VLSI ALGORITHMS FOR DOOLITTLE'S, CROUT'S, AND CHOLESKY'S METHODS

Lennart Johnsson
California Institute of Technology
Pasadena, California 91125

TM 5030

this paper will appear in
Proc. IEEE International Conference on
Circuits and Computers, ICCC 82
New York, September 29 - October 1, 1982

VLSI ALGORITHMS FOR DOOLITTLE'S, CROUT'S AND CHOLESKY'S METHODS

Lennart Johnsson

Computer Science
California Institute of Technology
Pasadena, CA 91125

ABSTRACT

In order to take full advantage of the emerging VLSI technology it is required to recognize its limited communication capability and structure algorithms accordingly. In this paper concurrent algorithms for the methods of Crout, Doolittle and Cholesky are described and compared with concurrent algorithms for Gauss', Given's and Householder's method. The effect of pipelining the computations in two dimensional arrays is given special attention.

INTRODUCTION

The need to solve linear systems of equations is frequent in engineering and science. The need occurs not only in traditional calculations but also in the area of signal processing. The interest in improved performance for engineering calculations is most significant for large problems while in signal processing applications the problem size typically is orders of magnitude less but the need to perform computations in real time demands a concurrent algorithm that can be conveniently implemented in hardware.

A linear system of equations is often represented by

$$Ax = y$$

where A is a matrix and x and y vectors. It is often the case that the matrix A has structural properties that can be used advantageously to reduce computations as well as the amount of hardware required. In the following the matrix A is assumed to have band structure. Algorithms developed for band matrices are, in general, also efficient for full matrices given the communication limitations imposed by VLSI technology.

By studying band matrices insight is gained into the relationship between problem characteristics and the organization of data flow and computations in a network of processing elements. Techniques for general sparse matrices have been developed extensively for sequential machines during the last decade. To devise efficient concurrent algorithms for a more general sparse matrix than a band matrix is a very complex task that is not treated here.

Many direct methods for the solution of linear systems of equations can be considered as consisting of two phases, one phase where the system matrix is factored into one upper triangular and one lower triangular matrix including forward substitution, and one phase for the back substitution. Here only the triangulation operation for Cholesky's, Crout's and Doolittle's methods[1] will be considered. The forward and back substitution can be performed in the same way for Gauss', Givens', Doolittle's, Crout's and Cholesky's methods. For concurrent algorithms and computational arrays for the forward and back substitution see e.g., Kung[2], Kung[3], Johnsson[4].

CONCURRENT ALGORITHMS

In Doolittle's, Crout's and Cholsky's methods, see e.g., Dahlquist[1], rows and columns are computed interchangably. In Doolittle's method the computations start with the first row, in Crout's with the first column. The system matrix A is assumed to be positive definite. Cholesky's method is a specialization to the symmetric case where the factors also are symmetrized, $A = LL^T$.

Figure 1 shows the order of computation for Doolittle's method. Operations that can be performed concurrently have the same number.

1	1	1	1	1	1	1	1	1	1
2	3	3	3	3	3	3	3	3	3
2	4	5	5	5	5	5	5	5	5
2	4	6	7	7	7	7	7	7	7
2	4	6	8	9	9	9	9	9	9
2	4	6	8	10	11	11	11	11	11
2	4	6	8	10	12	13	13	13	13
2	4	6	8	10	12	14	15	15	15
2	4	6	8	10	12	14	16	17	17
2	4	6	8	10	12	14	16	18	19

Figure 1: Computation order of Doolittle's method

The computational order for Crout's method is obtained by interchanging an odd number with the following even number and an even number with the preceeding odd number (or rows with columns).

In the study of concurrent algorithms it is often useful to define a computational window,[5] which displays the set of nontrivial computations that can be performed concurrently. The computational window for a band matrix is marked in boldface in Figure 2. The window moves down along the diagonal as the computations progress.

The upper left corner contains processed data that are not further used in the triangulation process. The lower right submatrix consists of unprocessed data. In the computation of the bottom row of the upper right triangle the upper row of the left triangle is rotated 90 degrees clockwise around the junction point between the two triangles and distributed to the columns in the upper right triangle. In the computation of the rightmost column of the lower triangle, the first column of the upper triangle is rotated 90 degrees counter clockwise around the junction point and distributed to the rows of the left triangle. It should be noticed that the top of the upper triangle in Figure 2 only performs trivial computations and can be discarded. In general, there will be min(r,s) x max(r,s) active elements.

