# Parallel Computations in *-Semirings

*S. Kamal Abdali**
Division of Computer and Computation Research
National Science Foundation
Arlington, VA 22230

**Abstract**

*-semirings are algebraic structures that provide a unified approach to several problem classes in computer science and operations research. For example, *-semirings can be used to describe the algebra related to regular expressions, graph-theoretical path problems, and compiled-code optimization. The theory of matrices over *-semirings has a number of similarities to linear algebra. For example, eliminants and asterates (closures) behave analogously in many ways to determinants and matrix inverses. Matrix computations over *-semirings are interesting in their own right as well as because of their potential applications to linear algebra. This paper uses the eliminant formulation of *-semiring properties to derive parallel algorithms for three kinds of problems involving matrices over *-semirings: eliminant computation, solution of linear systems over *-semirings, and matrix asteration. The algorithms discussed allow the most general computations in which the asteration operation in the base *-semiring is assumed to be non-trivial, and the matrices are assumed to be dense and without any particular structure.

## 1 Introduction

**-semirings* (also called *closed semirings*) are algebraic structures that provide a unified approach to 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 expression to describe the language accepted by a finite automaton, solving systems of linear equations, etc. The reader is referred to Conway[6], Carré[2], Tarjan[13], and Gondran & Minoux[8] for some of the most notable formulations of the *-semiring approach to these and other problems; the last reference also contains an extensive bibliography.

A large number of parallel and systolic algorithms have been published for matrix computations, path algebra, and other related problems (see, e.g., [3, 4, 5, 7, 10, 11, 12, 14]). Most of these computations belong to specialized *-semirings, and the algorithms exploit their special properties to attain high efficiencies. Several matrix computation algorithms are meant for sparse or specially

*The opinions expressed in this paper belong to the author, and should not be construed as the official viewpoint of the National Science Foundation.

structured matrices, and take advantage of these features for efficiency. The fact that in linear algebra the base *-semiring is actually a field is sometimes a source of algorithm acceleration. For example, then one can use fast matrix multiplication for obtaining fast algorithms for many other matrix computations. In the case of several path problems, the asteration (closure, star operation) of matrices is made significantly simpler because the asteration of base elements is trivial.

This paper discusses parallel computations over general *-semirings, involving dense, unstructured matrices. The concept of *eliminant* was introduced in a previous work[1] to give closed form expressions for matrix asterates and for describing solutions of linear systems of equations in a form reminiscent of Cramer's rule. Here we use the same eliminant formulation to obtain very simple parallel algorithms for these computations.

## 2 Basic Definitions and Properties

We summarize here the necessary definitions needed in this paper, referring the reader to [1] for details of the eliminant approach to *-semiring properties. A *-semiring is an algebraic structure consisting of a set together with two binary operations *addition* (denoted, $+$) and *multiplication* (denoted $\cdot$, or by juxtaposition), a unary operation *asteration*[1] (denoted $^*$), and two distinguished elements 0 and 1. Addition is associative and commutative, and has 0 as its identity element ($a+0=0+a=a$). Multiplication is associative, has 1 as its identity element ($a1=1a=a$), has 0 as zero ($a0=0a=0$), and is left and right distributive over addition. 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. In particular, $1^*$ need not be defined.

For a fixed positive integer $n$, the set of $n \times n$ matrices over a *-semiring S can itself be made a *-semiring $\mathcal{S}_n$ as follows: The zero of $\mathcal{S}_n$ is the matrix consisting entirely of zeros of S. The 1 of $\mathcal{S}_n$ is the identity matrix $\mathbf{I}$ with 1's along the principal diagonal, and 0's elsewhere. The addition and multiplication in $\mathcal{S}_n$ are defined in the usual way. Matrix asteration is defined in different ways in the literature. Here we pursue the approach of [1] to define asterates in terms of eliminants. With each square array of elements of a *-semirings, we associate a value in the *-semiring, and call this value the *eliminant* (of that array). We denote the eliminant of a square array by enclosing the array between braces $(\{\})$[2] or by $\mathrm{elim}(A)$ if the array has been given as a square matrix $A$. An

[1] Asteration is often called *closure* or *star*. The term *asterate* of $a$ for $a^*$ is found in [6].

[2] In [1], the notation for eliminants was the same as for determinants, since the two concepts have many similarities. The notation has been changed to avoid any possible confusion.

