Solving Linear Systems on Linear Processor Arrays Using a $*$-Semiring Based Algorithm

K.N. Balasubramanya Murthy & Srinivas Aluru*
Department of Computer Science
New Mexico State University
Las Cruces, New Mexico, USA
email: {*balu, aluru*}*@cs.nmsu.edu*

S. Kamal Abdali
National Science Foundation
Arlington, VA, USA
email: *kabdali@nsf.gov*

Abstract

$*$-semirings are algebraic structures that provide a unified approach to solve several problem classes in computer science and operations research. Matrix computations over $*$-semirings are interesting because of their potential applications to linear algebra. In this paper, we present a parallel algorithm for solving systems of linear equations on $*$-semirings using linear arrays. Most of the work in solving systems of linear equations assumes the elements are drawn from a field. As fields can be treated as $*$-simirings, our algorithm can be used to solve linear equations on fields as well. Interestingly, this approach results in a faster algorithm than the known parallel algorithms for this problem on fields using linear arrays. We also present a partitioning technique for solving a problem of larger size on an array of smaller size.

Key Words: Linear equations, $*$-semirings, eliminants, asterates, parallel computers, parallel algorithm, linear arrays.

1 Introduction

The problem of solving linear equations is common to many scientific and engineering applications. The fundamental problem of the designer of algorithms, for the solution of linear equations, has been to devise means for reducing the solution time. The basic problem in the solution of linear equations is to determine the unknown solution vector, x, in the equation $Ax = b$ (where A is a real non-singular matrix of order N and b is a known right hand side vector). There are mainly two classes of methods for solving $Ax = b$, namely iterative and direct (it is also possible to have methods that have the features of both). *Iterative methods* employ a starting solution and converge to a value that is close to the exact solution by iteratively refining the starting solution. However, iterative methods do not guarantee a solution for all systems of linear equations and they are more frequently used for solving large sparse systems of linear equations. On the other hand,

*Research supported by NSF CAREER award under CCR-9702991

direct methods are those in which for a given linear system of equations, exact solutions (within a round-off bound determined by machine precision) can be obtained with a pre-estimated number of arithmetic operations. The central idea of direct methods of solving linear equations is:

1. either to find the inverse of A and then compute $x = A^{-1}b$. The method based on $*$-semirings uses this concept without explicitly finding A^{-1} (but computes $A^{-1}b$ implicitly).

2. or to transform the coefficient matrix, A, into an equivalent triangular or diagonal form in which coupling between the unknowns is reduced. The methods based on Gaussian Elimination, LU decomposition, Givens rotations, and Householder reductions use the concept of converting the coefficient matrix A into an equivalent triangular form and then solve the resulting triangular system of linear equations. On the other hand, the methods based on Gauss-Jordan, Cramer's Rule, and Bidirectional Gaussian Elimination convert the given coefficient matrix A into diagonal form and then obtain the solution vector x through divisions.

There is a large amount of literature on parallel numerical algebra in the form of books and research papers [4, 8, 10, 11, 14, 18, 24, 25, 27]. Much of the large body of literature that deals with parallel linear system solvers focuses only on the parallelization of the sequential methods and to devise efficient strategies for scheduling coarse, medium, or fine grain tasks in them onto the processors in a parallel computer system. In [1, 7, 21, 26, 27], parallel algorithms based on Gaussian elimination, Gauss-Jordan, LU decomposition, Cholesky factorization, and Givens rotations for solving dense linear systems are discussed. In these papers, attempts have been made to improve the performance of the algorithms on different parallel computer architectures by using strategies such as pipelining, load balancing, subcube matrix decomposition, recursive matrix duplication, and vector segmentation. An overlaying technique (which is closely related to the idea of iterative refinement) has been proposed in [3] for solving linear equations in real-time computing. This technique utilizes an approximation of an inverse matrix as a sum of matrix products which allows the required solution to be obtained in $O(N^2)$ rather than in $O(N^3)$ operations.

Dongarra and others [9] discuss implementations of various forms of the QR factorization on Deneclor HEP. The authors have taken three algorithms, namely, Householder method based on high level modules, a windowed Householder method that avoids fork-join synchronization, and pipelined Givens method for their study, and concluded that pipelined Givens method is preferred as it involves fewer array references [9]. Nash and Hansen described an array with $\frac{3N^2}{2}$ processors taking $5N$ time steps for solving the problem using the Faddeeva algorithm in [23]. The overall perspective of parallel algorithms for solving dense, band, or block-structured problems arising in major areas of direct solution of linear systems, least square computations, eigenvalue and singular value computations, and rapid elliptic solvers have been provided in [10]. More emphasis has been given in this paper to certain computational primitives whose efficient execution on parallel and

Year	Author(s)	Processors	Time steps	Remarks
1990	Benaini & Robert [2]	$n-1$	n^2+n-1	Triangularization GE Algorithm with no pivoting
1992	Wyrzykoski [28]	n	n^2+2n-3	Triangularization GE Algorithm with partial pivoting
1995	Wyrzykoski & others [29]	$\frac{n+2}{2}$	$2n(n+1)$	Complete Solution GJ Algorithm with partial pivoting
1998	This Design	n	$\frac{n^2+5n-4}{2}$	Complete Solution $*$-semirings based Algorithm

Table 1: Comparison of various algorithms using $O(n)$ processors to solve a system of n linear equations.

vector computers is essential to obtain high performance algorithms.

