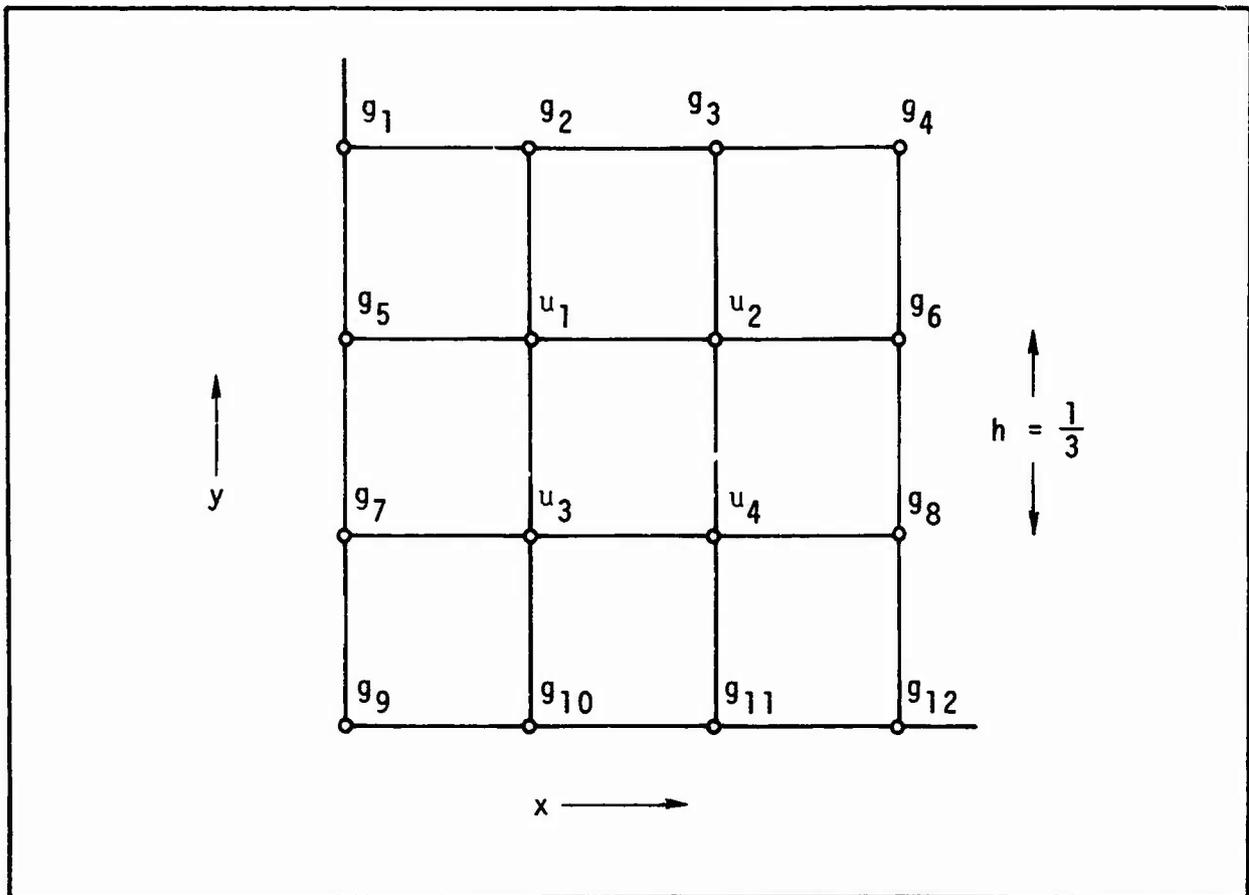


Figure 1 - Discrete Approximation to the Unit Square

$$u_1 = \frac{1}{4}(u_2 + u_3) + \frac{1}{4}(g_2 + g_5)$$

$$u_2 = \frac{1}{4}(u_1 + u_4) + \frac{1}{4}(g_3 + g_6)$$

$$u_3 = \frac{1}{4}(u_1 + u_4) + \frac{1}{4}(g_7 + g_{10})$$

$$u_4 = \frac{1}{4}(u_2 + u_3) + \frac{1}{4}(g_8 + g_{11}) \quad (4)$$

For this simple case, a direct method of solution for Equation 4 would seem the obvious choice, but we are using the example as a vehicle for introducing iterative methods of solution.

From the approximation of Equation 3, we can develop an iterative solution that proceeds as follows. We select initial estimates $\{u_1^{(0)}, u_2^{(0)}, u_3^{(0)}, u_4^{(0)}\}$ of $u(x, y)$ at the interior mesh points and then cyclically improve the estimates using Equation 3. For example: given the initial estimates, we improve the estimate at point 1 by computing

$$u_1^{(1)} = \frac{1}{4} \left(u_2^{(0)} + u_3^{(0)} \right) + \frac{1}{4} (g_2 + g_5). \quad (5)$$

Passing to interior point 2, we have the option of using either the initial estimate of u_1 or the new estimate just computed. If the first option is taken, we compute

$$u_2^{(1)} = \frac{1}{4} \left(u_1^{(0)} + u_4^{(0)} \right) + \frac{1}{4} (g_3 + g_6); \quad (6)$$

if the second option is taken, we compute

$$u_2^{(1)} = \frac{1}{4} \left(u_1^{(1)} + u_4^{(0)} \right) + \frac{1}{4} (g_3 + g_6). \quad (7)$$

Similarly, we can proceed to update points 3 and 4, either using only estimates available at the start of the updating cycle or using new estimates as they become available. If updating of estimates is deferred until the end of each cycle, or sweep over the mesh, we refer to the updating procedure as a "simultaneous displacement" procedure and express the passage from the k^{th} to the $(k + 1)$ st estimate by

$$\begin{aligned} u_1^{(k+1)} &= \frac{1}{4} \left(u_2^{(k)} + u_3^{(k)} \right) + \frac{1}{4} (g_2 + g_5) \\ u_2^{(k+1)} &= \frac{1}{4} \left(u_1^{(k)} + u_4^{(k)} \right) + \frac{1}{4} (g_3 + g_6) \\ u_3^{(k+1)} &= \frac{1}{4} \left(u_1^{(k)} + u_4^{(k)} \right) + \frac{1}{4} (g_7 + g_{10}) \\ u_4^{(k+1)} &= \frac{1}{4} \left(u_2^{(k)} + u_3^{(k)} \right) + \frac{1}{4} (g_8 + g_{11}) \end{aligned} \quad (8)$$

If new estimates are used as soon as these become available, we refer to the updating procedure as a "successive displacement" procedure, and the passage from the k^{th} to the $(k + 1)$ st estimate is given by

$$\begin{aligned}
u_1^{(k+1)} &= \frac{1}{4} \left(u_2^{(k)} + u_3^{(k)} \right) + \frac{1}{4} (g_2 + g_5) \\
u_2^{(k+1)} &= \frac{1}{4} \left(u_1^{(k+1)} + u_4^{(k)} \right) + \frac{1}{4} (g_3 + g_6) \\
u_3^{(k+1)} &= \frac{1}{4} \left(u_1^{(k+1)} + u_4^{(k)} \right) + \frac{1}{4} (g_7 + g_{10}) \\
u_4^{(k+1)} &= \frac{1}{4} \left(u_2^{(k+1)} + u_3^{(k+1)} \right) + \frac{1}{4} (g_8 + g_{11}) . \tag{9}
\end{aligned}$$

The iteration of Equation 9 will, for the example problem at least, give more rapid convergence than Equation 8. If the solution of the discrete problem is to be implemented on a conventional sequential processor, the iteration of Equation 9 is to be preferred. But what if a parallel processor is available and a processing unit can be assigned to each point of the mesh of Figure 1 (equivalently, to each variable of Equation 4), or indeed to each point of a much larger mesh, so that the updating of mesh point values can be carried on in parallel? Does the obvious potential for parallel execution inherent in the simultaneous displacement procedure make it the logical choice for parallel processing? And does the apparently sequential character of the successive displacement procedure render it unattractive for parallel execution? A reasonable answer to these and related questions requires additional analysis.