eliminant is evaluated as follows:

$$\begin{aligned} \{a\} &= a, \\ \left\{ \begin{array}{cc} a & b \\ c & d \end{array} \right\} &= d + ca^*b, \end{aligned}$$

$$\left\{ \begin{array}{ccccc} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{array} \right\} = \left\{ \begin{array}{ccccc} \left\{ \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right\} & \left\{ \begin{array}{cc} a_{11} & a_{13} \\ a_{21} & a_{23} \end{array} \right\} & & \dots & \left\{ \begin{array}{cc} a_{11} & a_{1n} \\ a_{21} & a_{2n} \end{array} \right\} \\ \left\{ \begin{array}{cc} a_{11} & a_{12} \\ a_{31} & a_{32} \end{array} \right\} & \left\{ \begin{array}{cc} a_{11} & a_{13} \\ a_{31} & a_{33} \end{array} \right\} & & \dots & \left\{ \begin{array}{cc} a_{11} & a_{1n} \\ a_{31} & a_{3n} \end{array} \right\} \\ \vdots & \vdots & & & \vdots \\ \left\{ \begin{array}{cc} a_{11} & a_{12} \\ a_{n1} & a_{n2} \end{array} \right\} & \left\{ \begin{array}{cc} a_{11} & a_{13} \\ a_{n1} & a_{n3} \end{array} \right\} & & \dots & \left\{ \begin{array}{cc} a_{11} & a_{1n} \\ a_{n1} & a_{nn} \end{array} \right\} \end{array} \right\}, n > 2.$$

**Example**

$$\begin{aligned} \left\{ \begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & i \end{array} \right\} &= \left\{ \begin{array}{cc} \left\{ \begin{array}{cc} a & b \\ d & e \end{array} \right\} & \left\{ \begin{array}{cc} a & c \\ d & f \end{array} \right\} \\ \left\{ \begin{array}{cc} a & b \\ g & h \end{array} \right\} & \left\{ \begin{array}{cc} a & c \\ g & i \end{array} \right\} \end{array} \right\} = \left\{ \begin{array}{cc} e + da^*b & f + da^*c \\ h + ga^*b & i + ga^*c \end{array} \right\} \\ &= i + ga^*c + (h + ga^*b)(e + da^*b)^*(f + da^*c). \end{aligned}$$

Asteration can now be defined in terms of eliminants. Given a matrix of order $n$, its asterate is a matrix, also of order $n$, each of whose elements is an eliminant of order $n + 1$ whose elements, in turn, are those of the given matrix augmented with a row and a column of appropriately chosen 1's and 0's, as follows. Suppose

$$A = \left[ \begin{array}{ccc} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{array} \right].$$

Then $A^*$ is the matrix $B$ whose generic element $b_{ij},\ i = 1, \dots, n,\ j = 1, \dots, n$, is given by:

$$b_{ij} = \left\{ \begin{array}{ccccccccc} a_{11} & \dots & a_{1,i-1} & a_{1i} & a_{1,i+1} & \dots & a_{1n} & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{j-1,1} & \dots & a_{j-1,i-1} & a_{j-1,i} & a_{j-1,i+1} & \dots & a_{j-1,n} & 0 \\ a_{j,1} & \dots & a_{j,i-1} & a_{j,i} & a_{j,i+1} & \dots & a_{j,n} & 1 \\ a_{j+1,1} & \dots & a_{j+1,i-1} & a_{j+1,i} & a_{j+1,i+1} & \dots & a_{j+1,n} & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & \dots & a_{n,i-1} & a_{ni} & a_{n,i+1} & \dots & a_{nn} & 0 \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 \end{array} \right\}.$$

Note that the above eliminant is obtained by bordering A one deep at right and bottom; the bordering elements are all zero, except for 1's in row $j$ and column $i$. Alternatively, the augmented row (resp., column) is the $i$th row (resp., the $j$th column) of the identity matrix $I$.

It is proved in [1] that $A^* = AA^* + I = A^*A + I$, so that the above definition of asteration is justified.

The references cited in the introduction contain many examples of *-semirings. Also, any field can be made into a *-semiring by defining $a^* = 1/(1-a)$ for all $a \neq 1$. For several network problems, the asteration operation, which is of crucial importance for matrices over the appropriate *-semiring, is either not needed or is trivial for the base *-semiring itself. (For example, $a^*$ might be 1 for all $a$). *-semirings with non-trivial asteration include the algebra of regular expressions and the field of reals made into a *-semiring as above.

From the law that asterates satisfy, it is clear that $a^*$ is a solution[3] of the equation $x = ax + b$. It is convenient to express systems of linear equations in $n$ unknowns $x_1, x_2, \ldots, x_n$ in the form

$$\begin{aligned} x_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1, \\ x_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2, \\ &\vdots \\ x_n &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n + b_n, \end{aligned}$$

It is shown in [1] that a solution of this system is given by

$$x_i = \left\{ \begin{array}{ccccccccc} a_{11} & \ldots & a_{1,i-1} & a_{1i} & a_{1,i+1} & \ldots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & \ldots & a_{n,i-1} & a_{ni} & a_{n,i+1} & \ldots & a_{nn} & b_n \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \end{array} \right\}, \; i = 1, \ldots, n.$$

Since *-semiring multiplication is not necessarily commutative, there is also a "right-handed" system of equations:

$$\begin{aligned} x_1 &= x_1a_{11} + x_2a_{12} + \cdots + x_na_{1n} + b_1, \\ x_2 &= x_1a_{21} + x_2a_{22} + \cdots + x_na_{2n} + b_2, \\ &\vdots \\ x_n &= x_1a_{n1} + x_2a_{n2} + \cdots + x_na_{nn} + b_n, \end{aligned}$$