It is well known that the performance of a parallel algorithm depends on the architecture of the multiprocessor on which it is implemented. It is possible that good parallel numerical algorithms may be obtained by viewing the algebraic problem afresh and developing methods which not only expose the available concurrency but also break the existing sequentiality in the solution procedure. Such an attempt is made in [16] which describes a $*$-semiring based algorithm using eliminants for solving the given set of linear equations on the PRAM model. In this paper, we discuss the implementation of this algorithm on a linear array of processors and its problem partitioning capabilities. A comparison of the various schemes that use $O(n)$ processors is given in Table 1.

The rest of the paper is organized as follows. In the next section, we discuss the basics of $*$-semirings along with the algorithm using eliminants for solving linear equations. The implementation of the algorithm on linear arrays and a problem partitioning technique to solve a problems of larger size on an array of smaller size are given in section 3. Lastly, section 4 gives our conclusions.

2 Solution of Linear Algebraic Equations

In a linear system $Ax = b$, the value of x_i $(i = 1, 2, ..., N)$ depends on the value of x_j $(j = 1, 2, ..., N$ and $i \neq j)$ indicating $(N-1)^{th}$ level dependency. Hence the influence of at most $(N-1)$ other unknowns has to be unraveled to find the solution to one unknown.

2.1 $*$-Semirings and Eliminants

$*$-semirings (also called *closed semirings*) are algebraic structures that provide a unified approach to solve a number of problems in computer science and operations research. Examples of such problems include finding shortest or most reliable paths in graphs, finding maximum network flows, cutset enumeration, computing the transitive closure of binary relations, finding the regular expressions to describe the language accepted by finite automaton, and solving linear systems of equations. The reader is referred to [12] for a detailed bibliography on $*$-semirings.

A semiring is an algebraic structure defined as $< S, +, ., 0, 1 >$ where S is a set, $+$ and . are operators, and 0 and 1 are identity elements for $+$ and ., respectively. A semiring with a unary operation called *asteration* (denoted by *) is referred to as a $*$-*semiring*. Asteration satisfies the law $a^* = aa^* + 1 = a^*a + 1$. The structure is closed with respect to addition and multiplication, but may not be closed with respect to asteration. Any field can be converted into a $*$-semiring by defining $a^* = 1/(1-a),\ \forall\ a \neq 1$. The concept of eliminant is introduced in [15] to give closed form expressions for matrix asterates and for describing solutions of linear equations over *-semirings.

Eliminants, which are akin to Schur products for fields, are defined as follows:

$$elim \left\{ \begin{array}{c} a \end{array} \right\} = a$$

$$elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{array} \right\} = a_{1,1} + a_{1,0}a_{0,0}^* a_{0,1}$$

$$elim \left\{ \begin{array}{ccc} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{array} \right\} = elim \left\{ \begin{array}{cc} elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{array} \right\} elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,2} \\ a_{1,0} & a_{1,2} \end{array} \right\} \\ elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,1} \\ a_{2,0} & a_{2,1} \end{array} \right\} elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,2} \\ a_{2,0} & a_{2,2} \end{array} \right\} \end{array} \right\}$$

$$= elim \left\{ \begin{array}{cc} a_{1,1} + a_{1,0}a_{0,0}^* a_{0,1} & a_{1,2} + a_{1,0}a_{0,0}^* a_{0,2} \\ a_{2,1} + a_{2,0}a_{0,0}^* a_{0,1} & a_{2,2} + a_{2,0}a_{0,0}^* a_{0,2} \end{array} \right\}$$

$$= (a_{2,2} + a_{2,0}a_{0,0}^* a_{0,2}) + (a_{2,1} + a_{2,0}a_{0,0}^* a_{0,1})(a_{1,1} + a_{1,0}a_{0,0}^* a_{0,1})^*(a_{1,2} + a_{1,0}a_{0,0}^* a_{0,2})$$

and in general

$$elim \left\{ \begin{array}{cccc} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{array} \right\}$$

$$= elim \left\{ \begin{array}{ccc} elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{array} \right\} & \cdots & elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,n-1} \\ a_{1,0} & a_{1,n-1} \end{array} \right\} \\ elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,1} \\ a_{2,0} & a_{2,1} \end{array} \right\} & \cdots & elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,n-1} \\ a_{2,0} & a_{2,n-1} \end{array} \right\} \\ \vdots & \vdots & \vdots \\ elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,1} \\ a_{n-1,0} & a_{n-1,1} \end{array} \right\} & \cdots & elim \left\{ \begin{array}{cc} a_{0,0} & a_{0,n-1} \\ a_{n-1,0} & a_{n-1,n-1} \end{array} \right\} \end{array} \right\}$$

