TOEPLITZ SOLVERS AND VECTOR PROCESSING

J.-M. Delosme [1], S .C. Eisenstat [2] , J.R. Massé [3]

1 Department of Electrical Engineering, Yale University, New Haven, Connecticut
2 Department of Computer Science, Yale University, New Haven, Connecticut
3 Centre de Calcul Scientifique de l'Armement, 35170 Bruz

Résumé

Alors que la transformée de Fourier rapide est de pratique courante sur les calculateurs vectoriels mono- ou multi-processeurs la situation est toute différente pour la résolution de systèmes de Toeplitz, problème de base du traitement du signal paramétrique. Dans cet article, deux réécritures de l'algorithme de Levinson sont proposées pour permettre d'utiliser deux ou quatre processeurs en parallèle et le récent algorithme de Delosme et Ipsen, conçu pour les réseaux systoliques, est aussi adapté pour le calcul vectoriel. Les performances expérimentales sur Cray-1S et Alliant FX/8 plaident en faveur du nouvel algorithme.

Abstract

While fundamental non-parametric signal processing algorithms, such as the FFT, are available on vector mono- or multi-processors, little attention has been devoted to basic problems in parametric signal processing, such as the solution of Toeplitz systems. To remedy this situation, the classical Levinson algorithm is reformulated to exploit vectorization and two- and four-processor concurrency. Furthermore the Toeplitz solver of Delosme and Ipsen, conceived with systolic array implementations in mind, is also adapted for vector processing. Performance measures on Cray-1S and Alliant FX/8 favor the new algorithm.

I - INTRODUCTION

Supercomputers, such as the Cray, CDC, Fujitsu, Hitachi and NEC systems, offer the best performance among general purpose computers. They employ multiple functional units with vector instructions [1]. The Cray-1 has a unique arithmetic pipelined unit and, in Cray Fortran, vectorization is handled through DO loops [2].

While supercomputers are the fastest, the highest speed/cost ratio is achieved by the minisupercomputers, a mid-range between supercomputers and superminicomputers. Minisupercomputers include Floating Point, Star Technologies, Convex Computer and Elxsi systems. The Alliant FX/8, a multiprocessor with up to 8 vector units, belongs to that class [3]. On FX/8, the FX/Fortran programming language offers both vectorization and concurrency through Fortran DO loops, as naturally as Cray Fortran does for vectorization only.

Cray Fortran implementations of non-parametric signal processing basics are available through several papers and subroutine libraries, e.g. [4],[5]. Surprisingly, little work has been done on vector implementations of Toeplitz system solvers, a basic topic for parametric signal processing [6]. In fact an early implementation of the original Levinson algorithm on Cray-1 [2] can easily be improved upon; moreover exploitation of both vectorization and concurrency has never been studied. In this paper the solution of Toeplitz systems is considered from the double perspective of vector and concurrent processing. The computations are analyzed, implemented and tested on both FX/8 and Cray-1S.

II - FEATURES OF THE ALLIANT FX/8

While descriptions of the Cray-1 series hardware and features are easily found in the literature [2], this is not yet the case for the FX/8 and we shall therefore briefly discuss its features.

The Alliant FX/8 [3] has up to 8 computational elements (CEs) able to work concurrently. Each CE contains a vector integer and floating point unit operating on vector data of length 32 in 32- or 64-bit precision, 8 floating point and 8 vector register sets for scalar and vector operations, and a concurrency control unit and bus. The CEs share a common global memory. A fast memory subsystem, including up to two computational processor caches, transmits data between CEs and memory.

