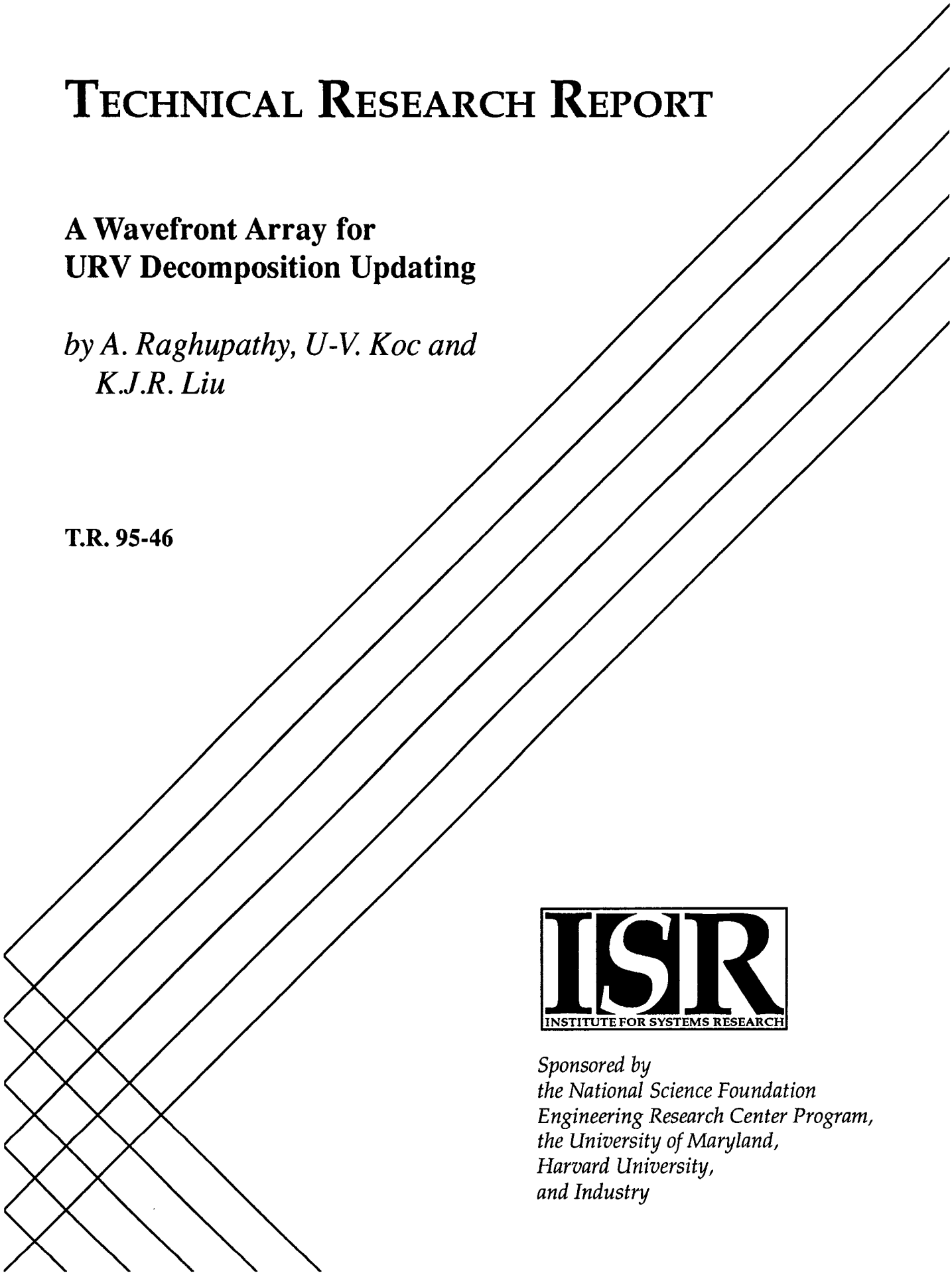
TECHNICAL RESEARCH REPORT

A Wavefront Array for URV Decomposition Updating

by A. Raghupathy, U-V. Koc and K.J.R. Liu

T.R. 95-46



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

A Wavefront Array for URV Decomposition Updating

Arun Raghupathy, Ut-Va Koc and *K. J. Ray. Liu*
Electrical Engineering Department and Institute for Systems Research
University of Maryland, College Park
EDICS No. SPL. SP. 5. 1