3. STATIONARY ITERATIVE TECHNIQUES AND PARALLEL PROCESSING

a. Basic Stationary Iteration

In the previous section, we considered the iterative solution of a particular system of linear equations that arose in the numerical solution of Laplace's equation. In this section, we shall examine more general systems

$$AX = B \tag{10}$$

for exhibiting their iterative solution via parallel processing. Our assumptions regarding the $n \times n$ coefficient matrix A remain those of Section 1; namely, that A is real and symmetric with positive diagonal elements. If we write Equation 10 in the equivalent form

$$X = MX + G , \tag{11}$$

we may specify iterative solutions to Equation 10 by selecting $X^{(0)}$, an initial approximation to X , and by iterating

$$X^{(k+1)} = MX^{(k)} + G. \quad (12)$$

The sequence $X^{(0)}$, $X^{(1)}$, $X^{(2)}$, . . . generated by the iteration will, for suitable conditions, converge to a solution of Equation 10. A wealth of literature exists regarding necessary and sufficient conditions for the convergence of Equation 12 (see, for example, References 1, 2, and 3). The matrix M is called the iteration matrix. If it does not vary from iteration to iteration, the iteration is said to be stationary. If we split the coefficient matrix A as

$$A = D - E - F, \quad (13)$$

where D is a diagonal matrix and $E(F)$ is lower (upper) triangular, then two equivalent forms of Equation 10 are given by

$$X = D^{-1} (E + F) X + D^{-1} B \quad (14)$$

and

$$X = (D - E)^{-1} F X + (D - E)^{-1} B \quad (15)$$

From the first form, we obtain the simultaneous displacement PJ iteration

$$X^{(k+1)} = D^{-1} (E + F) X^{(k)} + D^{-1} B; \quad (16)$$

from the second form, we obtain the successive displacement GS iteration

$$X^{(k+1)} = (D - E)^{-1} F X^{(k)} + (D - E)^{-1} B; \quad (17)$$

The iterations of Equations 8 and 9 are particular cases of Equations 16 and 17.

A variety of authors have achieved rather striking success in developing an accelerated version of Gauss-Seidel, known as the successive overrelaxation (SOR) method. We shall subsequently consider parallel execution of Gauss-Seidel and SOR, but first we shall consider the problem of developing an accelerated version of Point-Jacobi.

b. Accelerated Point-Jacobi

A potentially accelerated form of the PJ iteration can be developed as follows: We recall that the PJ iteration given by Equation 16 is just

$$X^{(k+1)} = D^{-1}(E + F)X^{(k)} + D^{-1}B. \quad (18)$$

If we denote by $\tilde{X}^{(k+1)}$ the $(k+1)$ st estimate of the solution vector X given by PJ and then specify $X^{(k+1)}$ as a weighted average of $\tilde{X}^{(k+1)}$ and $X^{(k)}$, we have

$$X^{(k+1)} = \omega \tilde{X}^{(k+1)} + (1 - \omega) X^{(k)}. \quad (19)$$

In matrix form, this would be

$$X^{(k+1)} = \left\{ (1 - \omega)I + \omega D^{-1}(E + F) \right\} X^{(k)} + \omega D^{-1}B. \quad (20)$$

We shall restrict $\omega > 0$ and refer to the iteration of Equation 20 as the parallel relaxation (PR) method. Evidently, if $\omega = 1$, PR reduces to PJ.

The question immediately arises as to what value of ω gives the maximum convergence rate and for what range of ω values is Equation 20 convergent. Let J denote the PJ iteration matrix:

$$J = D^{-1}(E + F), \quad (21)$$

and let P denote the PR iteration matrix:

$$P = \left\{ (1 - \omega)I + \omega J \right\}. \quad (22)$$

We denote the eigenvalues of J by $S(J)$, read "the spectrum of J ," and the eigenvalues of P by $S(P)$. Our assumption that the coefficient matrix A was real and symmetric implies that $S(A) \subset \mathbb{R}$; that is, the eigenvalues of A are contained in \mathbb{R} , the set of real numbers. The further assumption that A has positive diagonal elements implies that $S(J) \subset \mathbb{R}$. We see this by noting that, since the diagonal elements of A are positive, D^{-1} exists as do $D^{1/2}$ and $D^{-1/2}$, all being real. Now we may write $J = D^{-1}(E + F) = I - D^{-1}A$.

Since similarity transformations preserve eigenvalues, we have $S(D^{1/2}JD^{-1/2}) = S(J)$. But $\tilde{J} = D^{1/2}JD^{-1/2} = I - D^{-1/2}AD^{-1/2}$. Obviously, $\tilde{J}^T = \tilde{J}$; hence, \tilde{J} is real and symmetric and $S(\tilde{J}) \subset \mathbb{R}$. But $S(\tilde{J}) = S(J)$; hence, $S(J) \subset \mathbb{R}$.

It follows from Equation 20 that, if $\mu \in S(J)$ and $\lambda \in S(P)$ - that is, μ and λ are, respectively, eigenvalues of J and P - we have the relation

$$\begin{aligned}\lambda &= \omega\mu + (1 - \omega) \\ &= 1 - \omega(1 - \mu),\end{aligned}\tag{23}$$

and since μ is real, so is λ . Since we are restricting $\omega > 0$, it is evident from Equation 23 that, if the PR iteration is to converge, we must restrict $\mu < 1$. Within these restrictions, the optimum value of ω is that for which the spectral radius of P, denoted by $\rho(P)$, is minimized. Let us denote the ordered eigenvalues of J by $S(J) = \{\mu_1 \geq \mu_2 \geq \dots \geq \mu_n\}$ and denote the eigenvalues of P corresponding to μ_i by $\lambda(\omega, \mu_i) = 1 - \omega(1 - \mu_i)$. Then for $\omega = 0$, we have $\lambda(\omega, \mu_i) = 1$ for $i = 1, 2, \dots, n$. For $\omega > 0$, $\mu_i < \mu_j \Rightarrow \lambda(\omega, \mu_i) < \lambda(\omega, \mu_j)$. Hence, $\rho(P)$ is minimal for $\lambda(\omega, \mu_1) = -\lambda(\omega, \mu_n)$; that is for

$$1 - \omega(1 - \mu_1) = -[1 - \omega(1 - \mu_n)],\tag{24}$$

which implies that

$$\omega = \frac{2}{2 - (\mu_1 + \mu_n)}\tag{25}$$

We summarize these results in the following

Theorem

Let A be an $n \times n$ matrix that is real and symmetric with positive diagonal entries and let the PJ iteration matrix for A have eigenvalues bounded from above by 1. Then, the optimum relaxation parameter for PR is given by:

$$\omega_{\text{opt}} = \frac{2}{2 - (\mu_1 + \mu_n)}\tag{26}$$

An example of this is given by the matrix A defined as follows:

$$A = \begin{pmatrix} 4 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 6 \end{pmatrix}.\tag{27}$$

The eigenvalues of the related PJ matrix are $\{0.48835, 0.19870, -0.68705\}$ from which it follows ω_{opt} is given by

$$\begin{aligned}\omega_{\text{opt}} &= \frac{2}{2 - (0.48835 - 0.68705)} \\ &= 0.9096.\end{aligned}\tag{28}$$

The spectral radius of the corresponding PR iteration matrix is 0.5346. In this case, the optimal strategy for PR calls for underrelaxation ($\omega < 1$); whereas, for SOR the optimal strategy is [for property (A) matrices at least] always to overrelax.

Another interesting feature of PR is that not only can it accelerate convergence of the PJ iteration but it also can in some cases establish convergence. It is readily shown that, if the coefficient matrix A is diagonally dominant, convergence of the PJ iteration is assured; diagonal dominance of A is, however, not a necessary condition. For convergence of PJ, a condition that is both necessary and sufficient is that the spectral radius of the iteration matrix J be less than 1.0.

Let us consider a PJ iteration matrix J with eigenvalues $S(J)$ bounded from above by 1.0. Now for any eigenvalue $\mu \in S(J)$ the corresponding eigenvalue λ of the PR iteration matrix P is given by $\lambda(\omega, \mu) = 1 - \omega(1 - \mu)$. If we denote the ordered spectrum of J by $S(J) = \{\mu_1 \geq \mu_2 \geq \dots \geq \mu_n\}$ where $\mu_1 < 1$, then for $\omega \in (0, \infty)$ we have $\lambda(\omega, \mu_n) \leq \lambda(\omega, \mu_1) < 1$ always. To ensure $\lambda(\omega, \mu_n) > -1$, and hence convergence of the PR iteration, we need only restrict ω as follows:

$$0 < \omega < \frac{2}{1 - \mu_n} \quad (29)$$

We see then that for linear systems for which the corresponding PJ matrix J has eigenvalues bounded from above by 1.0, the convergence - destroying effects of negative eigenvalues whose magnitude exceeds 1.0 can be offset by proper selection of ω . As an example of the convergence-establishing properties of the PR iteration, consider the following matrix:

$$A = \begin{pmatrix} 6 & 1 & 2 & 1 & 3 \\ 1 & 5 & 4 & 0 & 1 \\ 2 & 4 & 8 & 1 & 2 \\ 1 & 0 & 1 & 4 & 3 \\ 3 & 1 & 2 & 3 & 8 \end{pmatrix} \quad (30)$$

The matrix A is in no way diagonally dominant, and the eigenvalues of the associated PJ matrix are given by $S(J) = \{0.67596, 0.56931, 0.23783, -0.32066, -1.16243\}$. Evidently, the occurrence of the eigenvalue $\mu = -1.16243$ will cause divergence for the PJ iteration. However, the use of PR with ω in the range

$$0 < \omega < \frac{2}{1 + 1.16243} = 0.92488 \quad (31)$$

will guarantee convergence. The optimal ω for this case is given by

$$\begin{aligned} \omega_{\text{opt}} &= \frac{2}{2 - (0.67596 - 1.16243)} \\ &= 0.80435. \end{aligned} \quad (32)$$

These results are presented in Figure 2.

