A Wavefront Array for URV Decomposition Updating

Arun Raghupathy, Ut-Va Koc and *K. J. R. Liu*
Electrical Engineering Department and Institute for Systems Research
University of Maryland, College Park
(301) 405-0035
arunr@eng.umd.edu, koc@eng.umd.edu, kjrliu@src.umd.edu

ABSTRACT

The rank revealing URV decomposition is a useful tool in many signal processing applications that require the computation of the noise subspace of a matrix. In this paper, we look at the problem of updating the URV decomposition and consider its implementation on a wavefront array. We propose a wavefront array for efficient VLSI implementation of the URV decomposition. The array provides a mechanism for accumulating the null space of the matrix as well as for handling rank changes during updating.

I. 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. In mathematical terms,we need to find an orthogonal matrix $V = [V_1\ V_2]$ such that AV_1 has no small singular values and AV_2 is small. For example,in applications such as antenna beamforming [5], spectral estimation [6], direction finding etc., the signal and noise subspaces are crucially needed. In the well known MUSIC and ESPIRIT algorithms we need either subspace of the data matrix A [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. 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 is discussed in [2].

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. A parallel algorithm was also suggested in [1] for implementation on a linear array of processors.

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 architechture that is VLSI implementable. A triangular array is required in order to pipeline the various steps of the algorithm, as will become clear later.

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

II. URV UPDATE

If the $n \times p$ data matrix A has a rank k with the singular values satisfying

$$\sigma_1 \geq \cdots \sigma_k > \sigma_{k+1} \geq \cdots \geq \sigma_p$$

where σ_k is large compared to σ_{k+1}, then the URV decomposition of A can be written as

$$A = U \begin{pmatrix} R & F \\ 0 & G \end{pmatrix} V^H. \quad (1)$$

In the above equation

1. U and V are $n \times n$ and $p \times p$ othogonal matrices respectively.
2. R and G are upper triangular.
3. $\inf(R) \cong \sigma_k$.
4. If we define $\nu = \sqrt{\| F \|^2 + \| G \|^2}$, then $\nu \cong \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_p^2}$.

The small singular values associated with noise have to be distinguished from the large ones associated with the signal. A tolerance *tol* must be specified in order to decide what is meant by small. For a URV decomposition ν is compared with *tol* in order to separate the signal space and the noise space. Refer to details in [1] about how *tol* can be chosen.

When a new data row of data z^T is available, it is appended to the original data matrix A. Now the problem of updating the URV decomposition is that of finding the decomposition of

$$\begin{pmatrix} A \\ z^T \end{pmatrix}$$

given the decomposition of A.Now, since

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

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

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

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

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

Step 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^H part and left rotating the x^H 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. In what follows, Step 2a is referred to as mode 1 and Step 2b as mode 2.

0-7803-2570-2/95 $4.00 ©1995 IEEE

III. WAVEFRONT ARRAY APPROACH

A wavefront array is preferred over a systolic array because of the following reasons. Firstly, in mode 2 we need to perform interleaved left and right rotations. 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-RLS [8] 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. This means that the functions of cells can be modified easily to take into account rank changes during an update. Thirdly, when the computation times of different processors are unequal, the wavefront array has the best possible average pipelining rate [3].

The matrix V is stored on the linear array of processors labelled V in Fig. 1 (one column on each processor). This linear array performs three other functions. First, it performs Step 1 (viz. the matrix vector multiplication that produces x and y). Secondly, it computes $\| y \|^2$. This result is used to decide between the 2 cases of Step 2 by the processor at the extreme right end of the linear array of V processors. Once this decision is made, the mode of operation of the array is decided. Thirdly, it accumulates 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 processors perform left rotations while the RL processors generate the left rotations as in the Gentleman-Kung array. In mode 2 the R processors perform only left rotations, the G processors perform interleaved left and right rotations and the F processors perform only right rotations. The RL processors generate the left rotations which are propagated along the row whereas the right rotations generated by the RR processors are propagated upwards along a column.

Since we have a wavefront array, the sequence of operations needs to be specified for each processor. The computation is data-driven in the sense that when the data required for a particular computation is available, that computation is completed. In order to check whether a data item is ready, certain handshaking signals are required between adjacent processors. The handshaking between processors is similar to that described in [7].