A solution of this system is given by:

$$x_i = \left\{ \begin{array}{cccc} a_{11} & \ldots & a_{1n} & 0 \\ \vdots & & \vdots & \vdots \\ a_{i-1,1} & \ldots & a_{i-1,n} & 0 \\ a_{i1} & \ldots & a_{in} & 1 \\ a_{i+1,1} & \ldots & a_{i+1,n} & 0 \\ \vdots & & \vdots & \vdots \\ a_{n1} & \ldots & a_{nn} & 0 \\ b_1 & \ldots & b_n & 0 \end{array} \right\}, \; i = 1, \ldots, n.$$

Here is another property of eliminants. Let a square matrix $M$ be decomposed in four blocks

$$M = \left[ \begin{array}{cc} A & B \\ C & D \end{array} \right]$$

[3]Note the use of "a solution" here instead of "the solution". For a general *-semiring, other conditions may be needed to restrict the solution. One may, for example, seek the least solution in some sense.

subject to the condition that $A$ and $D$ are square blocks. Then we have

$$\mathrm{elim}(M) = \mathrm{elim}(D + CA^*B).$$

This is reminiscent of the well-known linear algebra relation

$$\det(M) = \det(A)\det(D - CA^{-1}B).$$

## 3 Parallel Computation Models

We assume that our computations are done in the Parallel Random Access Machine (PRAM) model (see, e.g., [9]) in which all processors have access to a globally shared memory. Depending on the processors' access mode (concurrent or exclusive) to the memory locations for read and write operations, PRAMs are said to be of the CRCW, CREW, ERCW or EREW variety. It turns out that in the algorithms to be presented here the values of some data items will be used simultaneously by several processors, but a data item will be modified, if at all, by only a single processors. Thus the processing can be done equivalently in the CRCW or CREW mode.

We further assume that a $2 \times 2$ eliminant, that is, an expression of the form $d + ca^*b$, is computed in a single unit of time.

## 4 Computation of Eliminants

The definition of eliminant given in Section 2 is by an evaluation procedure which itself can be thought of as a very simple parallel algorithm. Given an eliminant of order $n > 2$, one obtains equivalent eliminants of successively smaller orders stopping when a single quantity, the value of the eliminant, is reached. Let a given eliminant of order $n > 2$ be

$$A = \left\{ \begin{array}{ccccc} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \ldots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \end{array} \right\}.$$

$A$ has the same value as the order $n-1$ eliminant

$$B = \left\{ \begin{array}{cccc} b_{11} & b_{12} & \ldots & b_{1,n-1} \\ b_{21} & b_{22} & \ldots & b_{2,n-1} \\ \vdots & \vdots & & \vdots \\ b_{n-1,1} & b_{n-1,2} & \ldots & b_{n-1,n-1} \end{array} \right\},$$

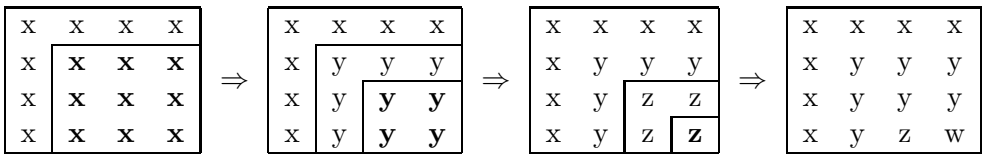
where

$$b_{ij} = \left\{ \begin{array}{cc} a_{11} & a_{1,j-1} \\ a_{i-1,1} & a_{i-1,j-1} \end{array} \right\} = a_{i-1,j-1} + a_{i-1,1}(a_{11})^* a_{1,j-1}, \text{ for} 2 \leq i, j \leq n.$$

The element $b_{ij}$ can be computed in one time unit according to our assumption. Thus $(n-1)^2$ processors can compute all elements of $B$ in a single time unit. The computation requires simultaneous access by all processors to the elements belonging to the first row or column of $A$. On the other hand, each element of B is modified by a separate processor. Thus the processing can be done equivalently in the CRCW or CREW mode.

In successive steps, eliminants of order $n-2, n-3, \ldots, 3, 2$ are generated in the same way, each requiring a single time unit by $(n-2)^2, (n-3)^2, \ldots, 9, 4$ processors, respectively. Finally, in a single time unit, a single processor replaces the last eliminant by an explicit value. Thus an $n \times n$ eliminant can be computed in parallel by $(n-1)^2$ processors in time $n-1$.

For use in the algorithms in the next two sections, the eliminant computation is performed in place as follows. The first step replaces all elements not belonging to either the first row or the first column; the second step replaces all elements not belonging to the first two rows or columns; and so on. The $(n-1)$st step replaces the bottom right corner element which then contains the eliminant's value. A schematic representation of this computation for the case $n = 4$ is shown below. The values to be replaced are shown in bold face. The values to be used but not modified are in the row and the column just above the values to be replaced, and are surrounded by single lines.