Eliminants can be used to express solutions of simultaneous linear equations in a form that is reminiscent of Cramer's rule but is more compact. Consider a system of n linear equations in n unknowns $x_0, x_1, \cdots, x_{n-1}$, given by

$$\begin{aligned} x_0 &= a_{0,0}x_0 + a_{0,1}x_1 + \cdots + a_{0,n-1}x_{n-1} + b_0, \\ x_1 &= a_{1,0}x_0 + a_{1,1}x_1 + \cdots + a_{1,n-1}x_{n-1} + b_1, \\ &\vdots \\ x_{n-1} &= a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots + a_{n-1,n-1}x_{n-1} + b_{n-1}. \end{aligned}$$

It is shown in [16] that a solution of this system is given by

$$x_i = \left\{ \begin{array}{cccccccc} a_{0,0} & \cdots & a_{0,i-1} & a_{0,i} & a_{0,i+1} & \cdots & a_{0,n-1} & b_0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,i-1} & a_{n-1,i} & a_{n-1,i+1} & \cdots & a_{n-1,n-1} & b_{n-1} \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \end{array} \right\}, \ i = 0, \ldots, n-1.$$

2.2 Solution of Linear Equations using $*$-semirings

The above $*$-semiring solution can be adapted for linear systems over fields also, since any field can be treated as a $*$-semiring by defining a^* to be $1/(1-a)$ for all $a \neq 1$. Throughout this paper, we use systems of linear equations on real numbers (a field) as examples for illustration. Given a system of linear equations of the form $Ax = b$, it can be rewritten as $x = (I - A)x + b$, where I is an identity matrix. Consider a system of n linear equations in n unknowns $x_1, x_2, \cdots, x_n$ given below:

$$\left[\begin{array}{ccccc} a_{0,0} & a_{0,1} & \cdots & a_{0,n-2} & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-2} & a_{1,n-1} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,n-2} & a_{2,n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{array} \right] \left[\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{array} \right] = \left[\begin{array}{c} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{array} \right]$$

Then they are rewritten in the standard form as:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} 1-a_{0,0} & -a_{0,1} & \cdots & -a_{0,n-1} & -a_{0,n} \\ -a_{1,0} & 1-a_{1,1} & \cdots & -a_{1,n-1} & -a_{1,n} \\ -a_{2,0} & -a_{2,1} & \cdots & -a_{2,n-1} & -a_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_{n,0} & -a_{n,1} & \cdots & -a_{n,n-1} & 1-a_{n,n} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

The solution of this system of equations is therfore given by

$$x_i \;=\; elim \left\{ \begin{array}{ccccccc} (1-a_{0,0}) & -a_{0,1} & \cdots & -a_{0,i} & \cdots & -a_{0,n-1} & b_0 \\ -a_{1,0} & (1-a_{1,1}) & \cdots & -a_{1,i} & \cdots & -a_{1,n-1} & b_1 \\ -a_{2,0} & -a_{2,1} & \cdots & -a_{2,i} & \cdots & -a_{2,n-1} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_{n-1,0} & -a_{n-1,1} & \cdots & -a_{n-1,i} & \cdots & 1-a_{n-1,n-1} & b_{n-1} \\ 0 & 0 & \cdots & 1 & \cdots & 0 & 0 \end{array} \right\}$$

We now demonstrate this method with an example of solving two linear equations.

Example:

Consider a system of two equations:

$$\begin{bmatrix} 5 & -3 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \;=\; \begin{bmatrix} 1 \\ 9 \end{bmatrix}$$

The system of equations is rewritten as:

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \;=\; \begin{bmatrix} -4 & 3 \\ -3 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \;+\; \begin{bmatrix} 1 \\ 9 \end{bmatrix}$$