The coefficient matrices with which we have dealt in the preceding discussion of the PR method have been assumed to be real and symmetric with positive diagonal elements. Very often, however, iterative solutions are sought for linear systems whose coefficient matrices possess Young's property (A). For such coefficient matrices, the eigenvalues of the corresponding PJ matrix J occur either as zero or plus-minus pairs. In such cases, the maximum eigenvalue $\mu_1 \in S(J)$, and the minimum eigenvalue $\mu_n \in S(J)$ would be related as $\mu_1 = -\mu_n$, in which case the optimum ω for PR would be given by

$$\omega_{\text{opt}} = \frac{2}{2 - (\mu_1 + \mu_n)} = 1, \quad (33)$$

and PR offers no advantage over PJ. In such cases, PJ still is attractive for implementation on parallel processors, and there exists the possibility for increasing the rate of convergence by utilizing semi-iterative techniques.

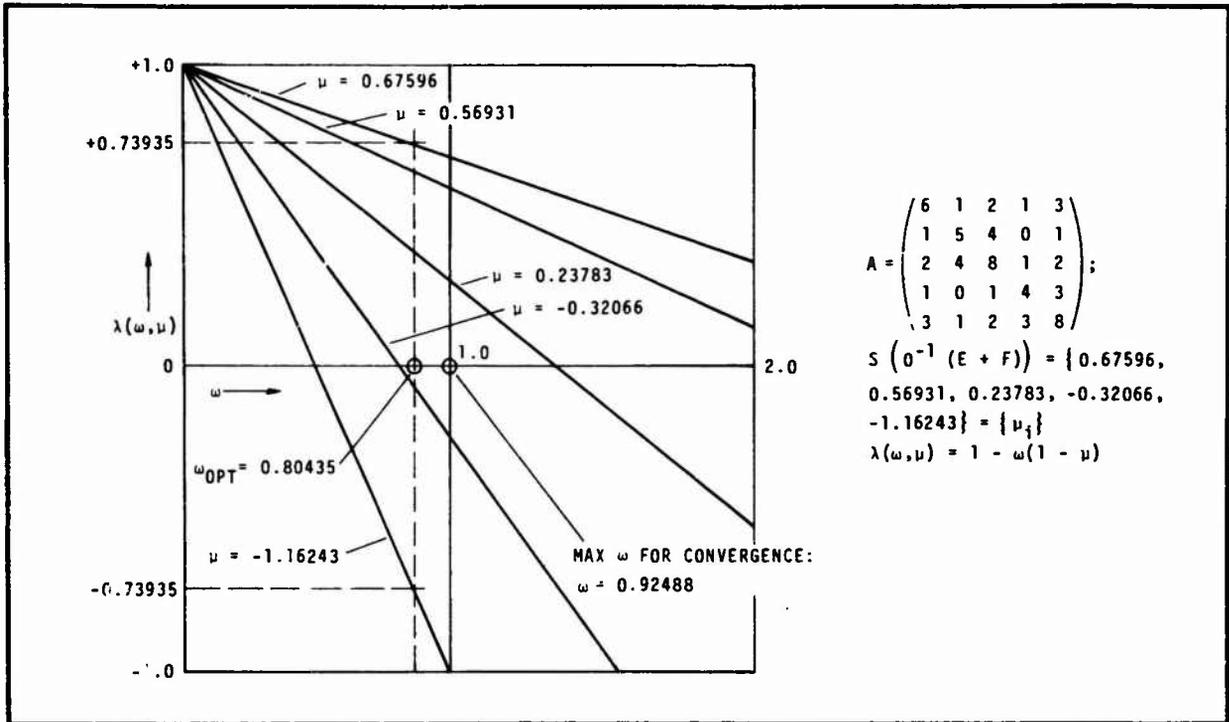


Figure 2 - ω -Dependent Eigenvalues for the PR Iteration Matrix $J = \{(1 - \omega) I + \omega D^{-1} (E + F)\}$ for the Given Matrix A

For PR in general, variations of the basic iteration may prove to be effective. In Item d, below, some possible variations of the PR iteration are outlined. In the following item the question of convergence rates is considered.

c. Comparison of Convergence Rates

In the preceding analysis, we have shown that for a large class of matrices, the use of PR gives better convergence rates than those achievable with PJ, but we have not developed a comparison of convergence rates, which is simple in the sense that we can say the PR convergence rate is k times the PJ convergence rate. In this item we shall exhibit several coefficient matrices for linear systems and compare the corresponding convergence rates for PR and PJ.

If for a linear system $AX = B$, we split the matrix A as $A = D - E - F$ and denote the corresponding PJ and PR iteration matrixes by J and P, respectively, then we have

$$J = D^{-1} (E + F)$$

and

$$P = \{ (1-\omega) I + \omega J \} .$$

For a convergent stationary iteration,

$$X^{(k+1)} = MX^{(k)} + G,$$

we can define the rate of convergence of the iteration by

$$R(M) = - \ln \rho(M),$$

where $\rho(M)$ is the spectral radius of the iteration matrix. We now exhibit several matrices A and compare the convergence rates for the corresponding PJ and PR iterations.

Matrix 1: 3 × 3

$$A = \begin{pmatrix} 4 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 2 \end{pmatrix} ; \omega_{\text{opt}} = 0.9096$$

$$\rho(J) = 0.6870 \Rightarrow R(J) = 0.3754$$

$$\rho(P) = 0.5346 \Rightarrow R(P) = 0.6262$$

Matrix 2: 3 × 3

$$A = \begin{pmatrix} 4 & 2 & 1 \\ 2 & 6 & 2 \\ 1 & 2 & 5 \end{pmatrix} ; \omega_{\text{opt}} = 0.9003$$

$$\rho(J) = 0.6704 \Rightarrow R(J) = 0.3999$$

$$\rho(P) = 0.5038 \Rightarrow R(P) = 0.6856$$

Matrix 3: 5 × 5

$$A = \begin{pmatrix} 8 & 1 & 2 & 1 & 3 \\ 1 & 7 & 4 & 0 & 1 \\ 2 & 4 & 10 & 1 & 2 \\ 1 & 0 & 1 & 6 & 3 \\ 3 & 1 & 2 & 3 & 10 \end{pmatrix}; \omega_{\text{opt}} = 0.8435$$

$$\rho(J) = 0.8785 \Rightarrow R(J) = 0.1295$$

$$\rho(P) = 0.5845 \Rightarrow R(P) = 0.5370$$

Matrix 4: 5 × 5

$$A = \begin{pmatrix} 6 & 1 & 2 & 1 & 3 \\ 1 & 5 & 4 & 0 & 1 \\ 2 & 4 & 10 & 1 & 2 \\ 1 & 0 & 1 & 6 & 3 \\ 3 & 1 & 2 & 3 & 10 \end{pmatrix}; \omega_{\text{opt}} = 0.8325$$

$$\rho(J) = 0.9899 \Rightarrow R(J) = 0.0102$$

$$\rho(P) = 0.6566 \Rightarrow R(P) = 0.4207$$

The results are summarized and compared in Table I.

TABLE I - COMPARISON OF CONVERGENCE RATES

Matrix	ω_{opt}	$\rho(J)$	$\rho(P)$	R(J)	R(P)	$R(P)/R(J)$
1	0.9096	0.6870	0.5346	0.3754	0.6262	1.67
2	0.9003	0.6704	0.5038	0.3999	0.6856	1.71
3	0.8435	0.8785	0.5845	0.1295	0.5370	4.14
4	0.8325	0.9899	0.6566	0.0102	0.4207	41.4

The results shown in Table I demonstrate that significant increases in convergence rate can be achieved by the use of PR. A conjecture can be made that the increases are most attractive where they are most needed; namely, for large matrices whose PJ iterations converge very slowly.

In the following item, some possible variations of the PR iterations are outlined.

d. Variations of PR Iteration

(1) Semi-Iteration

Iterative techniques such as PR can be extended to what are called semi-iterative methods. The extension goes like this. From a sequence of estimates $X^{(0)}, X^{(1)}, X^{(2)}, \dots, X^{(m)}$ a new estimate is formed

$$Y^{(m)} = \sum_{i=0}^m \alpha_i^{(m)} X^{(i)} .$$

The problem is to specify sets of constants $\{\alpha_i^{(m)}\}$ such that the sequence $\{Y^{(m)}\}$ converges rapidly to a solution X . The notion of semi-iteration has been rather extensively studied by a variety of authors (see Reference 1). For a linear system whose coefficient matrices satisfy certain conditions, semi-iterative methods can be constructed with respect to the Point-Jacobi iteration, whose convergence rates compete with SOR. Coupled with parallel processing capability, such methods should be quite attractive computationally. The use of semi-iterative methods in conjunction with optimal PR offers an interesting area of investigation, the results of which should prove to be of importance to parallel processing considerations.