In [1], the following property of eliminants is proved. Given an $n \times n$ matrix $A$ and an integer $r$ such that $1 \leq r < n$. Let $B$ be the $(n-r) \times (n-r)$ matrix whose $(i,j)$th element is a $(r+1) \times (r+1)$ eliminant whose elements consists of the $r \times r$ principal minor of $A$ augmented with the $(r+i)$th row of $A$ and the $(r+j)$th column of $A$. In symbols,

$$b_{ij} = \begin{Bmatrix} a_{11} & \ldots & a_{1r} & a_{1,r+j} \\ \vdots & & \vdots & \vdots \\ a_{r1} & \ldots & a_{rr} & a_{r,r+j} \\ a_{r+i,1} & \ldots & a_{r+i,r} & a_{r+i,,r+j} \end{Bmatrix}.$$

Then $\mathrm{elim}(A) = \mathrm{elim}(B)$. (Note that the case $r = 1$ of this property agrees with the evaluation rule in the definition of eliminants.)

Although it appears that one might obtain a faster algorithm by using this property (since each step could reduce the eliminant order by more than one), there seems to be no improvement over the algorithm just described. To see this, let $E(n)$ be the parallel time needed to evaluate an $n \times n$ eliminant. We have assumed that a processor computes a $2 \times 2$ eliminant in one time unit, that is, $E(2) = 1$. For any $n > 2$ and any choice of $r$, we can compute all $b_{ij}$'s in parallel, and then compute the eliminant $B$ itself. $B$ is of order $n-r$ and each $b_{ij}$ is of order $r+1$. Hence we have

$$E(n) = \min_{1 \leq r \leq n-2} E(r+1) + E(n-r).$$

It turns out that the best choice for $r$ is 1 and the best value of $E(n)$ is $n-1$, with at most $(n-1)^2$ processors needed at any step.

# 5 Solution of Linear Systems of Equations

Suppose we are given the following system of $n$ equations in $n$ unknowns $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} x_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1, \\ x_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2, \\ &\vdots \\ x_n &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n + b_n. \end{aligned}$$

The solution of this equation is given by

$$x_i = \left\{ \begin{array}{ccccccccc} a_{11} & \ldots & a_{1,i-1} & a_{1i} & a_{1,i+1} & \ldots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & \ldots & a_{n,i-1} & a_{ni} & a_{n,i+1} & \ldots & a_{nn} & b_n \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \end{array} \right\}, i = 1, \ldots, n.$$

To solve the system, we need the values of $n$ eliminants of order $n+1$ each. Since each of these can be computed using $n^2$ processors in $n$ time units, their simultaneous, independent computation is possible using $n^3$ processors. Considerable reduction in the number of required processors can be achieved, however, by exploiting common subcomputations, since the $n$ eliminants have identical elements except in their last rows. Moreover, these differing last rows have a very simple structure. To compute all of them in parallel, we organize the data in a single array of $2n$ rows and $n+1$ columns, as follows:

| | | | | |
|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | $\ldots$ | $a_{1n}$ | $b_1$ |
| $a_{21}$ | $a_{22}$ | $\ldots$ | $a_{2n}$ | $b_2$ |
| $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ |
| $a_{n1}$ | $a_{n2}$ | $\ldots$ | $a_{nn}$ | $b_n$ |
| 1 | 0 | $\ldots$ | 0 | 0 |
| 0 | 1 | $\ldots$ | 0 | 0 |
| $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ |
| 0 | 0 | $\ldots$ | 1 | 0 |

This array contains all the elements of the $n$ eliminants of order $n+1$ that we want to compute. The $n$ rows in the top half of this array are common to all these eliminants, while each row in the bottom half of the array constitutes the last row of a different eliminant. The $n$ eliminants can be computed by starting with this array and successively replacing its selected elements by the values of appropriate $2 \times 2$ eliminants. The first step replaces all elements in this array except the ones belonging to the first row or the first column; the second step replaces all elements except the ones belonging to the first two rows or the first two columns; and so on. The $n$th step replaces all elements except the ones belonging to the first $n$ rows or the first $n$ columns. Thus this step replaces only the $n$ elements in the bottom half of the last column. Each of these elements is the bottom right element of a different eliminant, and attains precisely the value of this eliminant when the $n$th step is complete.

Let us denote by $m_{ij}^{(k)}$ the value of the element in row $i$ and column $j$ of this array after the $k$th step, ($m_{ij}^{(0)}$ denote the array's initial values $a$'s, $b$'s, 0's and 1's as shown in the diagram above). The

first step uses but does not modify the elements in the first row or the first column, and replaces the remaining elements, for $i = 2, \ldots, 2n,\ j = 2, \ldots, n+1$, as follows:

$$m_{ij}^{(1)} = m_{ij}^{(0)} + m_{i1}^{(0)} (m_{11}^{(0)})^* m_{1j}^{(0)}.$$

On the surface, this involves computing $(2n-1) \times n$ new values. It turns out, however, that in the lower half of the array only the first row can change, because we have $m_{i1}^{(0)} = 0$ for $i = n+2, \ldots, 2n$, so that $m_{ij}^{(1)} = m_{ij}^{(0)}$. Thus, in the first step, we need replace only the $n^2$ elements that lie in the $n$ rows and $n$ columns $2 \leq i,\ j \leq n+1$. These elements can be computed in parallel by $n^2$ processors. Each processor needs to modify a single, separate element. But all processors need simultaneous access to the values in the first row and column which are themselves not modified. Hence both CREW and CRCW modes are acceptable for this computation.