Hence

$$x_0 = \; elim \left\{ \begin{array}{ccc} -4 & 3 & 1 \\ -3 & 0 & 9 \\ 1 & 0 & 0 \end{array} \right\} \;=\; elim \left\{ \begin{array}{cc} elim \left\{ \begin{array}{cc} -4 & 3 \\ -3 & 0 \end{array} \right\} elim \left\{ \begin{array}{cc} -4 & 1 \\ -3 & 9 \end{array} \right\} \\ elim \left\{ \begin{array}{cc} -4 & 3 \\ 1 & 0 \end{array} \right\} elim \left\{ \begin{array}{cc} -4 & 1 \\ 1 & 0 \end{array} \right\} \end{array} \right\}$$

$$= \; elim \left\{ \begin{array}{cc} 0+(-3)(-4)^*3 & 9+(-3)(-4)^*1 \\ 0+1(-4)^*3 & 0+1(-4)^*1 \end{array} \right\} \;=\; elim \left\{ \begin{array}{cc} \frac{-9}{5} & \frac{42}{5} \\ \frac{3}{5} & \frac{1}{5} \end{array} \right\}$$

$$= \; (\frac{1}{5}) + (\frac{3}{5})(\frac{-9}{5})^* (\frac{42}{5}) \;\; = \; (\frac{1}{5}) + (\frac{3}{5})(\frac{5}{14})(\frac{42}{5}) \;=\; 2$$

$$x_1 = elim \begin{Bmatrix} -4 & 3 & 1 \\ -3 & 0 & 9 \\ 0 & 1 & 0 \end{Bmatrix} = elim \begin{Bmatrix} elim \begin{Bmatrix} -4 & 3 \\ -3 & 0 \end{Bmatrix} elim \begin{Bmatrix} -4 & 1 \\ -3 & 9 \end{Bmatrix} \\ elim \begin{Bmatrix} -4 & 3 \\ 0 & 1 \end{Bmatrix} elim \begin{Bmatrix} -4 & 1 \\ 0 & 0 \end{Bmatrix} \end{Bmatrix}$$

$$= elim \begin{Bmatrix} 0+(-3)(-4)^*3 & 9+(-3)(-4)^*1 \\ 1+0(-4)^*3 & 0+0(-4)^*1 \end{Bmatrix} = elim \begin{Bmatrix} \frac{-9}{5} & \frac{42}{5} \\ 1 & 0 \end{Bmatrix}$$

$$= 0+(1)(\frac{-9}{5})^*(\frac{42}{5}) = 0+(1)(\frac{5}{14})(\frac{42}{5}) = 3$$

In order to solve a system of n linear equations in n unknowns, we need the values of n eliminants of order $n+1$ each. But these $n+1$ eliminants share the first n rows, and differ only in the $n+1^{th}$ row. The commonality among the eliminants can be utilized to save computation by organizing their elements in the following 2-dimensional array:

$$\begin{matrix} 1-a_{0,0} & -a_{0,1} & \cdots & -a_{0,n-1} & b_0 \\ -a_{1,0} & 1-a_{1,1} & \cdots & -a_{1,n-1} & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_{n-1,0} & -a_{n-1,1} & \cdots & 1-a_{n-1,n-1} & b_{n-1} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{matrix}$$

The eliminant computation needs to be carried out in the above formulation for n steps and at any step k, the rows that are operated on are $k, k+1, k+2, \cdots, n+k$ and similarly, the columns that are used in the computations are $k, k+1, k+2, \cdots, n$. At the end of n steps, we would get the solution vector x of n elements. The method can be described in a sequential algorithmic form as given below (with b_i's stored as $a_{i,n}$):

PROGRAM $*$-SEMIRINGS

BEGIN

(* INITIALIZATION PHASE *)

FOR $i = 0$ TO $n-1$ DO

FOR $j = 0$ TO $n-1$ DO

IF $(i = j)$ THEN $a_{i,j} = 1 - a_{i,j}$

ELSE $a_{i,j} = -a_{i,j}$

FOR $i = n$ TO $2n-1$ DO

$(1-a_{0,0})^{(0)}$	$-a_{0,1}^{(0)}$	$-a_{0,2}^{(0)}$	$-a_{0,3}^{(0)}$	$b_0^{(0)}$
$-a_{1,0}^{(0)}$	$(1-a_{1,1})^{(0)}$	$-a_{1,2}^{(0)}$	$-a_{1,3}^{(0)}$	$b_1^{(0)}$
$-a_{2,0}^{(0)}$	$-a_{2,1}^{(0)}$	$(1-a_{2,2})^{(0)}$	$-a_{2,3}^{(0)}$	$b_2^{(0)}$
$-a_{3,0}^{(0)}$	$-a_{3,1}^{(0)}$	$-a_{3,2}^{(0)}$	$(1-a_{3,3})^{(0)}$	$b_3^{(0)}$
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0

Table 2: Progression of the $*$-semirings algorithm for solving a system of 4 equations - step 0

FOR $j = 0$ TO n DO
IF $(i - n = j)$ THEN $a_{i,j} = 1$
ELSE $a_{i,j} = 0$
(* SOLUTION PHASE *)
FOR $k = 0$ TO $(n-1)$ DO
$temp = 1/(1 - a_{k,k})$
FOR $i = k+1$ TO $n+k$ DO
FOR $j = k+1$ TO n DO
$a_{i,j} = a_{i,j} + a_{i,k} * a_{k,j} * temp$
FOR $i = n$ TO $2n-1$ DO
$x_{i-n} = a_{i,n}$
END;

A schematic representation of the entire solution process for solving a system of 4 equations is given in Tables 2, 3, 4, and 5. The algorithm requires n virtual time steps to produce the solution vector, x. The total serial complexity in terms of time steps (a time step is defined as the amount of time taken to update an element) is given by:

$$T(n,1) = \sum_{k=0}^{n-1} \{n(n-k)\} = \frac{n^3+n^2}{2}$$

where $T(n,1)$ represents the time complexity for solving a problem size of n on one processor. For a system of 8 linear equations, the order in which the elements are updated using the $*$-semiring algorithm is shown in Table 7.