ABSTRACT

The rank revealing URV decomposition is an effective tool in many signal processing applications that require the computation of the noise subspace of a matrix. In this paper, we consider a parallel architecture for updating the URV decomposition on a wavefront array. The wavefront array provides an efficient real time mechanism for adaptive computation of the null space of a matrix as well as for handling rank changes during updating.

1 INTRODUCTION

Many signal processing problems require the computation of the approximate null space of the data arranged in the form of a $n \times p$ matrix A. The rows of the matrix represent the samples of a signal. For example, in applications such as antenna beamforming, spectral estimation and direction finding, the signal and/or noise subspaces need to be estimated [4]. The singular value decomposition (SVD) is the common tool used to estimate the null space of a matrix. Besides the fact that it is expensive to compute, it is also expensive to update [1] [2]. Most of the known exact updating schemes for the SVD require $O(p^3)$ operations. So it is difficult to track the null space of a matrix using SVD. Another approach is to decompose the data matrix, using a rank revealing version of the QR decomposition, into the product of an orthogonal matrix, an upper triangular matrix and a permutation matrix. A systolic array implementation of the Chan-Foster RRQR algorithm has been discussed in [3].

Recently, a new efficient numerical tool was introduced that can update the null space of a matrix when a new row of data arrives. The rank revealing URV decomposition proposed in [1] is an intermediary between the SVD and the RRQR that can be updated easily using $O(p^2)$ operations while still providing an explicit basis for the null space. The URV decomposition and the QR decomposition constitute powerful computational tools for array processing [1] [4].

*This work is supported in part by the NSF grants MIP9457397 and MIP9309506, and the ONR grant N00014-93-11028.

In this paper we discuss how the rank revealing variant of the URV decomposition can be updated on a data-driven wavefront array. The emphasis here is to develop an architecture that can implement both the URV and QR decompositions so that all computational issues related to adaptive array processing can be performed on one single parallel processing environment.

2 URV UPDATE

If the $n \times p$ data matrix A has a rank k with the singular values satisfying $\sigma_1 \geq \cdots \sigma_k > \sigma_{k+1} \geq \cdots \geq \sigma_p$ with σ_k being large compared to σ_{k+1}, then the URV decomposition of A can be written as

$$A = U \begin{pmatrix} R & F \\ 0 & G \end{pmatrix} V^T. \tag{1}$$

In the above equation, U and V are $n \times n$ and $p \times p$ orthogonal matrices respectively. R and G are upper triangular with $\inf(R) \cong \sigma_k$. Also, if ν is defined by $\nu = \sqrt{\| F \|^2 + \| G \|^2}$, then $\nu \cong \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_p^2}$. In order to distinguish the small singular values from the large ones, a tolerance *tol* must be specified. For the URV decomposition ν is compared with *tol* in order to separate the signal space and the noise space.

When a new data row of data z^T is available, it is appended to the original data matrix A as its last row to form the new data matrix. The problem of updating the URV decomposition is that of finding the decomposition of the new data matrix given the decomposition of A. Since

$$\begin{pmatrix} U^T & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} A \\ z^T \end{pmatrix} V = \begin{pmatrix} R & F \\ 0 & G \\ x^T & y^T \end{pmatrix} = \hat{A} \tag{2}$$

where $\begin{pmatrix} x^T & y^T \end{pmatrix} = z^T V$, the problem reduces to that of updating $\hat{A}$. The following steps are required to update $\hat{A}$:

Step 1. Compute $z^T V$ to obtain $\begin{pmatrix} x^T & y^T \end{pmatrix}$.