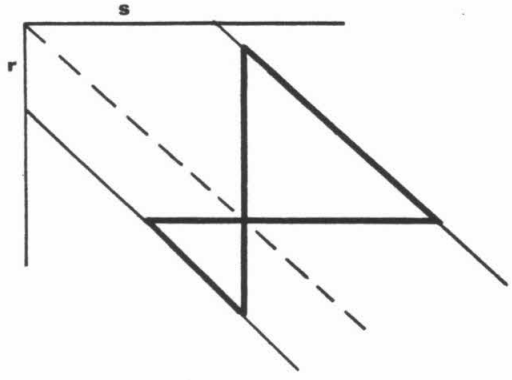


Figure 2: Computational window for Doolittle's, Crout's and Cholesky's methods

The computational window moves down the diagonal or conversely, the data passes through an array of computational elements. The matrix data flows in a direction parallel to the hypotenuse of the triangles. Hence, there are three main flow directions, precisely as in Gaussian elimination[2,3,4] and Given's rotations[6,7,8]. Figure 3 shows the directions for the upper triangle.

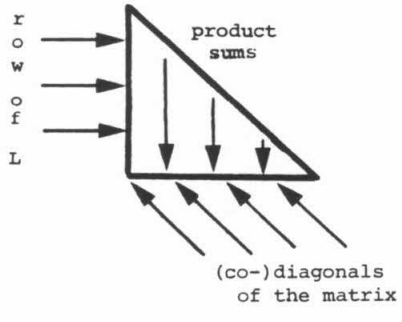


Figure 3: Stream lines of data flow for Doolittle's, Crout's and Cholesky's methods

The two triangular arrays defined by the computational window, Figure 2, can be arranged in many different ways. One is to place them side by side, another to interleave them after rotation and mirroring of one triangle, Figure 4.

The computed data that are not needed in the computations any longer may leave at the junction between the two triangular arrays, or U may leave at the hypotenuse of the L array and L at the hypotenuse of the U array. Note that the column of

Figure 4: Two alternative arrangements of processing elements

U that is needed for the computation of the next column of L can be passed directly into the L part of the array by passing it to its left neighbor and conversely the row of L that is needed in the computation of the next row of U can be passed directly to the U part of the array by passing it to the nearest neighbor to the right. This property is of particular interest in a pipelined arrangement.

The cells on the boundary of the arrays are somewhat different from the interior cells, except for the cells on the hypotenuse which are the same as the interior cells. The interior cells in the U triangle pass a value from left to right, unchanged, a value from the lower right to the upper left corner, also unchanged, and computes the product between these two values and adds it to an input value on the cell's upper side and puts the result out on the lower side. A pipelined version of an interior cell is shown in Figure 5. A non-pipelined version is obtained by omitting the intermediate shift register cells, [Z], on the horizontal and vertical paths.

The interior cells of the L part of the array can be obtained by mirroring the interior cells in the U part of the array.

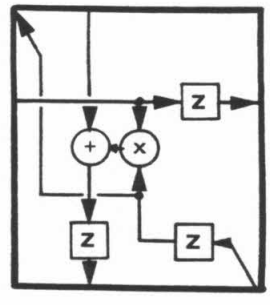


Figure 5: An interior cell, I, for U and after mirroring also for L in Doolittle's, Crout's and Cholesky's methods

The boundary cells are slightly different in the L and U parts and depends on the method chosen. The L boundary cells in Doolittle's method are similar to the U boundary cells in Crout's method and vice versa. Cells for the boundary along which data enters the array are shown in Figure 6. The cell in Figure 6b has a path through it for the diagonal elements of U in Doolittle's method and the diagonal elements of L in Crout's method. This extra path only serves the purpose of feeding the diagonal element to the edge of the array. The delay elements have the effect that it will be available at the same time as the other elements in that row for L and that column for U.

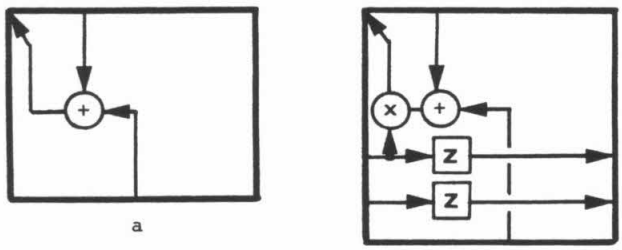


Figure 6: a) A boundary cell, A, for U in Doolittle's method and after mirroring also for L in Crout's method
b) A boundary cell, B, for U in Crout's method and after mirroring also for L in Doolittle's method

The cell in the lower left corner for U in Doolittle's method and the lower right corner for L in Crout's method has to compute an inverse of the result of the addition and will hence be slightly different from the cells in Figure 6 as shown in Figure 7a.

On the left boundary of the U part and the right boundary of the L part a special cell is required. Assuming the factors are collected at the ends of the array a joining cell as in Figure 7b can be used for Doolittle's and Crout's methods.

The algorithm to be described for Cholesky's method computes L^T. For Cholesky's method it is necessary to include in the array the column corresponding to the diagonal element. This column is not considered in Crout's method since $u_{kk} = 1$. Correspondingly, the row corresponding to the diagonal element is not included in Doolittle's method since $l_{kk} = 1$. The boundary cells on the lower boundary do not need the feed through capability of the cells in Figure 6b. Hence, they need only have the properties shown in Figure 8a.