2.3 Parallel Elimination Pattern in $*$-semiring Algorithm

The sequential $*$-semiring algorithm updates one element at a time to reduce the system $Ax = b$ to $A^{-1}b$ form. This requires a total of $\frac{n^3+n^2}{2}$ time steps to produce the complete solution. In this

$a_{1,1}^{(1)}$	$a_{1,2}^{(1)}$	$a_{1,3}^{(1)}$	$b_1^{(1)}$
$a_{2,1}^{(1)}$	$a_{2,2}^{(1)}$	$a_{2,3}^{(1)}$	$b_2^{(1)}$
$a_{3,1}^{(1)}$	$a_{3,2}^{(1)}$	$a_{3,3}^{(1)}$	$b_3^{(1)}$
$a_{4,1}^{(1)}$	$a_{4,2}^{(1)}$	$a_{4,3}^{(1)}$	$b_4^{(1)}$
1	0	0	0
0	1	0	0
0	0	1	0

Table 3: Progression of the $*$-semirings algorithm for solving a system of 4 equations - step 1

$a_{2,2}^{(2)}$	$a_{2,3}^{(2)}$	$b_2^{(2)}$
$a_{3,2}^{(2)}$	$a_{3,3}^{(2)}$	$b_3^{(2)}$
$a_{4,2}^{(2)}$	$a_{4,3}^{(2)}$	$b_4^{(2)}$
$a_{5,2}^{(2)}$	$a_{5,3}^{(2)}$	$b_5^{(2)}$
1	0	0
0	1	0

Table 4: Progression of the $*$-semiring algorithm for solving a system of 4 equations - step 2

$a_{3,3}^{(3)}$	$b_3^{(3)}$
$a_{4,3}^{(3)}$	$b_4^{(3)}$
$a_{5,3}^{(3)}$	$b_5^{(3)}$
$a_{6,3}^{(3)}$	$b_6^{(3)}$
1	0

Table 5: Progression of the $*$-semiring algorithm for solving a system of 4 equations - step 3

$b_4^{(4)}(x_0)$
$b_5^{(4)}(x_1)$
$b_6^{(4)}(x_2)$
$b_7^{(4)}(x_3)$

Table 6: Solution vector obtained after solving a system of 4 equations - step 4

	1							
	9	65						
	17	72	121					
	25	79	127	169				
	33	86	133	174	209			
	41	93	139	179	213	241		
	49	100	145	184	217	244	265	
	57	107	151	189	221	247	267	281 (x_0)
		114	157	194	225	250	269	282 (x_1)
			163	199	229	253	271	283 (x_2)
				204	233	256	273	284 (x_3)
					237	259	275	285 (x_4)
						262	277	286 (x_5)
							279	287 (x_6)
								288 (x_7)

Table 7: Sequential pattern of updating elements in the $*$-semiring algorithm (the numbers indicate the time step at which the element is updated)

	1							
	2	10						
	3	11	18					
	4	12	19	25				
	5	13	20	26	31			
	6	14	21	27	32	36		
	7	15	22	28	33	37	40	
	8	16	23	29	34	38	41	43 (x_0)
		17	24	30	35	39	42	44 (x_1)
			25	31	36	40	43	45 (x_2)
				32	37	41	44	46 (x_3)
					38	42	45	47 (x_4)
						43	46	48 (x_5)
							47	49 (x_6)
								50 (x_7)

Table 8: Parallel pattern of updating elements in the $*$-semiring algorithm (the numbers indicate the time step at which the element is updated)

algorithm, n rows with $n-k+1$ elements in each row are updated at any step k. If we employ a linear array of n processors with pipelined updating, it is possible to obtain the complete solution in $\frac{n^2+5n-4}{2}$ time steps. Table 8 shows the parallel update pattern for a system of 8 linear equations.

3 $*$-semiring Based Algorithm on a Linear Array

A key issues in parallel computing is efficient task partitioning and scheduling: one first identifies a suitable task granularity (a measure of the amount of computation involved) of the problem, and then applies a scheduling method to allocate the computational tasks of a parallel program (or algorithm) onto the available processors in a multiprocessor system such that the completion time (or schedule length or makespan or program execution time) is minimized. The standard task size mostly depends on the computation and communication capabilities of the multiprocessor used for executing the task system without violating the precedence constraints.