(2) Variable Parameter Iteration

The PR iteration given by Equation 20 uses a constant acceleration parameter ω . If we were to vary the parameter from iteration to iteration, denoting the parameter for the k^{th} iteration by ω_k , the resulting iteration would be given by

$$X^{(k+1)} = \left\{ (1 - \omega_k)I + \omega_k D^{-1} (E + F) \right\} X^{(k)} + \omega_k D^{-1} B \quad (34)$$

If we denote the k^{th} error by $\epsilon^{(k)} = X - X^{(k)}$, then we can readily show that

$$\epsilon^{(k)} = \prod_{i=1}^k \left\{ (1 - \omega_i)I + \omega_i D^{-1} (E + F) \right\} \epsilon^{(0)} .$$

We see that the k^{th} error is the product of the initial error and a polynomial in the PJ iteration matrix. Proper selection of the set $\{\omega_i\}$ may allow improvements over the fixed parameter scheme. Certainly the $\{\omega_i\}$ can be specified so as to allow us to generate the Chebyshev semi-iterative method with respect to PJ. A form of the variable PR method might also occur in a computational context where a constant, optimal ω is desired, but must be successively approximated as the iteration proceeds by a sequence of ω 's which converge to the optimum value. Another context in which computations that are formally the same as variable PR might occur in the numerical solution of initial value problems by marching processes. In some cases, the equations of the marching process are given in a matrix form that is just that of Equation 34 with the exception that the ω_k are replaced by variable time increments Δt_k . Results obtained in the study of variable PR are thus directly applicable to the study of numerical solution of initial value problems.

(3) Partitioning

The form of the PR iteration may also be changed by computational considerations. Very often the linear systems encountered in practice possess $n \times n$ coefficient matrices whose order n is quite large, say 10^3 or greater. In many cases, the matrices are sparse and computer storage requirements for such matrices are not severe as the size of the matrices might indicate. None the less, the main memory capacity of even large computers may be insufficient to contain all the data needed to execute an iterative solution to a large linear system and paging procedures must be employed to transfer data between main memory and some mass memory device such as a disk, drum, or tape. The problem of main memory storage capacity limitations must of course be considered when practical implementation of iterative methods for linear systems on parallel processors is considered.

In addition to practical considerations, there are theoretical questions connected with the use of matrix partitioning, which is often employed in conjunction with paging procedures. In this regard, consider a linear system $AX = B$ and its solution by a stationary iteration.

$$X^{(k+1)} = MX^{(k)} + G.$$

The iteration may be written in partitioned form as follows:

$$\begin{pmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ X_m \end{pmatrix}^{(k+1)} = \begin{pmatrix} M_{11} & M_{12} & \dots & M_{1m} \\ M_{21} & M_{22} & \dots & M_{2m} \\ \cdot & \cdot & \cdot & \cdot \\ M_{m1} & M_{m2} & \dots & M_{mm} \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ X_m \end{pmatrix}^{(k)} + \begin{pmatrix} G_1 \\ G_2 \\ \cdot \\ \cdot \\ G_m \end{pmatrix} \quad (35)$$

Equation 35 can be written equivalently as

$$\begin{aligned} X_1^{(k+1)} &= M_{11}X_1^{(k)} + M_{12}X_2^{(k)} + \dots + M_{1m}X_m^{(k)} + G_1 \\ X_2^{(k+1)} &= M_{21}X_1^{(k)} + M_{22}X_2^{(k)} + \dots + M_{2m}X_m^{(k)} + G_2 \\ \cdot & \\ \cdot & \\ \cdot & \\ X_m^{(k+1)} &= M_{m1}X_1^{(k)} + M_{m2}X_2^{(k)} + \dots + M_{mm}X_m^{(k)} + G_m \end{aligned} \quad (36)$$

Now if the estimates $X_1^{(k+1)}, X_2^{(k+1)}, \dots, X_m^{(k+1)}$ are to be computed sequentially, then a successive displacement updating might be employed and the updating equations (36) would take on the form

$$\begin{aligned}
X_1^{(k+1)} &= M_{11}X_1^{(k)} + M_{12}X_2^{(k)} + \dots + M_{1m}X_m^{(k)} + G_1 \\
X_2^{(k+1)} &= M_{21}X_1^{(k+1)} + M_{22}X_2^{(k)} + \dots + M_{2m}X_m^{(k)} + G_2 \\
&\cdot \qquad \qquad \qquad \cdot \\
&\cdot \qquad \qquad \qquad \cdot \\
&\cdot \qquad \qquad \qquad \cdot \\
X_m^{(k+1)} &= M_{m1}X_1^{(k+1)} + M_{m2}X_2^{(k+1)} + \dots + M_{mm}X_m^{(k)} + G_m
\end{aligned} \tag{37}$$

The updating procedures given by Equations 36 and 37 might well be employed in a situation where a computer's main memory capacity is insufficient to contain the data required for the full matrix multiplication $MX^{(k)}$, but is sufficient to contain all the data required for a multiplication such as $M_{11}X_1^{(k)}$. In such cases, data required for multiplications $M_{ij}X_j^{(k)}$ can be paged sequentially into main memory from mass memory as partial sums are accumulated to form the result $X_i^{(k+1)}$. Just what the effect of employing a successive displacement procedure such as Equation 37 would be on the convergence properties of the iteration is, in general, not known. Investigation of the properties of an iteration such as Equation 37 would be of both theoretical and practical interest.

e. Commentary

In Section 3, we have considered the question of solving linear systems by stationary iterative techniques and have developed an accelerated version of the PJ method, which we call parallel relaxation. In Section 4, we shall consider the implementation of PR on an associative processor and subsequently show that our results apply to the implementation of stationary iterations in general.

4. PARRALLEL EXECUTION ON AN ASSOCIATIVE PROCESSOR

a. Introduction

The simultaneous displacement updating employed in the PR method is formally well suited to parallel processing. The practical question arises as to whether existing or near-term parallel processors are available for implementation of PR and, if so, what will be the parallel execution time and how does it compare with the execution time for sequential processing?

Over the last several years, a number of parallel processor designs have been proposed (see References 4 through 8). Most of these designs are of theoretical interest only. At this writing, only two - the Illiac IV and the associative processor (AP) - can be considered both practical and near term. The Illiac IV, which is currently under development by Burroughs Corporation and the University of Illinois, is now nearing completion. A number of companies are working on AP's, but so far the only one scheduled for delivery and actual operation is the AP developed by Goodyear Aerospace. Called STARAN 4, the Goodyear Aerospace AP is scheduled for delivery to the FAA in May 1971. It will be used by the FAA in an air traffic control system at the Knoxville, Tenn., airport.

The Goodyear Aerospace AP has been selected as the parallel processor on which to study the implementation of PR for parallel execution. Since the structure and operation of the AP are not widely known, Items b and c are devoted to a general description of the AP and its operation.

b. AP Structure

The Goodyear Aerospace-developed associative processor is a stored program digital computing system capable of operating on many data items simultaneously; both logical and arithmetic operations are available. The principal components of an AP system are as follows:

1. An associative memory array (AM) in which data are stored on which the AP operates. Typically, the AM may consist of 4096 words, each with 256 bits.
2. A response store configuration for each word of the AM, which provides arithmetic capability, read/write capability, and indication of logical operation results.
3. An (optional) funnel memory of as many words as the AM, each word with 32 to 128 bits, depending on user need. The funnel memory provides the AP system with both high-speed temporary storage and high-speed I/O to external devices. Data transfer between the AM and funnel memory is on a serial-by-bit, parallel-by-word basis; data transfer between the funnel memory and

external devices is on a parallel-by bit, serial-by-word basis.

4. A data/instruction memory in which are stored the AP program (that is, the list of instructions executed by the AP) and data items required by (or generated by) the AP but not maintained in the AM.
5. A control unit that directs the AP to execute the instructions specified by the AP program; this control unit is similar to control units found in conventional computers. Communication channels are provided between the data/instruction memory and the sequential control unit and between both of these units and external devices.

One other unit of the AP that must be mentioned is the comparand register (CR). The CR may contain as many bits as an AM word and is used both to transmit data into the AM in a parallel-by-bit, serial-by-word basis and to specify masking conventions for AP operations.

A simplified representation of the AP is given in Figure 3.

c. AP Operations

We are principally concerned with the parallel execution of arithmetic operations in the AP and, to a lesser extent, with the transfer of data within the AP. An understanding of the AP's parallel arithmetic capability can be facilitated by considering Figure 4, which depicts the word/field structure of a hypothetical 10-word, 20-bit AM (refer to Reference 9 for a full discussion of the AP's logical and arithmetic capability).

In Figure 4, each of the 20 bit words has been arbitrarily divided into two 5-bit fields and one 10-bit field. Other field assignments could have been made, and they need not be the same for all words. Field specifications are made by the programmer in accordance with computational and storage requirements at a given stage of the program; the specification is logical, not physical, and can be changed for any or all words at any point in the program by the programmer.