FX/Fortran is the main programming language on the FX/8. Concurrency and vectorization are controlled through Fortran DO loops. The simplest way to generate concurrency and vectorization with a single DO loop is the vector-concurrent mode in which each CE takes some of the iterations and executes them with vector operations. Where DO loops are nested, FX/Fortran runs the innermost in vector mode and the next outer loop in concurrent mode thus running these loops in concurrent-outer-vector-inner (COVI) mode. The other modes, which do not include both vectorization and concurrency, are the vector mode, scalar concurrent mode and scalar mode. Vector, scalar concurrent and vector concurrent mode are good for vector lengths greater than 16, $3 \cdot p$ and $16 \cdot p$ respectively, where p is the number of CEs. The modes may result from the context in the Fortran code or may be explicitly specified with *optimization directives* such as CVD NOCONCUR or CVD NOVECTOR. These directives allow to override the compiler when optimization does not enhance performance. For instance, with very short vector lengths, optimization of a DO loop may slow down execution.

III - TOEPLITZ SOLVERS

III.1 - Levinson recursions

An $n \times n$ symmetric Toeplitz matrix, T, has identical elements along its diagonals:

$$T_{i,j} = t_{|i-j|+1} \quad \text{for } 1 \le i,j \le n$$

Levinson's original algorithm [7] solves linear systems $T \cdot x = y$ with Toeplitz coefficient matrices and arbitrary right hand sides. The algorithm calls for $n-1$ iterations where iteration m essentially consists of 4 dot products and 2 vector-multiply-and-add of length m or $m+1$ (Fig. 1). Petersen has implemented that algorithm on the Cray-1 [2].

In signal processing, an improved version [8] is typically used in which R_m is computed recursively using a single scalar multiply-and-add at each iteration:

$$R_{m+1} = R_m - \rho_m \cdot \Delta_m$$

In signal processing terminology, the ρ_m are the reflection coefficients and the R_m are the prediction error variances. The recursion may be embedded in the ρ_m vector-multiply-and-add operation (Fig. 2). Now iteration m requires only 2 dot products and 2 vector-multiply-and-adds, hence implementing these recursions would already improve on Petersen's work. Unfortunately from the vector or concurrent processing perspective, the 4 vector operations are dependent on each other. Furthermore since n is typically small (less than 30) in signal processing applications (although some geophysical applications require orders up to 1,000), it is unlikely that the vectors will be cut into pieces in some vector concurrent mode. Yet one can exploit the independence between the last two vector operations of iteration m and the first two of iteration $m+1$ and embed vectors x and C in a two-dimensional array S:

$$\begin{bmatrix} S^m_{2,0} \\ \vdots \\ S^m_{m+2,0} \end{bmatrix} = \begin{bmatrix} x^m_1 \\ \vdots \\ x^m_{m+1} \end{bmatrix}, \quad \begin{bmatrix} S^m_{1,1} \\ \vdots \\ S^m_{m+3,2} \end{bmatrix} = \begin{bmatrix} C^m_{m+3} \\ \vdots \\ C^m_1 \end{bmatrix}$$

Now, at each iteration, the 4 vector operations may be done concurrently 2 by 2 (Fig. 3).

[1] Work supported by the Army Research Office under Contract DAAL03-86-K-0158

A careful look at two consecutive steps brings more concurrency to the fore. The first vectorial part of iteration $m+1$,

$$\bar{S}^{m+2}_{m+4,j} = \begin{pmatrix} S^{m+1}_{2,j} \\ \vdots \\ S^{m+1}_{m+3,j} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+3} \\ \vdots \\ t_2 \end{pmatrix} = \begin{pmatrix} S^{m+1}_{2,j} \\ \vdots \\ S^{m+1}_{m+2,j} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+2} \\ \vdots \\ t_3 \end{pmatrix} + S^{m+1}_{m+3,j} \cdot t_2 \ ,$$

combined with the second vectorial part of iteration m,

$$\begin{pmatrix} S^{m+1}_{2,j} \\ \vdots \\ S^{m+1}_{m+2,j} \end{pmatrix} = \begin{pmatrix} S^{m}_{2,j} \\ \vdots \\ S^{m}_{m+2,j} \end{pmatrix} - S^{m+1}_{m+3,j} \cdot \begin{pmatrix} S^{m}_{m+2,1} \\ \vdots \\ S^{m}_{2,1} \end{pmatrix} \ ,$$