Table 8 illustrates that the algorithm has a systematic update pattern. The update of elements requires operations to be performed on adjacent rows. Therefore, an architecture with neighbor processor communication, such as linear array or mesh, is sufficient to implement the algorithm in

step	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8
1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10
3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,11
4	4,5	4,6	4,7	4,8	4,9	4,10	4,11	4,12
5	5,6	5,7	5,8	5,9	5,10	5,11	5,12	5,13
6	6,7	6,8	6,9	6,10	6,11	6,12	6,13	6,14
7	7,8	7,9	7,10	7,11	7,12	7,13	7,14	7,15

Table 9: The rows that are operated upon by the various processors during the progression of the $*$-semiring based algorithm for solving a system of 8 equations on 8 processors

parallel.

Consider a linear array of n processors to solve a system of n linear algebraic equations using the $*$-semiring based algorithm. Let us assume that the processors (fitted with communication co-processors) in the linear array are connected by bidirectional links for parallel data transfer. Since the algorithm operates on rows, the ideal choice would be to allocate one row of the linear system per processor. To begin with, processor p_i for $i = 0, 1, \cdots, n-1$ is allocated row i and the algorithm progression takes place in n steps. Initial allocation of rows for solving a system of 8 equations on a linear array of 8 processors is shown in Figure 1. At any step k for $k = 0, 1, \cdots, n-1$, processor i sends data to and receives data from the processor $i+1$ in a pipelined manner (to be more precise $n-k+1$ elements are sent and received at each step k). However, processor $n-1$ generates data in a systolic fashion row-wise. The rows generated by processor $n-1$ are $n, n+1, \cdots, 2n-1$. At the end of the algorithmic progression, processor p_i will have the solution of the unknown x_i. Table 9 indicates the rows that are operated upon by the processors at the various steps of the algorithm progression.

The total parallel completion time for solving a system of n equations using the $*$-semiring based algorithm in terms of time steps can be computed as follows:

$$T(n,n) = \underbrace{n}_{first\ column} + \underbrace{\sum_{k=1}^{n-1}(n-k+2)}_{rest\ of\ the\ columns}$$

$$= \frac{(n^2+5n-4)}{2}$$

where $T(n,n)$ represents the total number of time steps required for solving n linear equations using

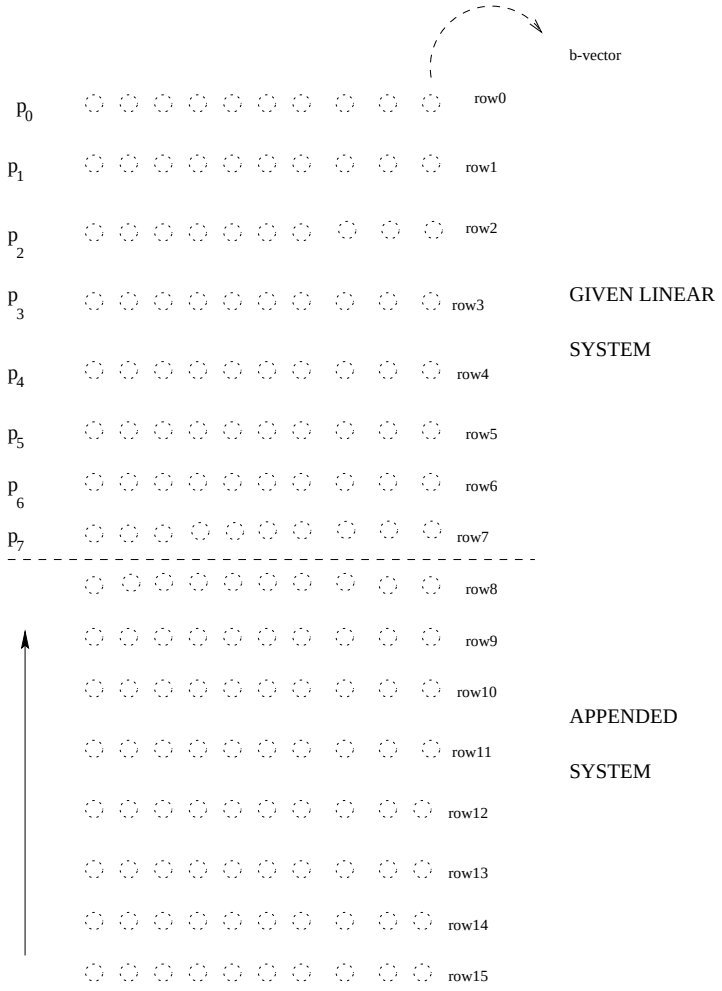

Figure 1: Allocation of rows to processors for solving a system of 8 linear equations

the $*$-semiring based algorithm on a linear array of n processors. We can easily conclude that the $*$-semiring algorithm is faster than the existing algorithm assuming the linear array architecture permits overlapping computation and communication.