The change in the above array resulting from the first step is schematically depicted below. The values not known to be 0 or 1 are represented by the symbol $x$. The elements shown in bold face are the only ones to be replaced. The second rectangle represents the state after the first step, and does not show the elements in the first row or column of the original array since these elements do not participate in any future computation.

| | | | | |
|---|---|---|---|---|
| x | x | x | x | x |
| x | **x** | **x** | **x** | **x** |
| x | **x** | **x** | **x** | **x** |
| x | **x** | **x** | **x** | **x** |
| 1 | **0** | **0** | **0** | **0** |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |

$\Rightarrow$

| | | | |
|---|---|---|---|
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |

In the second step, we need to compute new values of the elements in the last $2n-2$ rows and $n-1$ columns, by using the rule

$$m_{ij}^{(2)} = m_{ij}^{(1)} + m_{i2}^{(1)} (m_{22}^{(1)})^* m_{2j}^{(1)},$$

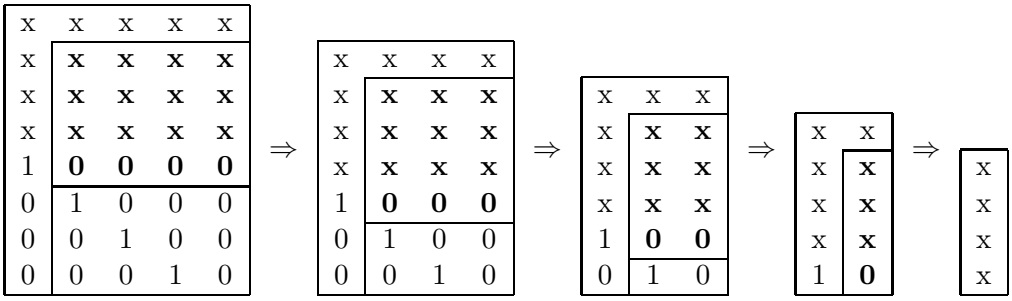
for $i = 3, \ldots, 2n,\ j = 3, \ldots, n+1$. But notice that the first step leaves the array with the pattern of 1's and 0's similar to the one before. So in the second step, the elements in the bottom $n-2$ rows retain their values, and only the $n \times (n-1)$ elements belonging to the $n$ rows 3 through $n+2$ and the last $n-1$ columns need to be replaced. These can be computed in parallel by $n \times (n-1)$ processors.

In general, the $k$th step has to compute the values

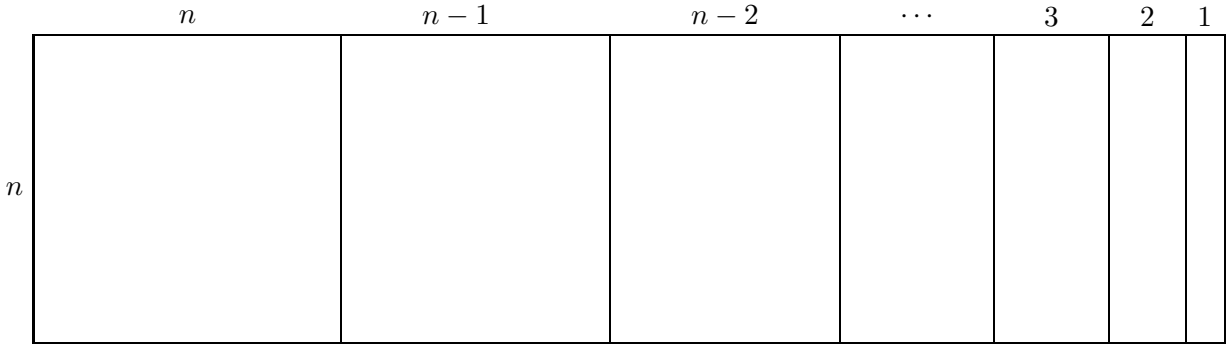
$$m_{ij}^{(k)} = m_{ij}^{(k-1)} + m_{ik}^{(k-1)} (m_{kk}^{(k-1)})^* m_{kj}^{(k-1)},$$

for $i = k+1, \ldots, 2n,\ j = k+1, \ldots, n+1$. But it follows by a simple induction that for $i = n+k+1, \ldots, 2n,\ j = k+1, \ldots, n+1$, $m_{ij}^{(k)} = m_{ij}^{(k-1)}$ (each is either 0 or 1). So the above computation need be done only for the $n$ rows $i = k+1, \ldots, n+k$ and $n-k$ columns $j = k+1, \ldots, n+1$. This can be done in parallel using $n \times (n-k)$ processors.