gives $\bar{S}^{m+2}_{m+4,j} = \alpha_j^m - S^{m+1}_{m+3,j} \cdot \beta^m + S^{m+1}_{m+3,j} \cdot t_2$, where

$$\alpha_j^m = \begin{pmatrix} S^{m}_{2,j} \\ \vdots \\ S^{m}_{m+2,j} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+3} \\ \vdots \\ t_3 \end{pmatrix} \quad \text{and} \quad \beta^m = \begin{pmatrix} S^{m}_{m+2,1} \\ \vdots \\ S^{m}_{2,1} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+3} \\ \vdots \\ t_3 \end{pmatrix} \ .$$

Furthermore

$$\beta^m = S^m_{m+2,1} \cdot t_{m+3} + \begin{pmatrix} S^{m}_{m+1,1} \\ \vdots \\ S^{m}_{2,1} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+2} \\ \vdots \\ t_3 \end{pmatrix}$$

combined with the second part of iteration $m-1$, gives

$$\beta^m = S^m_{m+2,1} t_{m+3} + \begin{pmatrix} S^{m-1}_{m+1,1} \\ \vdots \\ S^{m-1}_{2,1} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+2} \\ \vdots \\ t_3 \end{pmatrix} - S^m_{m+2,1} \begin{pmatrix} S^{m-1}_{2,1} \\ \vdots \\ S^{m-1}_{m+1,1} \end{pmatrix}^T \cdot \begin{pmatrix} t_{m+2} \\ \vdots \\ t_3 \end{pmatrix} .$$

Thus β^m may be calculated recursively via a single scalar recursion:

$$\beta^m = \beta^{m-1} + S^m_{m+2,1} \cdot \left(t_{m+3} - \alpha_1^{m-1} \right) \ .$$

A new writing of the Levinson recursions is thus found in which the four vector operations may be done concurrently (Fig. 4). The price paid is just an extra scalar recursion.

III.2 - Hyperbolic Cholesky solver

Delosme and Ipsen presented their unnormalized Toeplitz hyperbolic Cholesky solver (THCS) in a systolic perspective [9]. A vector decomposition is proposed here.

Figure 5 lists the equations of THCS, using the same notations as in III.1. Once the reflection coefficients, ρ_j, are calculated, there is no dependency between Step 1 and Step 2. Since the two steps have the same unnormalized hyperbolic rotations, $\begin{pmatrix} 1 & -\rho_j \\ -\rho_j & 1 \end{pmatrix}$, they may be concatenated into common vector operations (Fig. 6). Iteration m has two vector-multiply-and-add operations of length $2 \cdot (n-m)$ forming a *butterfly* like in the FFT. Step 3 may be embedded in a similar structure with vectors of length $n-m$ (Fig. 7).

Albeit the vector lengths are typically small (see III.1), it is possible with THCS to take advantage of longer vectors for problems with several right hand sides by simply concatenating the different y's with the factorization of T (see Fig. 6-7). This is the vector translation of the pipelining proposed for systolic implementations [9].

IV - PROGRAMMING

We shall now discuss implementational issues both on the vector multiprocessor Alliant FX/8 and on the Cray-1S.

IV.1 - COVI mode

As seen in Section III, the 2×2 and 1×4 versions of the Levinson recursions have, unlike the usual version, potential for both vectorization and exploitation of concurrency. The natural way to express concurrency and vectorization with FX/Fortran is the so-called COVI mode [3]. Loops DO 1 and DO 2 of TLSA (Fig. 8) express concurrency. The vector operations, written here explicitly in extended FX/Fortran, are within these loops. In routine TSA, the analogue of TLSA for the 1×4 version, the MIMD feature of the 1×4 version disabled recognition of the COVI mode as such by the compiler. To get around this problem, the two basic vector computations were included in subroutines called concurrently with the CONCALL optimization directive. With new versions of the compiler this stratagem will probably no longer be needed.