3.1 A Problem Partitioning Technique

The linear array design presented in the previous section requires an array of n processors to solve linear equations of size n or less. In this section, we present a problem partitioning scheme using which a given linear array can handle problems of any size. The basic idea here is to divide the problem into a number of blocks such that the number of blocks equals the number of processors in the linear array. This way each block can be mapped onto one processor in the linear array and the computation can proceed as though the size of the problem is the same as that of the hardware. In each processor, however, one full block of problem must be executed instead of one row and this would be done serially. Here the assumption is that the processors in the linear array are capable of handling large amounts of data.

Let p denote the size of the linear array and n denote the size of the problem. Assume $p < n$ and n is a multiple of p. By dividing n by p, we get a block size of $\frac{n}{p}$. Instead of mapping one row per processor, we map a block of $\frac{n}{p}$ rows to each processor. With this, each processor now performs computations on $\frac{n}{p}$ rows sequentially. This partitioning scheme is depicted in Figure 2 for solving 8 linear algebraic equations on a linear array of 4 processors.

This scheme has almost no control overhead or communication overhead as we assume that the required data is stored in the respective processors all the time. Apart from the initial mapping of the n rows to p processors, this scheme has no other subsequent mappings or unmappings. This scheme supports any problem size without requiring any fundamental change in the flow of data or additional overheads.

4 Conclusions

In this paper, we presented a parallel algorithm for solving systems of linear equations on $*$-semirings using linear arrays. Since most applications of solving systems of linear equations involve elements drawn from fields, much of the work has focussed in this area. Our results demonstrate that faster algorithms can be obtained by considering $*$-semirings and treating fields as special cases of $*$-semirings. It is important to study the numerical stability of the $*$-semirings based approach, especially in comparison to the numerical stability characteristics of the algorithms for fields. We are currently exploring this issue.

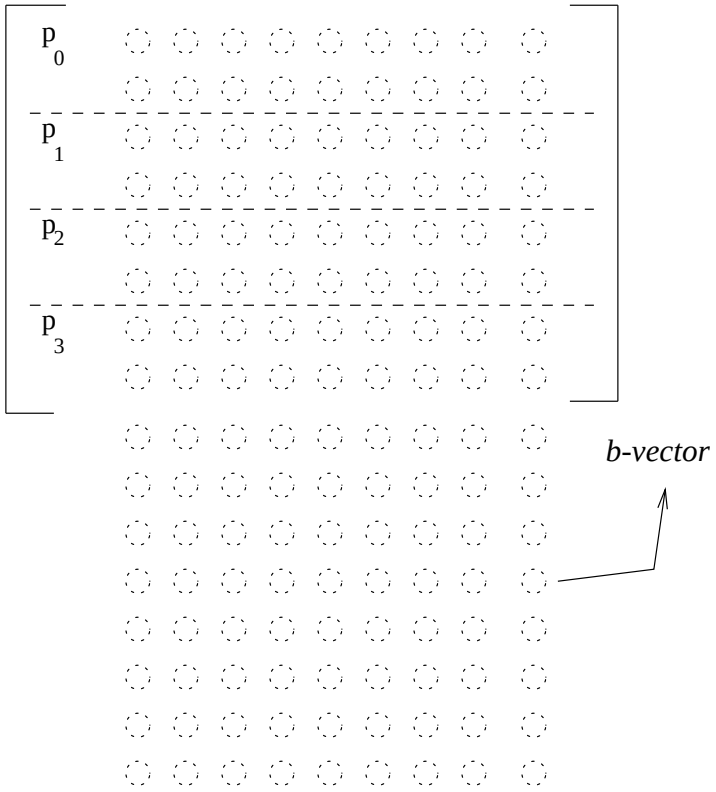

Figure 2: Solving a system 8 equations on an array of 4 processors using the $*$-semiring based algorithm

References

[1] K.N. Balasubramanya Murthy, K.Bhuvaneswari, and C. Siva Ram Murthy, "A new algorithm based on Givens rotations for solving linear equations on fault tolerant mesh-connected processors", *IEEE Transactions on Parallel and Distributed Systems*, vol.9, No.8, August 1998, pp.825-832.

[2] A. Benaini and Y. Robert, "A modular systolic linear array for Gaussian elimination", *International Journal of Computer Mathematics*, vol. 36, 1990, pp. 105-118.

[3] S. Y. Berkovich, "An overlaying technique for solving linear equations in real-time computing", *IEEE Transactions on Computers*, vol. 42, no. 5, May 1993, pp. 513-517.

[4] D.P.Bertsekas and J.N.Tsitsiklis, *Parallel and Distributed Computations - Numerical Methods*, Prentice Hall Inc., New Jersey, 1989.