A schematic representation of the entire computation for the case $n = 4$ is shown below. The values not known to be 0 or 1 are represented by the symbol $x$. The elements in the top row or left column are used but not themselves modified in the current step, and are not used in succeeding steps. Hence these elements are not shown in subsequent steps. The elements shown in bold face are the only ones needed to be replaced in any step.




The entire computation, requiring successive evaluation of blocks of $n^2, (n-1)n, \ldots, 2n, n$ eliminants of order 2 for a total of $n^2(n+1)/2$ of these quantities, can be diagrammed as follows:




The quantities in each block depend on the ones in the previous one. Hence the number of steps to compute this sequence of quantities cannot be smaller than $n$. The maximum number of processors usefully engaged is $n^2$, in the first step. Later steps utilize successively fewer processors. Hence, a system of $n$ linear equations can be solved in time $n$ using $n^2$ processors. The above diagram makes it obvious how to schedule a smaller number of processors. The main constraint is that the previous blocks of evaluations be complete before getting into the next one. For example, if only $n$ processors are available, they can be scheduled to compute one column of the above diagram in each step, requiring a total of $n + (n-1) + \cdots + 2 + 1 = n(n+1)/2$ steps. In general, with $kn$ processors, for some $1 \leq k \leq n$, the solution requires between $n$ and $n(n+1)/2$ steps.

The parallel solution of the "right-handed" system of $n$ equations in $n$ unknowns is very similar, and can be obtained in time $n$ with $n^2$ processors.

# 6 Asteration or Reflexive Transitive Closure

Suppose we are given the following $n \times n$ matrix to asterate:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}.$$

Then $A^*$ is the $n \times n$ matrix $B$ whose element $b_{ij}, i = 1, \dots, n,\ j = 1, \dots, n$, is given by:

$$b_{ij} = \begin{Bmatrix} a_{11} & \dots & a_{1,i-1} & a_{1i} & a_{1,i+1} & \dots & a_{1n} & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{j-1,1} & \dots & a_{j-1,i-1} & a_{j-1,i} & a_{j-1,i+1} & \dots & a_{j-1,n} & 0 \\ a_{j,1} & \dots & a_{j,i-1} & a_{j,i} & a_{j,i+1} & \dots & a_{j,n} & 1 \\ a_{j+1,1} & \dots & a_{j+1,i-1} & a_{j+1,i} & a_{j+1,i+1} & \dots & a_{j+1,n} & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & \dots & a_{n,i-1} & a_{ni} & a_{n,i+1} & \dots & a_{nn} & 0 \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 \end{Bmatrix}.$$

To compute $A^*$, we need to compute these $n^2$ eliminants of order $n+1$ each. It is, of course, possible to compute all these eliminants in parallel in time $n$ by using a separate set of $n^2$ processors for each eliminant, requiring a total of $n^4$ processors. But these elements differ only in their last rows and columns which themsleves consist entirely of 0's and a single 1. Just as we exploited this feature of the eliminants to reduce the number of processors in solving linear systems , we will show that the asteration can also be performed in time $n$ on $n^2$ processors. For this, we combine the elements of all the eliminants into a single $2n \times 2n$ rectangular array as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | $\dots$ | $a_{1n}$ | 1 | 0 | $\dots$ | 0 |
| $a_{21}$ | $a_{22}$ | $\dots$ | $a_{2n}$ | 0 | 1 | $\dots$ | 0 |
| $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ | $\vdots$ | | $\vdots$ |
| $a_{n1}$ | $a_{n2}$ | $\dots$ | $a_{nn}$ | 0 | 0 | $\dots$ | 1 |
| 1 | 0 | $\dots$ | 0 | 0 | 0 | $\dots$ | 0 |
| 0 | 1 | $\dots$ | 0 | 0 | 0 | $\dots$ | 0 |
| $\vdots$ | $\vdots$ | | $\vdots$ | $\vdots$ | $\vdots$ | | $\vdots$ |
| 0 | 0 | $\dots$ | 1 | 0 | 0 | $\dots$ | 0 |

The upper left quarter of this array contains the first $n$ rows and columns common to all the $n^2$ eleminants. The upper right quarter contains one column from each eliminant. The lower left quarter contains one row from each eliminant. Finally, the lower right quarter contains the zeros that belong to the bottom right diagonal positions of the eliminants.

The $n^2$ eliminants are computed by starting with this array and successively replacing its selected elements by the values of appropriate $2 \times 2$ eliminants. The first step replaces all elements in this array except the ones belonging to the first row or the first column; the second step replaces all elements except the ones belonging to the first two rows or the first two columns; and so on.

The $n$th step replaces all elements except the ones belonging to the first $n$ rows or the first $n$ columns. Thus this step replaces only the $n^2$ elements in the lower right quarter of the. Each of these elements is the bottom right element of a different eliminant, and attains precisely the value of this eliminant when the $n$th step is complete. The lower right quarter of the array thus ends up with the elements of $A^*$, in the correct order of row and columns.