IV.2 - Vector-concurrent mode

The COVI mode, a good choice when the vectors are small, loses its appeal for large vectors since it does not necessarily allow all the CEs to be active. In fact, routines TLSA or TSA, which are 2×2 or 1×4 concurrent, activate potentially only 2 or 4 of the up to 8 CEs. This leads to the use of the alternative vector-concurrent mode[3]. In this mode, instead of exploiting the above 2×2 or 1×4 concurrency, each of the four vector operations is dispatched to *all* the CEs. At some critical iteration ($m=31$ from experiments), the driver routine switches from COVI mode (routine TLSA (Fig. 8) or TSA) to vector-concurrent mode (routine TLSB or TSB).

IV.3 - Load savings

When only one CE is available, the vector-concurrent mode reduces to the vector mode; this is the natural mode on the Cray-1S. Interestingly the concurrent writings of the Levinson recursions are still the most efficient. This is because in these writings the right hand sides are identical by pairs in independent computations thus saving some load operations to vector registers. Subroutine TLSB runs this way under vector and vector-concurrent modes. This is impossible with the usual version of the Levinson recursions due to the dependencies. Such savings hold also with the left hand side vectors in the 1×4 computation. On Cray-1 this improvement is hidden by all the loads and stores needed for the dot products.

IV.4 - Butterfly computations

An even more elegant use of registers may be reached with the THCS algorithm. In both basic substeps the vector computations are composed of vector-multiply and adds forming butterfly operations like in the FFT (Fig. 6 and Fig. 7). In each of the two butterflies, the reflection coefficient and the two right hand side vectors are used twice. They are kept in scalar and vector registers during the butterfly processing. On Cray-1, the load and store operations alternate with the multiply or adds so as to produce some pipelining. Furthermore, as has been done for the FFT[10], the butterfly structure allows taking advantage of longer vectors by concatenating L computations. This corresponds exactly to the pipelining charts found in a systolic perspective [9]. Since in the butterfly scheme the reflection coefficients must be identical for the L system solutions, the concatenation is limited to L different right hand sides with the same coefficient matrix. To do so y_i and subsequently z_{ij}, y_{ij}, f_{ij}, g_{ij} and x_i are expanded from scalars to L-component subvectors.

V - COMPARED BEHAVIOR

As discussed in Section IV, THCS consumes more floating point operations, $O(3n^2)$ flops, than the Levinson recursions, $O(2n^2)$, but allows better megaflop rates. On both FX/8 and Cray-1S the rates turn out to be roughly proportional with a 3:2 factor (Fig. 9). Competitive speed performances are thus observed, with a slight advantage for THCS clearly visible on the Cray-1S (Fig. 10).

Little difference is observed between the 2×2 and 1×4 writings of the Levinson recursions. As mentioned previously the 1×4 writing of the Levinson recursions should be improved under new versions of FX/Fortran. The great superiority of THCS comes from its ability of concatenating efficiently L system solutions (Fig. 11). Solution of a 9th order system with $L=5$ right hand sides is as efficient as the solution of a system of order 18.

REFERENCES

[1] Kuck, D.J., *The Structure of Computers and Computations*, John Wiley & Sons, New York, 1978.

[2] Petersen, W.P., "Vector Fortran for Numerical Problems on Cray-1," *Comm. ACM*, Vol. 26, No. 11, pp. 1008-21, 1983.

[3] *ALLIANT FX/Series Product Summary*, Alliant Computer Systems Corp., June 1985.

[4] Swarztrauber, P.N., "Vectorizing the FFTs," in *Parallel Computations*, G. Rodrigue ed., Academic Press, New York, 1982.

[5] Temperton, C., "Fast Fourier Transforms on Cray-1," *European Ctr. for Medium Range Weather Forecasts*, T.R. 21, Jan. 1979.