The corner cell for Cholesky's method has to perform a square root computation and an inversion, Figure 8b. Instead of a cell joining the U and L arrays as in Figure 7b a reflection cell as in Figure 8c is required for Cholesky's method.

The cells in Figures 5-7 together with the connection plan in Figure 9a defines concurrent algorithms for Doolittle's and Crout's methods. The cells in Figures 5, and 8 and the connection plan in Figure 9b defines a concurrent algorithm for Cholesky's method.

The logic required for initialization is not included in the figures above. The feedback has to be broken to allow for correct initialization. In Figure 9a the I and B cells in the left triangle denotes the mirrored version of the cells. These algorithms can also be described in a formal way, e.g., by using the notation in Cohen[9], Johnsson[10].

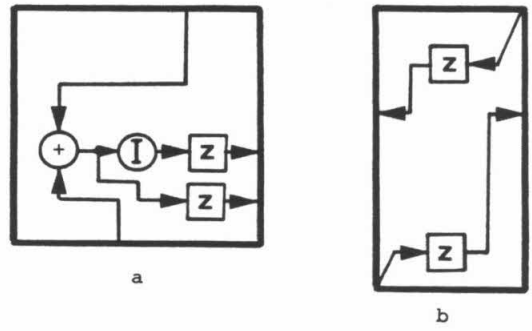


Figure 7: a) Corner cell, C, for L in Crout's method and after mirroring for U in Doolittle's method

b) A cell, J, joining the L and U parts for Doolittle's method and after mirroring also for Crout's method

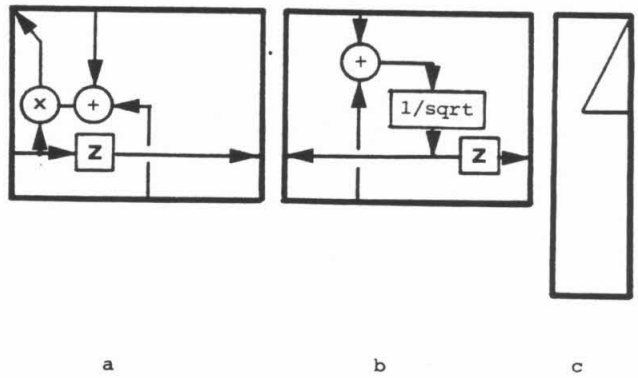
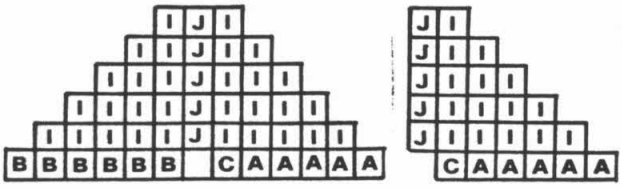


Figure 8: a) Boundary cell, B, for Cholesky's method
b) Corner cell, C, for Cholesky's method
c) A left boundary cell, J, for Cholesky's method



a b

Figure 9: a) An array for Doolittle's method and after mirroring along the J cells also for Crout's method
b) An array for Cholesky's method

TEMPORAL CONSIDERATIONS

Pipelining

In the above description of concurrent algorithms pipelining has been used to avoid broadcasting in the horizontal direction. This pipelining implies that the computations of successive elements in a row of U and a column of L is initiated and completed at successive cycles.

The algorithms are also pipelined in the vertical direction. Through this pipelining several cycles are required for the computation of each element in U and L. The advantage is that a cycle in a synchronous implementation only has to be long enough to accommodate one multiply-add operation. Without the vertical pipelining a cycle would have to be long enough to accommodate one multiplication and the additions in a chain of min(r,s) adders.

There is an additional kind of pipelining assumed in the algorithms above. The computations of different rows of U and columns of L are pipelined. Furthermore, if more than one matrix shall be triangulated only the first matrix requires an initialization period, since the initialization of subsequent matrices can be performed concurrently with the computations on the preceeding matrix.

Synchronization

Pipelining the computations in the horizontal direction requires successive input data streams, codiagonals, to be delayed by one cycle, skewed with respect to each other. The vertical pipelining in combination with the horizontal pipelining and the delays along the diagonal path requires the data within each data stream to be spread out in time so that new data are accepted only every third cycle in any data stream.

The synchronization between computations can be illustrated by wave fronts. A wave front can be associated with a column of U or a row of L. For the pipelining scheme implied by the cells above the wave fronts and stream lines are shown in Figure 10. The distance between successive wave fronts corresponds to three cycles or cells.