Step 2. $\hat{A}$ is updated in one of the following ways based on the relative values of *tol*, ν and $\| y \|^2$.

Case 2a) If $\sqrt{\nu + \| y \|^2} \leq tol$, we are assured the rank cannot increase and $\hat{A}$ can be triangularized by a sequence of left rotations.

Case 2b) If $\sqrt{\nu + \| y \|^2} > tol$, an increase of rank occurs. The matrix has to be transformed to an upper triangular form without destroying all the small values of F and G. This is done by interleaving left rotations with right rotations for the y^T part and left rotating the x^T part into R. In this case the V matrix must be updated because right rotations are involved.

Note that all rotations referred to in this paper correspond to Givens rotations.

3 SYSTEM COMPONENTS AND FUNCTIONS

For URV decomposition, a wavefront array is preferred over a systolic array because of the following reasons. Firstly, we need to perform interleaved left and right rotations in one of the steps. The computational wavefronts of the interleaved left and right rotations do not flow unhindered but cross each other. Also, this left rotation differs from the left rotations in the Gentleman-Kung's array for QRD [6] in that the rotation is applied to the ith and $(i+1)$th rows instead of between the ith row and the last row of the augmented data matrix. Thus we prefer a data driven approach. Secondly, a wavefront array provides flexibility for programming[5] . This means that the functions of cells can be modified easily to take into account rank changes during an update.

The operation consists of 2 modes. Mode 1 corresponds to **Case 2a**) while Mode 2 corresponds to **Case 2b**). The matrix V is stored on the linear array of processing elements (PE's) labeled V in Fig. 1 (one column on each processor). The data z^T is fed serially to the V PE's at the far left. The V PE's multiply z^T by V to produce the vectors x^T and y^T which are then transmitted upwards to the RR PE's. The V PE's also compute $\| y \|^2$. This result is used by the processor V0 at the extreme right end of the linear array of V PE's to decide between Mode 1 and Mode 2 operations for the particular update. Finally, the V PE's also accumulate the right rotations in Mode 2 by right rotating the V matrix.

The triangular part of the array stores the R, F and G matrices (Fig. 1). In mode 1 the R, G and F PE's perform left rotations while the RL PE's generate the left rotations as in the Gentleman-Kung array. In mode 2 the R PE's perform only left rotations, the G PE's perform interleaved left and right rotations and the F PE's perform only right rotations. The RL PE's generate the left rotations which are propagated along the row whereas the right rotations generated by the RR PE's are propagated upwards along a column.

For each processor we need to specify the sequence of computations that it needs to perform (see Table 1). The computation is data-driven rather than being globally synchronized on a global clock. This means that a processor starts a particular computation when the data required for it is made available by the neighbouring PE's. This requires some sort of handshaking between PE's.

The rank of the data matrix may change during an update. As a result of this the size of the R, F and G matrices can change. This means some processors have to change their function. We introduced what we refer to in this paper as rank-masking to handle this problem. Each processor stores one value termed the rank-mask that determines what functions it performs during an update. If the rank changes, then we must be able to update the rank-mask to reflect the operations that the PE's need to perform during the next update. The mode of operation is communicated to all PE's using control signals in order to facilitate this updating. The rank-mask is defined as follows. The R PE's have a mask value of 0. Let k represent the initial rank of the data matrix. The integral parts of the mask values of columns $(k+1), (k+2), \cdots, (k+(p-k))$ are $1, 2, \cdots, (p-k)$. The F

PE's have an additional fractional part of 0.5. The control pattern that updates the rank-mask when the rank increases from $k = 3$ to $k = 4$ is shown in Fig. 2. A "-1" is propagated upwards on each column that has non-zero rank-mask values and a "0" is propagated up the columns with zero rank-mask values. Also, the RL PE with rank-mask 1 sends a control signal of "0.5" along their rows while all other RL PE's with non-zero rank-mask send a "0" along their rows. At each PE, the control signals entering it from below (and possibly from the left) are summed to get the new rank-mask value. Under the additional constraint that, if the newly computed rank-mask is 0.5, then it is modified to 0, the update is completed correctly.