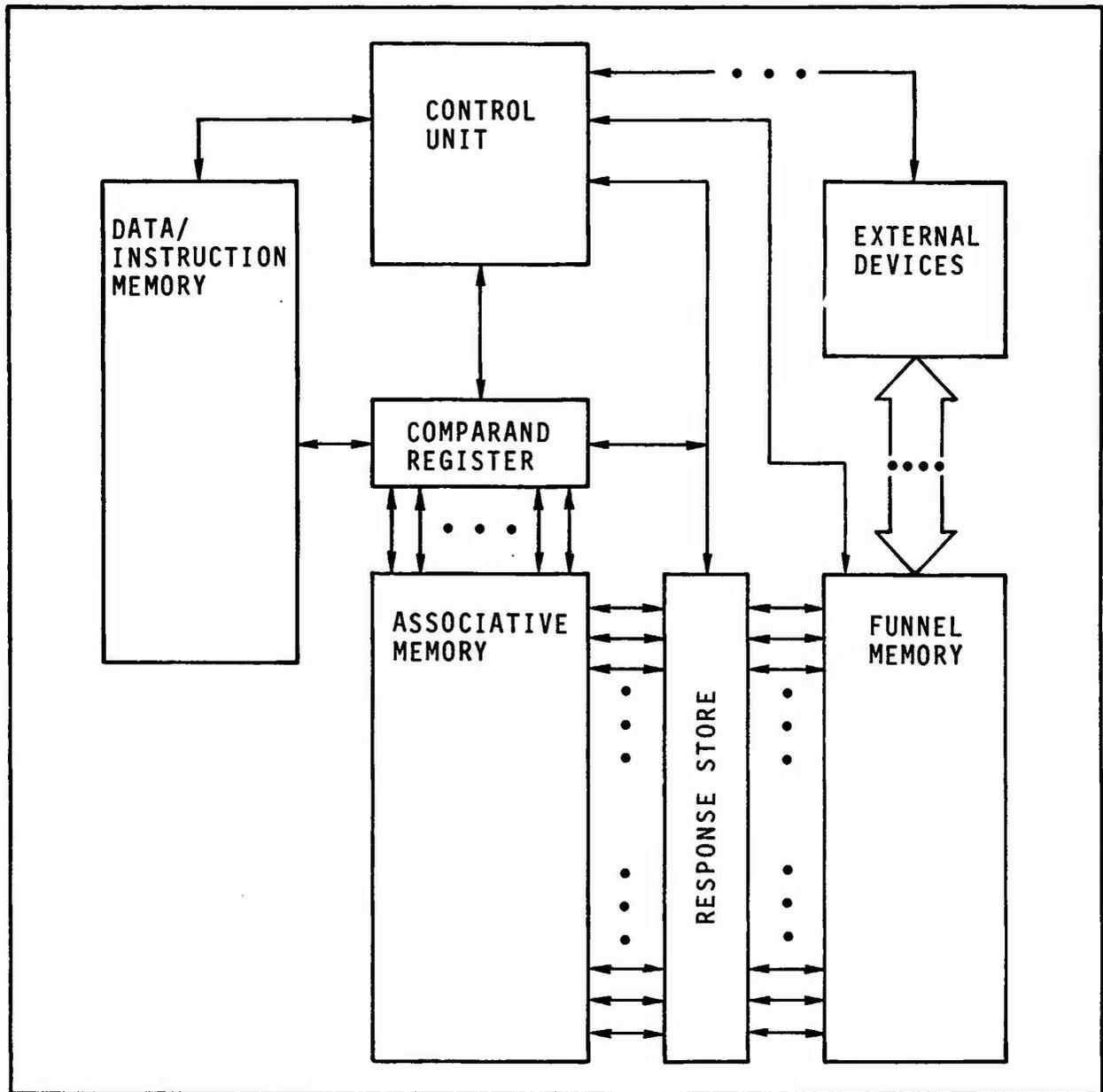


Figure 3 - Simplified AP Structure

If, for $i = 1, 2, \dots, 10$, we denote word i by w_i ; the contents of field 1 of w_i by $(F1_i)$; the contents of field 2 by $(F2_i)$; and the contents of field 3 by $(F3_i)$, then (at least) the following computations can be done in parallel (that is, simultaneously by word, sequential by bit). Such parallel computations are often called vector computations, since they involve like operations on corresponding elements of two vectors of operands.

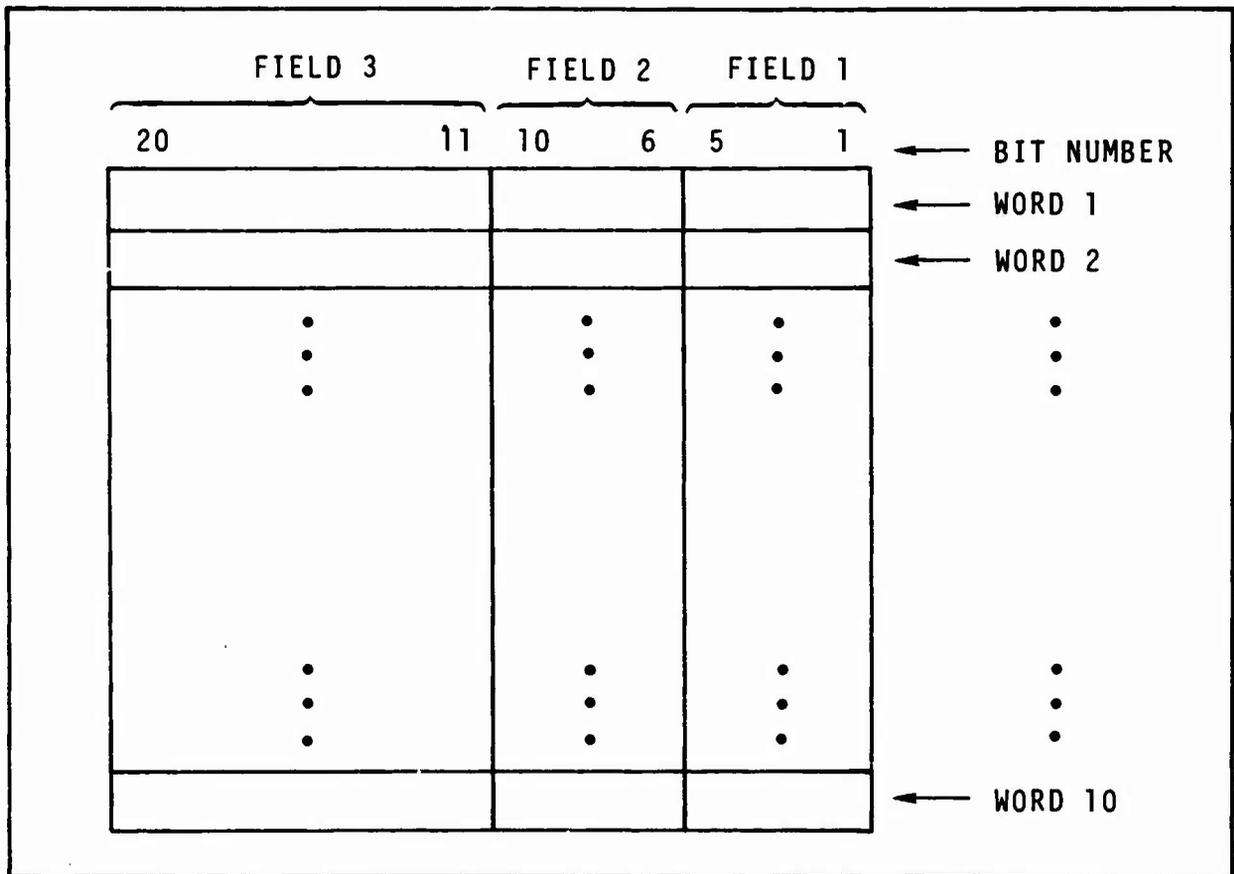


Figure 4 - AM Structure

$$\begin{aligned}
 & (F1_i) \oplus (F2_i) \\
 & \text{or} \quad i = 1, 2, \dots, 10 \wedge \oplus \in \{+, -, * \div\} \\
 & (F2_i) \oplus (F1_i)
 \end{aligned}$$

The field into which the results of the operations are stored is specified by the programmer. For example, the results of the \pm operations could be stored in either field 1, field 2, or field 3. We denote this, for example, by:

$$\begin{aligned}
 & (F1_i) \pm (F2_i) \longrightarrow F1_i \quad i = 1, 2, \dots, 10 \\
 & \text{or} \\
 & (F1_i) \pm (F2_i) \longrightarrow F2_i \quad i = 1, 2, \dots, 10 \\
 & \text{or} \\
 & (F1_i) \pm (F2_i) \longrightarrow F3_i \quad i = 1, 2, \dots, 10
 \end{aligned}$$

In the first two specifications, the original values $(F1_i)$ or $(F2_i)$, respectively, would be destroyed; in the third specification, $(F1_i)$ and $(F2_i)$ would be unaltered.