The style of computation is similar to the one used in the solution of linear systems, and hence will be described in less detail than in the previous section. We denote by $m_{ij}^{(k)}$ the value of the element in row $i$ and column $j$ of this array after the $k$th step. Initially, the array elements have the values given by

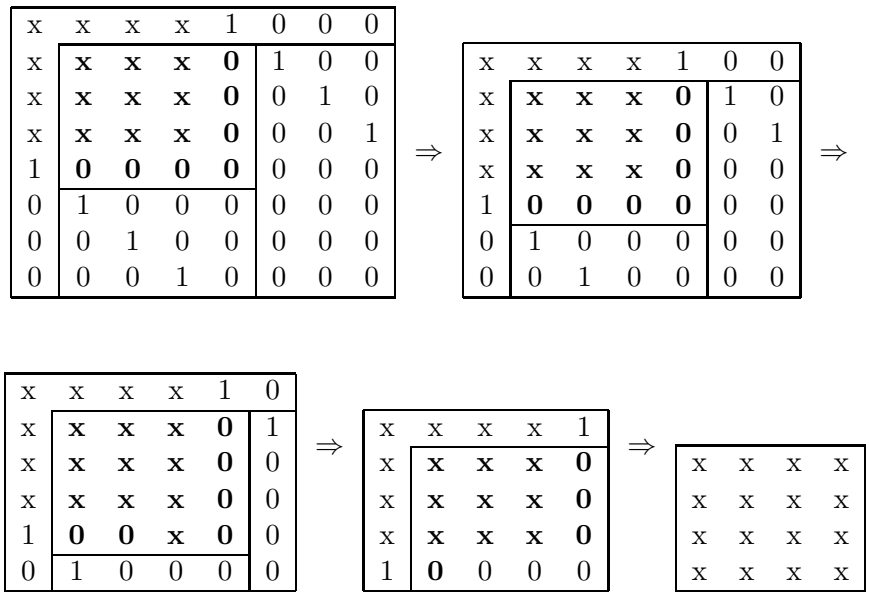
$$m_{ij}^{(0)} = \begin{cases} a_{ij}, & 1 \le i,j \le n, \\ 0, & n+1 \le i,j \le 2n, \\ \delta_{i,n-j}, & 1 \le i \le n, n+1 \le j \le 2n, \\ \delta_{n-i,j}, & n+1 \le i \le 2n, 1 \le j \le n. \end{cases}$$

as shown in the diagram above. ($\delta_{rs}$ is 1 for $r = s$, 0 otherwise.) In the $k$th step, all elements of the array not belonging to the first $k$ rows and columns are replaced as follows:

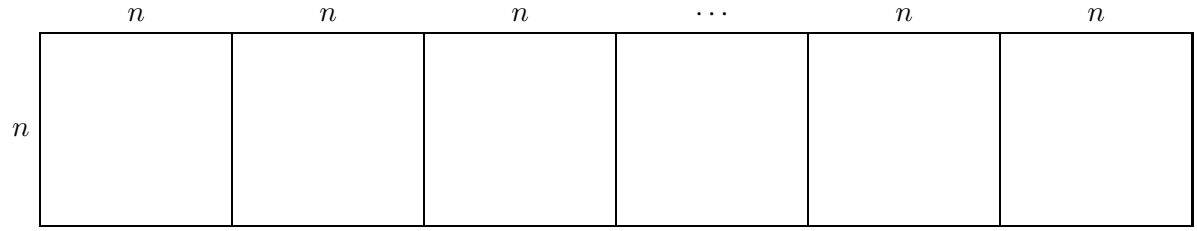
$$m_{ij}^{(k)} = m_{ij}^{(k-1)} + m_{ik}^{(k-1)} (m_{kk}^{(k-1)})^* m_{kj}^{(k-1)},$$

for $i = k+1, \ldots, 2n,\ j = k+1, \ldots, 2n$. This seems to involve the computation of $(2n-k)^2$ quantities in the $k$th step, $k = 1, \ldots, n$. But it turns out, as can be proved by a simple induction, that $m_{ij}^{(k)} = m_{ij}^{(k-1)}$ (each is either 0 or 1) for $i, j = n+k+1, \ldots, 2n$. Hence the above computation needs to be done only for $i, j = k+1, \ldots, n+k$, and can be accomplished in parallel by $n^2$ processors in unit time.

A schematic representation of the entire computation for the case $n = 4$ is shown below. The values not known to be 0 or 1 are represented by the symbol $x$. The elements in the top row or left column are used in evaluating the elements to be replaced, but are not affected by the current or subsequent steps, and are, therefore, not shown in subsequent steps. The elements shown in bold face are the only ones needed to be replaced in any step.