In mode 1, the left rotations differ from those performed by the PE's in the Gentleman-Kung systolic array for QRD in that, the new data is fed in from the diagonal elements of the array rather than from one of the other sides [4]. Each processor (F,G or R) receives data from the PE below it and passes the data upwards until it reaches the top of the array. Then the processor waits for the data from the processor above it, perform left rotations on the data and then passes the result to the PE below it. The RL PE's generate the left rotation parameters by annihilating the data obtained from processors above them and then transmit the left rotation parameters along the row.

In mode 2, the RR cells receive the result of the matrix-vector multiplication from the V PE's and then generate the right rotation parameters by annihilating *yin* in the pair (*y yin*) (See Table 1). The result is transmitted to the next adjacent RR processor through *yout*. Note that the lth column of PE's (consisting of F and G PE's) perform the right rotations corresponding to the lth and $(l+1)$th columns. Also observe that the G PE's of the ith row perform left rotation between the ith and $(i+1)$th rows whereas the R PE's perform left rotation parameters to annihilate the x vector. The RL PE's with non-zero rank-mask values generate the left rotation parameters to annihilate the non-zero sub-diagonal elements produced by the right rotation while the other RL PE's generate left rotation parameters to annihilate the x vector. The G PE's perform left and right rotations using the cosine and sine parameters generated by the RR and RL PE's while the F PE's perform only the right rotations that are generated by the RR PE's. The details are shown in Table 1.

In mode 2, the V PE's that have a non-zero rank-mask also have to accumulate the right rotations into the V matrix (i.e they have to perform right rotations similar to the F PE's). We used a linear array for doing the matrix vector multiplication as opposed to a full square array with each processor storing one element of V. This is because the bottleneck lies in the computation of $\| y \|^2$ (since it is available only after the whole matrix-vector multiplication is completed), so there is no gain in further pipelining the multiplication.

4 PERFORMANCE AND CONCLUSION

For continuous mode 1 operation, if the serial feed-in of the first row of data is started at time 0, then after carefully analyzing the various computational wavefronts we find that the next set of

data can be piped into the system at time $3p$. The $3p$ delay is required for the multiplication to be completed and then for the control signal to propagate back to the V PE's. If we allow the control to be broadcast to all RR PE's, then this delay can be reduced to $2p$ (viz. the time it takes to complete the matrix-vector multiplication).

The performance of the system under the assumption that the control is broadcast to all RR PE's as summarized in Table 2. Note that we have assumed that each operation takes unit time. Each entry in the table corresponds to the time taken for that particular step when there is no pipelining between various steps. The entry "the total time when pipelined" takes into account two kinds of pipelining, one between the steps and the other between one update and the following update. So the total time specified under mode 1 (i.e $2p-1$) indicates that if a sequence of mode 1 updates are performed then a new row can be fed in every $2p-1$ units of time. In mode 2 the corresponding time is $3p-1$ because in this case the rank update requires control propagation before the rotations can be completed.

It is also interesting to note that this implementation (with minor modifications) can have broader application than what has been mentioned in this paper. For example, this array can be used to implement the QRD-RLS although the throughput may not be as good as the Gentleman-Kung's systolic array. All we need is an additional column of PE's to the extreme right to compute the residual and some minor modifications to processor functions.

In this paper we have presented a VLSI architecture for the implementation of the URV update which involves complex data flows that cannot be easily handled by a systolic design. The solution that we proposed in the form of a wavefront array has the additional advantages of flexibility and easy programmability. Computer simulations confirmed the results presented in this paper.