The rank of the data matrix may change during an update. As a result of this, the functions that the processors need to perform during the next update may change. Since the functions of the R,F and G processors differ from one another in mode 2, a rank change may cause a processor to change its function. The problem is solved by introducing what is referred to as rank-masking. 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 this rank-mask appropriately. The same applies for the processors in the linear array of V processors too. The details will be discussed in the next section.

In our implementation, we do not consider a decrease in rank. Since a rank decrease will not occur often in practice, we assume that a check is made off-line to check whether the R matrix is defective in rank. We can check whether the matrix is defective in rank by using a condition estimator(eg. see [9]).

IV. IMPLEMENTATION DETAILS

The data z^T is fed serially to the V processors at the far left (as shown in Fig. 1). The V processors multiply z^T by V to produce the vectors x and y. The x and y vectors are transmitted upwards to the RR processors. The computation of $\| y \|^2$ is pipelined into the matrix-vector multiplication so that $\| y \|^2$ is available to the V processor labelled V0 at the far right. The V0 processor also has the values of ν and the tolerance tol available, so that it can decide on the mode of operation for the particular update. Note that the mode of operation also determines whether the rank changes.

The mode of operation is communicated to all processors in the system using control signals, as briefly mentioned before. The rank-mask is defined as follows.The R processors 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 processors 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 rank increase means that the R matrix increases in size. Also, a rank increase means the G matrix must decrease in size. 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 processors with old rank-mask 0 send a control signal of "0.5" along their rows while all RL processors with non-zero rank-mask send a "0" along their rows. At each processor, the control signals entering it from below (and possibly from its 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, we see that the update is then completed. Note that all the control signals are pipelined into the computations as much as possible to minimize delays.

In mode 1, left rotations need to be performed. The functions performed by the processors are similar to those performed by the processors in the Gentleman-Kung systolic array. The difference is that in this case the new data is fed in from the diagonal elements of the array rather than from the top. The functions of the processors are described in Table 2. Each processor (F,G or R) receives data from below and passes it upwards until it reaches the top of the array. Then the processors wait for the data from above (which is the data that was sent up after undergoing all the previous left rotations), perform rotations on it and pass the result below. The RL processors differ in that they generate the left rotation parameters by annihilating the data obtained from above.Also, they transmit the left rotation parameters along the row.

In mode 2, the operation is more involved. The RR cells receive the result of the matrix-vector multiplication from the V processors. Then they generate the right rotation by annihilating yin in the pair (y yin). The result is transmitted to the next adjacent RR processor through $yout$. Note that the lth column of processors (consisting of F and G processors) perform the right rotations corresponding to the lth and $(l+1)$th columns. Another important point to note is that the left rotation performed by the G processors in mode 2 is different from that performed by the R processors. The R processors perform left rotations to annihilate the x vector whereas the G processors of the ith row perform left rotation between the ith and $(i+1)$th rows. The RL processors with

non-zero rank-mask values generate the left rotations to annihilate the non-zero sub-diagonal element produced by the right rotation. The G processors perform left and right rotations using the cosine and sine parameters generated by the RR and RL processors. The right rotations are performed on the data contained in the processor and the data of its nearesr right neighbour. The left rotations are performed on the data that the processor contains and the data obtained from its nearest neighbour below it. The details are shown in Table 2.

In mode 1, the V processors only do multiplication.In mode 2, the V processors that have a non-zero rank-mask also have to accumulate the right rotations into the V matrix. This is achieved by allowing the V processors to perform the same operations as the F processors. We chose a linear array for storing V as opposed to a full square array with each processor storing one element of V for the following reason. Using a square array would produce no speed-up because the bottleneck lies in the computation of $\| y \|^2$ (since it is available only after the whole matrix-vector multiplication is complete). There is no gain in further pipelining the multiplication.

For continuous mode 1 operation, if the serial feed-in of the first row of data is started at time 0, then after carefully analysing 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 processors. If we allow one control broadcast line connecting all RR processors (this is permitted because clock skew is not a significant factor in wavefront array design), then this delay can be reduced to $2p$(viz. the time it takes to complete the matrix-vector multiplication).