In $*$ or \div operations, a double-length product or quotient will be available. To save the double-length result, we would be restricted to placing the result in the double-length field, $F3$. For example,

$$(F1_i) * (F2_i) \longrightarrow F3_i \quad i = 1, 2, \dots, 10$$

The original values $(F1_i)$, $(F2_i)$ would be unaltered.

Operations such as those described above are referred to as within-word arithmetic operations. We also have available register-to-word operations and between-word operations.

In register-to-word operations, the contents of a specified field of the comparand register, denoted by (CR) , are used as an operand. A typical register-to-word operation would be:

$$(CR) * (F1_i) \longrightarrow F3_i \quad i = 1, 2, \dots, 10$$

or

$$(CR) \pm (F2_i) \longrightarrow F2_i \quad i = 1, 2, \dots, 10$$

In between-word operations, the operand pairs derive from different words. For example, in the operation

$$(F1_i) * (F1_{i+2}) \longrightarrow F3_i \quad i = 1, 2, \dots, 8,$$

field 1 of word 1 is multiplied by field 1 of word 3, and the result is placed in field 3 of word 1; field 1 of word 2 is multiplied by field 1 of word 4, and the result is placed in field 3 of word 2 ...; field 1 of word 8 is multiplied by field 1 of word 10, and the result is placed in field 3 of word 8. Likewise, we could specify an operation such as

$$(F1_i) + (F2_{i+1}) \longrightarrow F1_i \quad i = 1, 2, \dots, 9.$$

We note that, for between-word operations, the distance between words from which operand pairs are derived is constant; that is, with each word i , we associate a word $i \pm \Delta$.

Such between-word operations are executed in parallel but are more time consuming than within-word or register-to-word operations. The increase in time is proportional to the distance Δ .

In the preceding examples, operand pairs were derived from either AM word/field locations or the comparand register, and results were stored in AM word/field locations. For AP systems incorporating a funnel memory, one element of each operand pair can be derived from the funnel memory, and results can be stored in the funnel memory, with operations taking place, as before, in parallel. Simple data transfer operations between the AM and the funnel memory proceed in a word-parallel, bit-serial fashion.

The bit-serial nature of AP operations results in long execution times if computation is considered on a per-word basis. The source of computational advantages for an AP lies in the AP's ability to do many, indeed thousands, of operations in a word-parallel fashion and thus give, for properly structured computations, effective per-word execution times that are very attractive.

d. Parallel Relaxation Structure

For a linear system $AX = B$, the PR iteration is given by

$$X^{(k+1)} = \left\{ (1 - \omega)I + \omega D^{-1} (E + F) \right\} X^{(k)} + \omega D^{-1} B. \quad (38)$$

We shall let the $n \times n$ PR iteration matrix be denoted by

$$P = (p_{ij}) = \left\{ (1 - \omega)I + \omega D^{-1} (E + F) \right\}, \quad (39)$$

and we shall let the vector $\omega D^{-1} B$ be denoted by $G = (g_i)$. The elements of $P = (p_{ij})$ and $G = (g_i)$ do not change from iteration. The $k+1$ st estimates of $\{x_i\}$ are computed as

$$x_i^{(k+1)} = \sum_{j=1}^n p_{ij} x_j^{(k)} + g_i, \quad i = 1, 2, \dots, n. \quad (40)$$

The structure of the computations in Equation 40 suggests storing data in an AP as shown in Figure 5, where we consider a 5×5 example.

		$x^{(k)}$			$x^{(k+1)}$		
		FIELDS					
1	2	3	4	5	6	7	WORD
g ₁	P11	x ₁					1
	P12	x ₂					2
	P13	x ₃					3
	P14	x ₄					4
	P15	x ₅					5
g ₂	P21	x ₁					6
	P22	x ₂					7
	P23	x ₃					8
	P24	x ₄					9
	P25	x ₅					10
g ₃	P31	x ₁					11
	P32	x ₂					12
	P33	x ₃					13
	P34	x ₄					14
	P35	x ₅					15
g ₄	P41	x ₁					16
	P42	x ₂					17
	P43	x ₃					18
	P44	x ₄					19
	P45	x ₅					20
g ₅	P51	x ₁					21
	P52	x ₂					22
	P53	x ₃					23
	P54	x ₄					24
	P55	x ₅					25

Figure 5 - AP Data Storage Scheme

The storage scheme employed involves redundant storage of current estimates of the variables x_1, x_2, \dots, x_n . The redundancy is employed to gain computational speed; this will become clear in the following discussion.

We shall describe the operations required for one iteration in a step-by-step fashion with reference to corresponding figures depicting the contents of the AP memory.

For each word (1 through 25) in step 1, we multiply the contents of field 2 by the contents of field 3 and store the double-length product in field 4. The resulting state of the AP memory is given in Figure 6.

Step 2 actually is a subroutine type of operation in which the double precision

sums $\sum_{j=1}^n p_{ij} x_j$ are formed in a treed operation that injects parallel

		$x^{(k)}$			$x^{(k+1)}$		
		FIELDS					
1	2	3	4	5	6	7	WORD
9 ₁	P ₁₁	x ₁	P ₁₁ x ₁				1
	P ₁₂	x ₂	P ₁₂ x ₂				2
	P ₁₃	x ₃	P ₁₃ x ₃				3
	P ₁₄	x ₄	P ₁₄ x ₄				4
	P ₁₅	x ₅	P ₁₅ x ₅				5
9 ₂	P ₂₁	x ₁	P ₂₁ x ₁				6
	P ₂₂	x ₂	P ₂₂ x ₂				7
	P ₂₃	x ₃	P ₂₃ x ₃				8
	P ₂₄	x ₄	P ₂₄ x ₄				9
	P ₂₅	x ₅	P ₂₅ x ₅				10
9 ₃	P ₃₁	x ₁	P ₃₁ x ₁				11
	P ₃₂	x ₂	P ₃₂ x ₂				12
	P ₃₃	x ₃	P ₃₃ x ₃				13
	P ₃₄	x ₄	P ₃₄ x ₄				14
	P ₃₅	x ₅	P ₃₅ x ₅				15
9 ₄	P ₄₁	x ₁	P ₄₁ x ₁				16
	P ₄₂	x ₂	P ₄₂ x ₂				17
	P ₄₃	x ₃	P ₄₃ x ₃				18
	P ₄₄	x ₄	P ₄₄ x ₄				19
	P ₄₅	x ₅	P ₄₅ x ₅				20
9 ₅	P ₅₁	x ₁	P ₅₁ x ₁				21
	P ₅₂	x ₂	P ₅₂ x ₂				22
	P ₅₃	x ₃	P ₅₃ x ₃				23
	P ₅₄	x ₄	P ₅₄ x ₄				24
	P ₅₅	x ₅	P ₅₅ x ₅				25

Figure 6 - AP Storage Configuration after Step 1

computation into the basic summing process. The summing subroutine is described in Figure 7 and is executed in parallel by the AP for each consecutive set of 5 (for the example, n in general) words. The result of this sum-

ming operation is that in field 4 word 1 contains $\sum_{j=1}^n p_{1j}x_j$; word 6 contains

$\sum_{j=1}^n p_{2j}x_j$; ..., word 21 contains $\sum_{j=1}^n p_{5j}x_j$. In general, for an $n \times n$ sys-

tem, these sums would be accumulated in field 4 of words 1, n + 1, 2n + 1, ..., (n - 1)n + 1. The resulting state of the AP memory is given in Figure 8.