References

[1] G. W. Stewart, "An Updating Algorithm for Subspace Tracking," *IEEE Transactions on Signal Processing*, vol. 40, no. 6, June 1992.

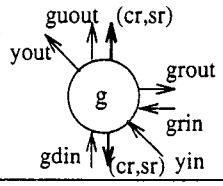
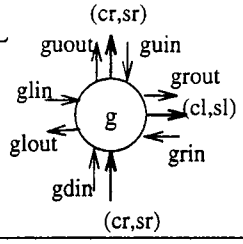
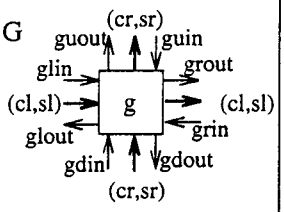
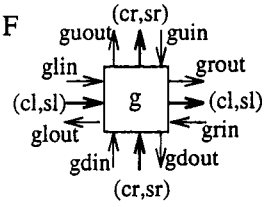
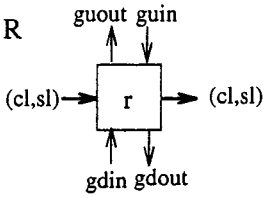
[2] K. J. R. Liu, D. P. O'Leary,G. W. Stewart and Y. -J. J. Wu, "URV ESPRIT for Tracking Time-Varying Signals," *IEEE Trans. on Signal Processing*, Vol 42, No 12, pp.3441-3448, Dec. 1994.

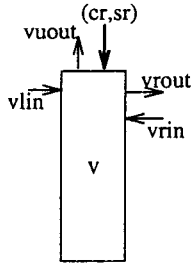
[3] F. Lorenzelli, P. C. Hansen, T. F. Chan and K. Yao, "A Systolic Algorithm of Chan/Foster RRQR Algorithm," *IEEE Transactions on Signal Processing*, vol. 42, no. 8, August 1994.

[4] Simon Haykin, *Adaptive Filter Theory*, Prentice-Hall, 1991.

[5] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall, 1988, pp. 295-302.

[6] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic array," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 298, p. 298, 1981.

Processor Type	Mode 1	Mode 2
RR 	y=gdin guout=y	if rank-mask=0 y=gdin guout=y else [yout,cr,sr]=gen-rt[y yin] [guout grout] $\overset{cr}{\underset{sr}{\leftarrow}}$ rot-rt[0 grin]
RL 	guout=gdin [g,cl,sl]=gen-lt[g guin]	if rank-mask=0 guout=gdin [g,cl,sl]=gen-lt[g guin] else [g grout] $\overset{cr}{\underset{sr}{\leftarrow}}$ rot-rt[g grin] [g,cl,sl]=gen-lt[g gdin] ;glout=g ; g=glin ;guout=g ; g=guin
G 	guout=gdin [g gdout] $\overset{cl}{\underset{sl}{\leftarrow}}$ rot-lt[r guin]	[g grout] $\overset{cr}{\underset{sr}{\leftarrow}}$ rot-rt[g grin] ;glout=g ; g=glin [g gdout] $\overset{cl}{\underset{sl}{\leftarrow}}$ rot-lt[g gdin] ;guout=g ; g=guin
F 	guout=gdin [g gdout] $\overset{cl}{\underset{sl}{\leftarrow}}$ rot-lt[r guin]	[g grout] $\overset{cr}{\underset{sr}{\leftarrow}}$ rot-rt[g grin] ;glout=g ; g=glin
R 	guout=gdin [r gdout] $\overset{cl}{\underset{sl}{\leftarrow}}$ rot-lt[r guin]	guout=gdin [r gdout] $\overset{cl}{\underset{sl}{\leftarrow}}$ rot-lt[r guin]