[5] A. Bojanczyk, R. P. Brent, and H. T. Kung, "Numerically stable solution of dense systems of linear equations using mesh-connected processors", *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 1, March 1984, pp. 95-104.

[6] K. Bhuvaneswari, K. N. Balasubramanya Murthy and C. Siva Ram Murthy, "A new and faster Gaussian elimination based fault tolerant systolic linear system solver", *Journal of Parallel and Distributed Computing*, vol. 44, 1997, pp. 107-122.

[7] M.Cosnard, M.Tchuente, and B.Tourancheau, "Systolic Gauss- Jordan elimination for dense linear systems", *Parallel Computing*, vol.10, 1989, pp.117-122.

[8] J.W.Demmel, M.T.Heath, and H.A. Van der Vorst,"Parallel numerical algebra", Acta Numerica, 1993, pp.111-197.

[9] J.J.Dongarra, A.Sameh, and D.C.Sorenson, "Implementation of some concurrent algorithms for matrix factorization", *Parallel Computing*, vol.3, 1986, pp.25-34.

[10] K.Gallivan, R.J.Plemmons, and A.H.Sameh, "Parallel algorithms for dense linear algebra computations", *SIAM Review*, vol.32, no.1, March 1990, pp.54-135.

[11] G.Golub and C.Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, Maryland, 1989.

[12] M.Gondran and M.Minoux, *Graphs and Algorithms*, Wiley, New York, 1984.

[13] M.T.Heath and C.H.Romine, "Parallel solution of triangular systems on distributed-memory multiprocessors", *SIAM Journal on Scientific and Statistical Computing*, vol.9, no.3, May 1988, pp.558-587.

[14] D.Heller, "A survey of parallel algorithms in numerical linear algebra", *SIAM Review*, vol.20, no.4, October 1978, pp.740-777.

[15] S. Kamal Abdali and B.D.Saunders, "Transitive closure and related semiring properties via eliminants", *Theoretical Computer Science*, vol.40, 1985, pp.257-274.

[16] S. Kamal Abdali, "Parallel computations in $*$-semirings", in *Computational Algebra*, K.G. Fischer, P. Loustaunau, et. al. (eds.), Marcel Dekker, New York, 1994.

[17] T. Kimura, *"Gauss-Jordan elimination by VLSI mesh-connected processors"*, Infotech State of the Art Report: Supercomputers, (Josshope, C. and Hockney, R. eds.), Vol.2 (InfoTech, Maidenhead, United Kingdom, 1979), pp. 271-290.

[18] S.Lakshmivarahan and S.K.Dhall, *Analysis and Design of Parallel Algorithms - Arithmetic and Matrix Problems*, McGraw-Hill Publishing Company, New York, 1990.

[19] G.Li and T.F.Coleman, "A parallel triangular solver for a distributed-memory multiprocessor", *SIAM Journal on Scientific and Statistical Computing*, vol.9, no.3, May 1988, pp.485-502.

[20] C.J.Lin, "Systolic algorithm for the solution of dense linear equations", *International Journal of Computer Mathematics*, vol.35, 1990, pp.159-167.

[21] R.Melhem, "Parallel Gauss-Jordan elimination for solution of dense linear equations", *Parallel Computing*, vol.4, 1987, pp.339-343.

[22] R.K.Montoye and D.H.Lawrie, "A practical algorithm for solution of triangular systems on a parallel processing system", *IEEE Transactions on Computers*, vol.31, no.11, November 1982, pp.1076-1082.

[23] J.G.Nash and S.Hansen, "Modified Faddeeva algorithm for concurrent execution of linear algebraic equations", IEEE Transactions on computers, vol.37, no.2, February 1988, pp.129-136.

[24] J.M.Ortega and R.G.Voigt, "Solution of partial differential equations on vector and parallel computers", *SIAM Review*, vol.27, no.2, June 1985, pp.149-240.

[25] A. H. Sameh and D. J. Kuck, "Parallel direct linear system solvers - A survey", *Parallel Computers - Parallel Mathematics, M. Feilmeier (ed.)*, North-Holland, Amsterdam, The Netherlands, 1977, pp. 25-30.

[26] Shietung Peng and Stanislav Sedukhin, "Array processors design for division-free linear system solving", *The Computer Journal*, vol. 39, no. 8, 1996, pp. 713-722.

[27] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing - Design and Analysis of Algorithms*, Benjamin/Cummings Publishing Company, 1994.

[28] R. Wyrzykowski, "Processor arrays for matrix triangularization with partial pivoting", *IEE Proc. (part E - Computers and Digital Techniques)*, vol. 139, no. 2, March 1992, pp. 165-169.

[29] R. Wyrzykowski, J. S. Kanevski, and H. Piech, "One-dimensional processor arrays for linear algebraic problems", *IEE Proc. (part E - Computers and Digital Techniques)*, vol. 142, no. 1, January 1995, pp. 1-4.