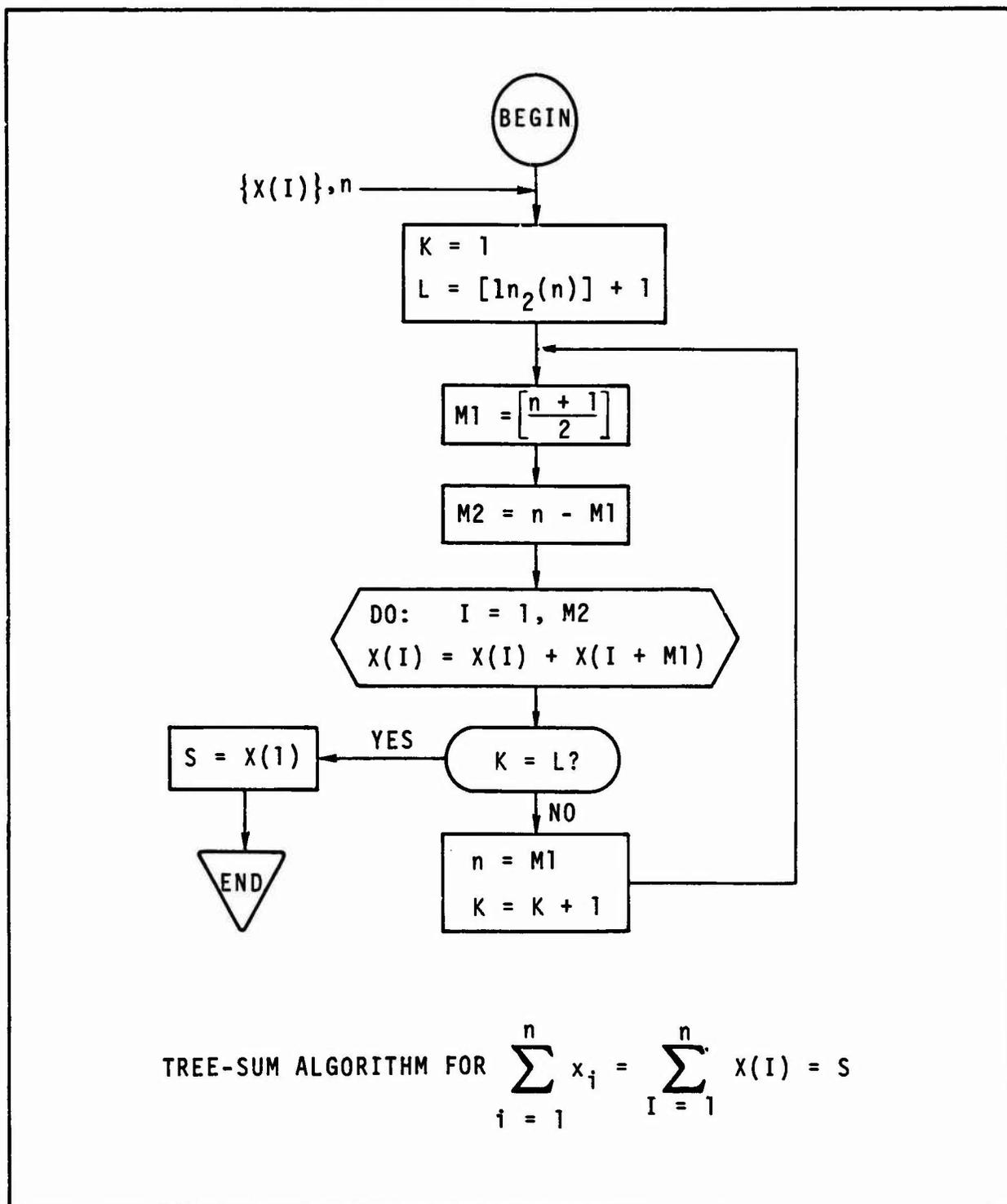


Figure 7 - Tree-Sum Algorithm

		$x^{(k)}$			$x^{(k+1)}$		
		FIELDS					
1	2	3	4	5	6	7	WORD
9 ₁	p_{11}	x_1	$p_{11}x_1$	$\sum p_{1j}x_j$			1
	p_{12}	x_2	$p_{12}x_2$				2
	p_{13}	x_3	$p_{13}x_3$				3
	p_{14}	x_4	$p_{14}x_4$				4
	p_{15}	x_5	$p_{15}x_5$				5
9 ₂	p_{21}	x_1	$p_{21}x_1$	$\sum p_{2j}x_j$			6
	p_{22}	x_2	$p_{22}x_2$				7
	p_{23}	x_3	$p_{23}x_3$				8
	p_{24}	x_4	$p_{24}x_4$				9
	p_{25}	x_5	$p_{25}x_5$				10
9 ₃	p_{31}	x_1	$p_{31}x_1$	$\sum p_{3j}x_j$			11
	p_{32}	x_2	$p_{32}x_2$				12
	p_{33}	x_3	$p_{33}x_3$				13
	p_{34}	x_4	$p_{34}x_4$				14
	p_{35}	x_5	$p_{35}x_5$				15
9 ₄	p_{41}	x_1	$p_{41}x_1$	$\sum p_{4j}x_j$			16
	p_{42}	x_2	$p_{42}x_2$				17
	p_{43}	x_3	$p_{43}x_3$				18
	p_{44}	x_4	$p_{44}x_4$				19
	p_{45}	x_5	$p_{45}x_5$				20
9 ₅	p_{51}	x_1	$p_{51}x_1$	$\sum p_{5j}x_j$			21
	p_{52}	x_2	$p_{52}x_2$				22
	p_{53}	x_3	$p_{53}x_3$				23
	p_{54}	x_4	$p_{54}x_4$				24
	p_{55}	x_5	$p_{55}x_5$				25

Figure 8 - AP Storage Configuration after Step 2

For words 1, 6, 11, 16, and 21 in step 3, field 1 is added to the upper half of field 5, and the sum is placed in field 6; field 6 of words 1, 6, 11, 16, and 21, respectively, contains now the $k + 1$ st estimates of $x_1, x_2, x_3, x_4,$ and x_5 . The k th estimates are retained redundantly in field 3. The resulting state of the AP memory is given in Figure 9.

In step 4, we prepare for convergence testing. For words 1, 6, 11, 16, and 21, field 3 is subtracted from field 6, and the difference is placed in field 7. In words 1, 6, 11, 16, and 21, respectively, field 7 now contains the differences $x_1^{(k+1)} - x_1^{(k)}, x_2^{(k+1)} - x_2^{(k)}, \dots, x_5^{(k+1)} - x_5^{(k)}$; we next execute step 5, which forms absolute values.

For words 1, 6, 11, 16, and 21 in step 5, we set field 7 equal to the absolute value of the previous contents. The AP memory state resulting from steps 4 and 5 is given in Figure 10.

		$x^{(k)}$			$x^{(k+1)}$		
		FIELDS					WORD
1	2	3	4	5	6	7	
g_1	p_{11} p_{12} p_{13} p_{14} p_{15}	x_1 x_2 x_3 x_4 x_5	$p_{11}x_1$ $p_{12}x_2$ $p_{13}x_3$ $p_{14}x_4$ $p_{15}x_5$	$\sum p_{1j}x_j$	$\Sigma + g_1$		1 2 3 4 5
g_2	p_{21} p_{22} p_{23} p_{24} p_{25}	x_1 x_2 x_3 x_4 x_5	$p_{21}x_1$ $p_{22}x_2$ $p_{23}x_3$ $p_{24}x_4$ $p_{25}x_5$	$\sum p_{2j}x_j$	$\Sigma + g_2$		6 7 8 9 10
g_3	p_{31} p_{32} p_{33} p_{34} p_{35}	x_1 x_2 x_3 x_4 x_5	$p_{31}x_1$ $p_{32}x_2$ $p_{33}x_3$ $p_{34}x_4$ $p_{35}x_5$	$\sum p_{3j}x_j$	$\Sigma + g_3$		11 12 13 14 15
g_4	p_{41} p_{42} p_{43} p_{44} p_{45}	x_1 x_2 x_3 x_4 x_5	$p_{41}x_1$ $p_{42}x_2$ $p_{43}x_3$ $p_{44}x_4$ $p_{45}x_5$	$\sum p_{4j}x_j$	$\Sigma + g_4$		16 17 18 19 20
g_5	p_{51} p_{52} p_{53} p_{54} p_{55}	x_1 x_2 x_3 x_4 x_5	$p_{51}x_1$ $p_{52}x_2$ $p_{53}x_3$ $p_{54}x_4$ $p_{55}x_5$	$\sum p_{5j}x_j$	$\Sigma + g_5$		21 22 23 24 25

Figure 9 - AP Storage Configuration after Step 3

Step 6, like step 2, actually is a subroutine type of operation. We actually do the convergence testing by doing a less-than comparand search in the AP. Effectively, we ask whether for all variables x_1 , x_2 , x_3 , x_4 , and x_5 the magnitude of the difference of the k th and $k + 1$ st estimates is less than a specified tolerance. If so, we say convergence has been achieved, and the latest estimates would be printed out. If not, more iteration would be required, and we would pass to step 7.

In step 7, the new estimates for x_1 , x_2 , x_3 , x_4 , and x_5 are sequentially read out of field 6 of words 1, 6, 11, 16, and 21. Following each read, the x_i value is redundantly written into its proper position in field 3 by a word parallel/bit parallel write.

In step 8, iteration continues; we begin again at step 1.