V 	No operation	if rank-mask=0
		No operation
		else
		for (1<=i<=p)
		[v(i) vrout] $\overset{cr}{\underset{sr}{\leftarrow}}$ rot-rt[v(i) vrin] ;vlout=v(i) ;
		v(i)=vlin ;

KEY: $[x\ z] \overset{cr}{\underset{sr}{\leftarrow}} \text{rot-rt}[x\ y] \equiv \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} cr & sr \\ -sr & cr \end{bmatrix}$

$[x,cr,sr]=\text{gen-rt}[x\ y] \equiv$ Find cr and sr such that: $\begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} cr & sr \\ -sr & cr \end{bmatrix}$

$[x\ z] \overset{cr}{\underset{sr}{\leftarrow}} \text{rot-lt}[x\ y] \equiv \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} cl & sl \\ -sl & cl \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

$[x,cl.sl]=\text{gen-lt}[x\ y] \equiv$ Find cl and sl such that: $\begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} cl & sl \\ -sl & cl \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Table 1: Cell Operations in Different Modes

Step of URV update	Time in Mode 1	Time in Mode 2
Matrix Multiplication	$2p-1$	$2p-1$
Control Propagation	p	p
Rotations	$2p-1$	$p-k$
Total time when pipelined	$2p-1$	$3p-1$

Table 2: Performance of the System

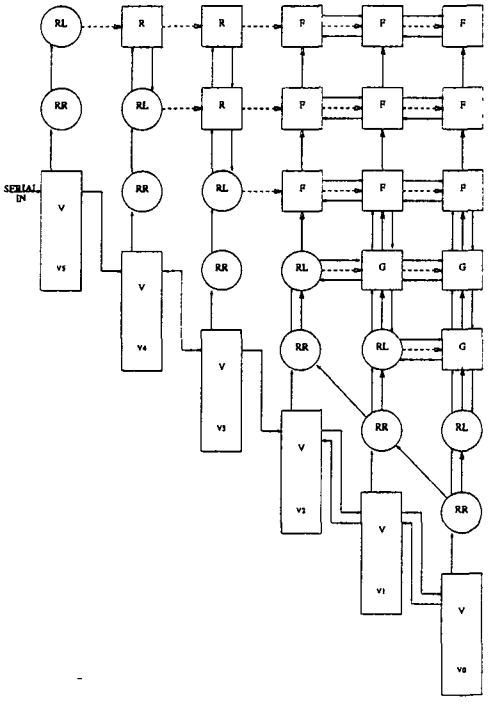


Figure 1: The wavefront array for URV update

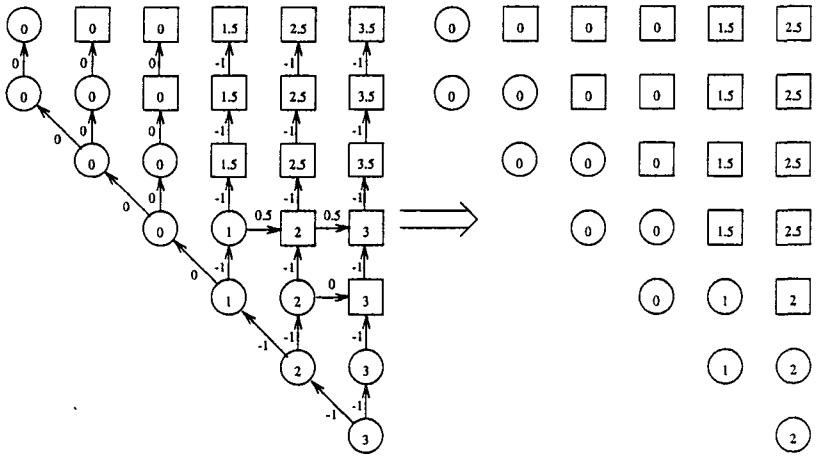


Figure 2: The Rank Update