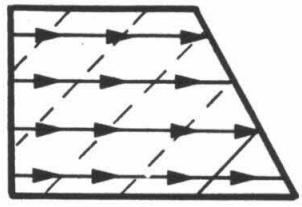


Figure 10: Stream lines and wave fronts

A wave front can also be defined as the set of elements being computed concurrently. Defined in this way it is natural to picture a wave front in relation to the original matrix. The wave fronts will in this case be in the same relative position as shown in Figure 10.

CONCLUSIONS

The concepts of a computational window and data streams are useful in finding the communication requirements in computational networks. The communication required by the concurrent algorithms for Doolittle's, Crout's and Cholesky's methods that are described here is the same as the communication required by many concurrent algorithms for Gaussian elimination without pivoting and Given's method.

The complexity of operations that a node performs is, for most nodes, limited to one addition and one multiplication. Up to three registers are used for temporary storage. Some additional logic is required for the initialization phase. The complexity of a cell is independent of the problem size. The problem size is reflected in the size of the array used. If the problem is of a size larger than what the array can hold, external storage and a block oriented algorithm can be used. By modifying the array slightly it is possible to have the array perform the different kind of operations required for the different steps of a block algorithm, see e.g. Johnsson[4]. Another solution is to equip the internal nodes with additional storage and explicit control. Suitable algorithms and data structures for such an architecture can be found by folding the in space fully instantiated algorithms or arrays onto themselves.

In a synchronous design as used to illustrate the algorithms here a cycle is limited by a multiply-add or divide and add operation in the pipelined alternatives. For Cholesky's method the limiting factor for the cycle time is the computation of the inverse square root. A fast algorithm such as proposed by Chen[11] or Walther[12] can be used as a remedy. A concurrent algorithm based on hyperbolic rotations for Cholesky's method has been described in Ahmed et al[8].

Pipeling used as a technique to avoid broadcasting in computational arrays affects not only the synchronization between different data streams but

also the separation in time between elements in one and the same stream.

The pipelined algorithms presented here computes the factorization of a N by N matrix in O(3*N) steps, where a step is determined by the time required for a multiply-add operation. Hence, the number of steps is equal to the number of steps required in Gaussian elimination without pivoting and Given's method. The number of cells is essentially the same for Gauss', Doolittle's, Crout's and Given's method. Cholesky's method that exploits the symmetry of a matrix uses half as many cells as the other methods. The complexity of cells is essentially the same except for Given's method that requires cells of a higher complexity. Householder's method can only use a pipelined linear array efficiently and will hence require more steps, (r+s+3)*N, for the factorization.

ACKNOWLEDGEMENT

The author gratefully acknowledges the support provided by the Defense Advanced Research Project Agency, order 3771, and monitored by the Office of Naval Research under contract N00014-79-C-0597. Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion of DARPA, the U.S. Government, nor any person or agency connected with them.

REFERENCES

1. Dahlquist G., Bjorck A., Anderson N., Numerical Methods, Prentice-Hall, 1974.

2. Kung, H.T. and Leiserson, Charles E., "Algorithms for VLSI Processor Arrays," Introduction to VLSI Systems, Addison-Wesley, 1980, pp. 271-294, Mead, Carver A. and Conway, Lynn A.

3. Kung, S.Y., "VLSI Matrix Computation Array Processor," MIT Conference on Advanced Research in Integrated Circuits, MIT, February 1980, pp. .

4. Johnsson, L., "Computational Arrays for Band Matrix Equations," Tech. report 4287, Caltech Computer Science Department, May 1981.

5. Johnsson L., "A Computational Array for the QR-method," Proceeedings, Conferences on Advanced Research in VLSI, Artech House, January 1982, pp. 123-129.

6. Gentleman, S. Morven, Kung H. T., "Matrix Triangulation by Systolic Arrays," Tech. report, Carnegie-Mellon University, 1981.

7. Johnsson S. L., "Pipelined Linear Equation Solvers and VLSI," Microelectronics '82, Institution of Electrical Engineers, Australia, May 1982, pp. 42-46.

8. Ahmed H.M., Delsome J.-M., Morf M., "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," IEEE Computer, No. 1, January 1982, pp. 65-82.

9. Cohen D., "Mathematical Approach to Iterative Computational Networks," Proceedings of the Fourth Symposium on Computer Arithmetic, IEEE Computer Society, October 1978, pp. 226-238.

10. Johnsson, S. Lennart, Weiser, U., Cohen, D., and Davis, A., "Towards a Formal Treatment of VLSI Arrays," Proceedings of the Second Caltech Conference on VLSI, Caltech Computer Science Department, January 1981, pp. .

11. Chen T.C., "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots," IBM J. Research and Development, Vol. 16, No. 4, July 1972, pp. 380-388.

12. Walther J.S., "A Unified Algorithm for Elementary Functions," Spring Joint Computer Conference, IEEE Computer Society, 1971, pp. 379-385.