		$x^{(k)}$			FIELDS		$x^{(k+1)}$		
1	2	3	4	5	6	7		WORD	
g_1	p_{11}	x_1	$p_{11}x_1$	$\sum p_{1j}x_j$	$\sum^+ g_1$	$ x_1^{(k+1)} - x_1^{(k)} $		1	
	p_{12}	x_2	$p_{12}x_2$				2		
	p_{13}	x_3	$p_{13}x_3$				3		
	p_{14}	x_4	$p_{14}x_4$				4		
	p_{15}	x_5	$p_{15}x_5$				5		
g_2	p_{21}	x_1	$p_{21}x_1$	$\sum p_{2j}x_j$	$\sum^+ g_2$	$ x_2^{(k+1)} - x_2^{(k)} $		6	
	p_{22}	x_2	$p_{22}x_2$				7		
	p_{23}	x_3	$p_{23}x_3$				8		
	p_{24}	x_4	$p_{24}x_4$				9		
	p_{25}	x_5	$p_{25}x_5$				10		
g_3	p_{31}	x_1	$p_{31}x_1$	$\sum p_{3j}x_j$	$\sum^+ g_3$	$ x_3^{(k+1)} - x_3^{(k)} $		11	
	p_{32}	x_2	$p_{32}x_2$				12		
	p_{33}	x_3	$p_{33}x_3$				13		
	p_{34}	x_4	$p_{34}x_4$				14		
	p_{35}	x_5	$p_{35}x_5$				15		
g_4	p_{41}	x_1	$p_{41}x_1$	$\sum p_{4j}x_j$	$\sum^+ g_4$	$ x_4^{(k+1)} - x_4^{(k)} $		16	
	p_{42}	x_2	$p_{42}x_2$				17		
	p_{43}	x_3	$p_{43}x_3$				18		
	p_{44}	x_4	$p_{44}x_4$				19		
	p_{45}	x_5	$p_{45}x_5$				20		
g_5	p_{51}	x_1	$p_{51}x_1$	$\sum p_{5j}x_j$	$\sum^+ g_5$	$ x_5^{(k+1)} - x_5^{(k)} $		21	
	p_{52}	x_2	$p_{52}x_2$				22		
	p_{53}	x_3	$p_{53}x_3$				23		
	p_{54}	x_4	$p_{54}x_4$				24		
	p_{55}	x_5	$p_{55}x_5$				25		

Figure 10 - AP Storage Configuration after Step 5

More economical use of storage could be made in storing intermediate results. The storage scheme shown was selected for elucidation of AP operations, not optimal use of storage.

The evident source of computational advantage for the AP lies in its parallel arithmetic capability. In a stationary iteration $X^{(k+1)} = MX^{(k)} + G$, not only can each element in the matrix product $MX^{(k)}$ be computed in parallel, but the scalar products required for each element can be computed in parallel and the subsequent summing process treed. Within AP capacity, only the treed summing process is explicitly dependent timewise on n (the system size), and the requisite number of computational levels increases with $\ln_2(n)$. In contrast to this near-independence of n for parallel execution in an AP, sequential methods of execution will require an execution time varying with n^2 since each element of the product $MX^{(k)}$ will require up to n multiplies

and there are n such elements. This basic advantage enjoyed by the AP will be evident in the timing estimates described in Item e.

e. Timing Estimates

In this item, we present computer-dependent timing estimates for solving an $n \times n$ system of linear equations $AX = B$ by the PR iterative technique. Each iteration will generate a new estimate for each variable x_i , $i = 1, 2, \dots, n$ where we have $X = (x_1, x_2, \dots, x_n)^T$. We shall refer to one complete updating of the variables as a sweep over the set $\{x_i\}$. Depending on the computer organization selected, the operations required in a sweep will be executed either sequentially or in a parallel fashion. The number of sweeps required for convergence will be problem dependent. The computer organizations we consider are the following: (1) parallel processor, Goodyear Aerospace associative processor; and (2) sequential processors, computer 1 (C1) and computer 2 (C2). We employ C1 and C2 as pseudonyms for two commercially available computers of modern design and wide usage that have floating-point multiply times of $21.5 \mu\text{sec}$ and $4.5 \mu\text{sec}$, respectively, for 24-bit mantissas.

In Item f, we shall see that results given here are applicable to parallel execution of stationary iterative techniques in general, whether simultaneous or successive displacement updating procedures are employed.

In Table II, we summarize the execution times required for one sweep employing the PR technique. Total problem execution time would increase with the number of sweeps or iterations required and with required I/O and housekeeping operations. The times given are meant only as estimates for comparing parallel versus sequential execution and for exhibiting the increasing advantage of parallel execution as system size increases. As pointed out in Item d, the reason computational advantage accrues to parallel execution as n increases is that sequential execution time increases with n^2 while the parallel execution time increases with $\ln_2(n)$.

The AP times in Table II are given for 20 and 30 bits. These times are for fixed-point computation (double precision inner product accumulation is employed). Existing models of the AP do not incorporate hard-wired floating-point operations. Floating point is available via software and, if employed,

TABLE II - EXECUTION TIME IN MILLISECONDS FOR
ONE SWEEP BY THE PR METHOD

System size	Time per sweep (iteration), milliseconds			
	C1*	C2 ⁺	AP, 20 bits	AP, 30 bits
5 × 5	0.98	0.27	0.54	0.94
25 × 25	21.62	5.60	0.95	1.47
50 × 50	85.00	21.60	1.38	2.02

*Computer 1 (C1): 21.5 μ sec multiply.

⁺Computer 2 (C2): 4.5 μ sec multiply.

overall execution times will typically increase by a factor of approximately 1.4.

f. General Stationary Iteration

We have examined the structuring of the PR method for parallel execution on the AP and have developed timing estimates for one sweep or iteration. For comparison purposes, we also developed estimates of execution time for a PR sweep using sequential processors. From this comparison, it was seen that computational advantage rested with the AP and that the advantage increased with system size.

One might object to the timing comparison by noting that, for sequential execution, GS and not PR usually would be chosen, and GS would probably give a better convergence rate with little penalty in terms of execution time per sweep. It also could be pointed out that GS often can be accelerated by employing SOR.

In answering such an objection, we might first observe that, in general, very little can be said about the relative convergence rates of PJ or GS. In fact, convergence of one method does not imply convergence of the other. We note further that the results of Item b of Section 3 allow the acceleration of PJ via PR in cases where theory for SOR has not yet been developed. But such

answers do not get at the heart of the matter. The answer we must make is that the successive displacement GS or SOR methods can be executed in parallel.

The amenability of GS or SOR to parallel execution is, I believe, sometimes obscured by the use of the Laplace equation example we employed earlier. In such examples, we cyclically update interior mesh points, using - for successive displacements - new pointwise estimates as they become available. This sequential formulation of the computational procedure tends to convey the impression that new estimates must be computed sequentially if successive displacement updating is employed, the impression probably being strengthened by writing the updating equations in a form such as Equation 9. But although under sequential execution the computation of each new estimate in turn uses the most recently computed estimates, the chain of computations ultimately traces back to estimates available at the beginning of the sweep. The specification of this chain for each point or variable will allow the computations to proceed in parallel. In fact, we have already specified the chain of computations in the matrix formulation of GS, given by Equation 17; namely,

$$X^{(k+1)} = (D - E)^{-1} FX^{(k)} + (D - E)^{-1} B.$$

Evidently, GS also can be executed in parallel if we make the initial investment of computing $(D - E)^{-1} F$. In fact, any stationary iteration, SOR included, which we write as

$$X^{(k+1)} = MX^{(k)} + G$$

can be executed in parallel on an AP using the same computational procedure developed for PR. Parallel computation will require that the iteration matrix M be computed and stored in the AP, but once available, the AP execution time is independent of the analytical complexity of M . The question of how best to compute a particular M , either by parallel or sequential methods, is not considered here nor is the problem-dependent question of comparing total execution times for the several iterative methods considered. Our point is that apparently sequential techniques such as SOR can be executed in parallel.

5. SUMMARY

We have considered the parallel implementation of stationary iterative techniques on an associative processor. Execution times for parallel and sequential processing have been compared with the advantage seen to be with parallel processing, an advantage that increases with system size. It was observed that the parallel processing capability of the AP is applicable not only to simultaneous displacement techniques such as Point-Jacobi or parallel relaxation but also to successive displacement techniques such as Gauss-Seidel or successive overrelaxation.

6. REFERENCES

1. Varga, R. S., "Matrix Iterative Analysis," Prentice-Hall Inc., Englewood Cliffs, N.J., 1962.
2. Wachspress, E. L., "Iterative Solution of Elliptic Systems," Prentice-Hall Inc., Englewood Cliffs, N.J., 1960.
3. Forsyth, G. and Wasow, W., "Finite Difference Methods for Partial Differential Equations," John Wiley & Sons, New York, N. Y., 1960.
4. Holland, John, "A Universal Computer Capable of Executing an Arbitrary Number of Subprograms Simultaneously," Eastern Joint Computer Conference, 1959.
5. Slotnick, D. L., "The Fastest Computer," Scientific American, Vol 224, No. 2, February 1971.
6. Batcher, K. E., "Sorting Networks and their Application," Proceedings of the Spring Joint Computer Conference, 1968.
7. Stone, H. S., "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, Vol C-20, No. 2, February 1971.
8. Rudolph, J. A., et al., "The Coming of Age of the Associative Processor," Electronics, 15 February 1971.
9. GER-15096, "STARAN IV Programming Manual," Goodyear Aerospace Corporation, Akron, Ohio, December 1970.