The computation, requiring $n$ blocks of $n^2$ eliminants of order 2 for a total of $n^3$ of these quantities, can be diagrammed as follows:




The quantities in each block depend on the ones in the previous one. Hence the number of steps to compute this sequence of quantities cannot be smaller than $n$. Using $n^2$ processors, the quantities in each block can be computed in parallel in a single step, and the whole computation can be performed in exactly $n$ steps, with no idling of processors. If the number of processors available is less than $n^2$, they have to be scheduled so that each block of computations is completed before beginning another. Within a block, of course, the computation can be performed in any order. With $n^2/r$ processors where $r$ is an integer divisor of $n^2$, the computation can be done in $rn$ steps with no idling. In general, with $kn$ processors, for some $1 \leq k \leq n$, asteration requires between $n$ and $n^2$ steps.

# 7 Conclusion

Parallel algorithms have been presented for matrix computations over *-semirings. The evaluation of an $n \times n$ eliminant, solution of linear systems of $n$ equations over *-semirings, and asteration of an $n \times n$ matrix each are done in time $n$ with $n^2$ processors under the CREW PRAM model. Karp and Ramachandran[9] call a parallel algorithm *efficient* if the time taken is a polylog function (of the problem size) and the product of the time taken and the processors used is within a polylog factor of the best sequential time algorithm for the same problem. Our algorithms are not efficient in that sense. The best sequential time for the general *-semiring asteration is $O(n^3)$. Hence an algorithm would be considered efficient if, for example, it could asterate in time $O(\log n)$ while using $O(n^3)$ processors. Of course, even the sequential algorithms known for the general *-semiring tend to be slower than the ones for "similar" problems in linear algebra and path algebras.

A difficulty is that several key properties of special *-semirings which have been exploited for devising fast algorithms are not known to hold in the general *-semiring. As an example, matrix computations can sometimes be made faster if the computed quantity can be expressed as a finite linear combination of the powers of the matrix. This is true for the reflexive transitive closure problem of Boolean matrices as well as for matrix inversion in linear algebra. Specifically, for an $n \times n$ Boolean matrix A, we know that

$$A^* = I + A + A^2 + A^3 + \cdots + A^m = (I + A)^m, \text{ for all } m \geq n - 1.$$

This linear combination be computed by matrix squaring $\log n$ times. The inverse of an $n \times n$ matrix over a field is expressible by Cayley-Hamilton theorem as

$$A^{-1} = p_0 I + p_1 A + p_2 A^2 + \cdots + p_{n-1} A^{n-1},$$

where $p$'s are related to the coefficients of the characteristic equation of $A$. This linear combination was utilized by Csanky[5] in the earliest fast parallel matrix inversion algorithm. An analog of the Cayley-Hamilton theorem or some other formula expressing matrix asterates as a finite linear combination of matrix powers is not known for the general *-semiring. It seems that more algebraic research on *-semirings is necessary before better algorithms can be expected.

## References


[1] S. K. Abdali and B. D. Saunders, "Transitive closure and related semiring properties via eliminants," *Theoretical Computer Science*, **40**, 257–274, 1985.

[2] B. A. Carré, *Graphs and Networks*, Clarendon Press, Oxford, 1979.

[3] G.-H. Chen, B.-F. Wang, and C.-J. Lu, "On the parallel computation of the algebraic path problem," *IEEE Trans. Parallel and Distributed Systems*, **3**, No. 3, 251–256, March 1992.

[4] P. Comon and Y. Robert, "A systolic array for computing $BA^{-1}$," *IEEE Trans. Accoustics, Speech, and Signal Processing*, **ASSP-35**, No. 6, 717–723, June 1987.

[5] L. Csanky, "Fast parallel matrix inversion algorithms," *SIAM J Computing*, **5**, 618–623, 1976.

[6] J.H. Conway, *Regular Algebra and Finite Machines*, Chapman & Hall, London, 1971.

[7] G. N. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications," *SIAM J Computing*, **16**, No. 6, 1004–1022, 1987.

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

[9] R. M. Karp and V. Ramachandran, "Parallel algorithms for shared-memory machines," in *Handbook of Theoretical Computer Science, Algorithms and Complexity (Volume A)*, ed. J. van Leeuwen, North Holland, Amsterdam, 869–941, 1990.

[10] A. Moffat and T. Takaoka, "An all pairs shortest path algorithm with expected time $O(n^2 \log n)$," *SIAM J Computing*, **16**, No. 6, 1023–1031, 1987.

[11] V. Pan and J. Reif, "Fast and efficient solution of path algebra problems," *JCSS*, **38**, No. 3, 494–510, June 1989.

[12] T. Risset and Y. Robert, "Synthesis of processor arrays for the algebraic path problem: unifying old results and deriving new architectures," Report 91-16, Labo. de l'info. du parallélisme, Ecole Normale Supérieure, Lyon, France, May 1991.

[13] R. E. Tarjan, "A unified approach to path problems," *JACM*, **28**, No. 3, 500–507, Oct. 1981.

[14] O. Wing and J. H. Huang, "A computation model of parallel solution of linear equations," *IEEE Trans. Computers*, **29**, No. 7, 632–638, July 1980.