V. PERFORMANCE AND CONCLUSION

The performance of the system is summarized in table 1. Note that we have assumed that each operation takes unit time and the broadcast line mentioned in the last section is used. Each entry in the table correspond to the time taken for each step when there is no pipeling 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.

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 1: Performance of the System

It is also interesting to note that this implementation (with minor modifications) can have broader application than 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 processors to the extreme right to compute the residual and some minor modifications to processor functions.

In this paper we have presented a VLSI architechture 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.

REFERENCES

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

[2] 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.

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

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

[5] J. G. McWhirter and T. J. Shepherd, "An Efficient Systolic Array for MVDR Beamforming," *Proc. Int. Conf. Systolic Array*, 1988, pp. 11-20.

[6] S. M. Kay, *Modern Spectral Estimation*, Englewood Cliffs, NJ : Prentice-Hall, 1988.

[7] S. Y. Kung, K. S. Arun, Ron. J. Gal-ezer and D. V. Bhaskar Rao, "Wavefront Array Processor: Language, Architecture and Applications," *IEEE Transactions on Computers*, Vol. C-31, No. 11, Nov. 1982.

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

[9] A. K. Kline, C. B. Moler, G. W. Stewart and J. H. Wilkinson, "An Estimate fo the Condition Number of a Matrix," *Siam Journal Numerical Analysis*, Vol. 16, No. 2, April 1979.

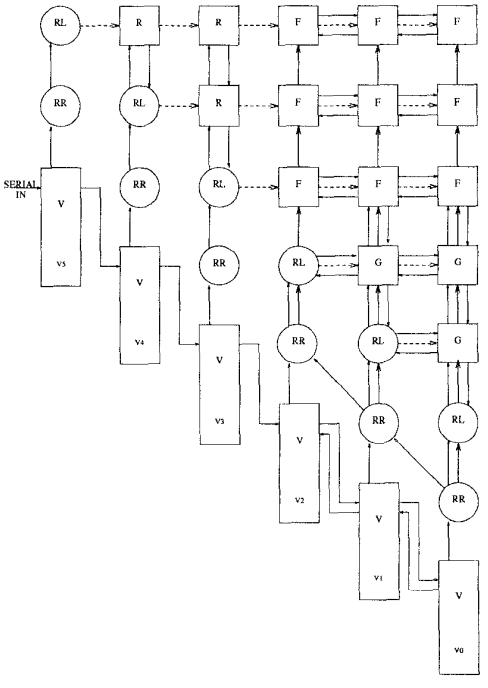
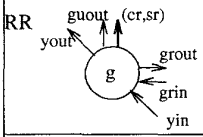
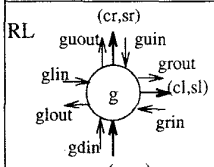
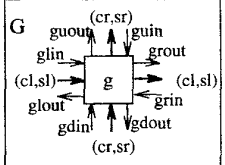
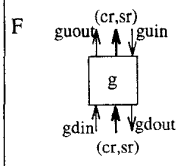
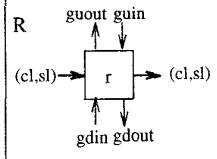
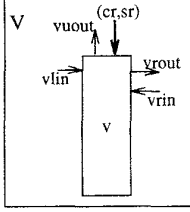


Figure 1: The wavefront array for URV update

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

$[x\ z] \overset{cr}{\underset{sr}{\leftarrow}} \text{rot-lt}[x\ y] \equiv \begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} cl & sl \\ -sl & cl \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$ $\quad [x,cl.sl]=\text{gen-lt}[x\ y] \equiv$ 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 2: Cell Operations in Different Modes

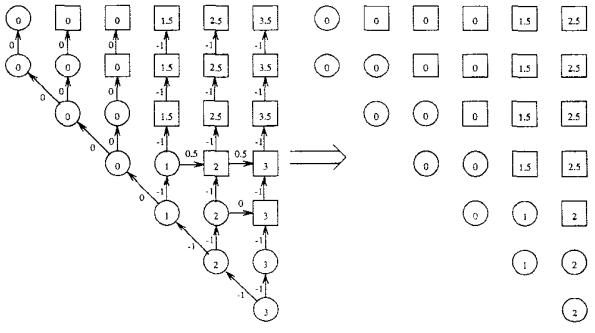


Figure 2: The Rank Update