[6] Makhoul, J., "Linear Prediction : a Tutorial Review," *Proc. IEEE*, Vol. 63, No. 4, pp. 561-80, April 1975.

[7] Levinson, N., "The Wiener RMS Error Criterion in Filter Design and Prediction," *J. Math. Phy.*, Vol. XXV, No. 4, pp. 261-78, Jan. 1947.

[8] Robinson, E., and S. Treitel, "Principles of Digital Wiener Filtering," *Geophysical Prospecting*, Vol. 15, 1967.

[9] Delosme, J.-M., and I.C.F. Ipsen, "Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations," *Linear Algebra and Applications*, Vol. 77, pp. 75-111, May 1986.

[10] Fornberg, B., "A Vector Implementation of the Fast Fourier Transform Algorithm," Dept. of Applied Mathematics, California Institute of Technology, Pasadena, California.

Initialization:

$R_0 = t_1$, $\Delta_0 = t_2$, $\rho_0 = \Delta_0 / R_0$, $c_1^0 = \rho_0$, $x_1^0 = y_1 / R_0$

Iteration m: $\quad 1 \le m \le n-2$

$$\Delta_m = t_{m+2} - \begin{bmatrix} c_1^{m-1} \\ \vdots \\ c_m^{m-1} \end{bmatrix}^T \cdot \begin{bmatrix} t_2 \\ \vdots \\ t_{m+1} \end{bmatrix}, \quad R_m = t_1 - \begin{bmatrix} c_1^{m-1} \\ \vdots \\ c_m^{m-1} \end{bmatrix}^T \cdot \begin{bmatrix} t_{m+1} \\ \vdots \\ t_2 \end{bmatrix}$$

$$\rho_m = \Delta_m / R_m \ , \quad c_1^m = \rho_m$$

$$\begin{bmatrix} c_2^m \\ \vdots \\ c_{m+1}^m \end{bmatrix} = \begin{bmatrix} c_1^{m-1} \\ \vdots \\ c_m^{m-1} \end{bmatrix} - \rho_m \cdot \begin{bmatrix} c_m^{m-1} \\ \vdots \\ c_1^{m-1} \end{bmatrix}$$

$$0 \le m \le n-2$$

$$\bar{x}_{m+2}^{m+1} = y_{m+2} - \begin{bmatrix} x_1^m \\ \vdots \\ x_{m+1}^m \end{bmatrix}^T \cdot \begin{bmatrix} t_{m+2} \\ \vdots \\ t_2 \end{bmatrix}, \quad R_{m+1} = t_1 - \begin{bmatrix} c_1^m \\ \vdots \\ c_{m+1}^m \end{bmatrix}^T \cdot \begin{bmatrix} t_{m+2} \\ \vdots \\ t_2 \end{bmatrix}$$

$$x_{m+2}^{m+1} = \bar{x}_{m+2}^{m+1} / R_{m+1}$$

$$\begin{bmatrix} x_1^{m+1} \\ \vdots \\ x_{m+1}^{m+1} \end{bmatrix} = \begin{bmatrix} x_1^m \\ \vdots \\ x_{m+1}^m \end{bmatrix} - x_{m+2}^{m+1} \cdot \begin{bmatrix} c_1^m \\ \vdots \\ c_{m+1}^m \end{bmatrix}$$

Solution: $\begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix}^T = \begin{pmatrix} x_1^{n-1} & \cdots & x_n^{n-1} \end{pmatrix}^T$

Figure 1. Original Levinson recursions.

Initialization:

$R_0 = t_1$, $\Delta_0 = t_2$, $\rho_0 = \Delta_0 / R_0$, $C_2^0 = \rho_0$, $x_1^0 = y_1 / R_0$, $C_3^0 = R_0 - \rho_0 \cdot \Delta_0$

Iteration m: $\quad 1 \le m \le n-2$

$$\Delta_m = t_{m+2} - \begin{bmatrix} C_2^{m-1} \\ \vdots \\ C_{m+1}^{m-1} \end{bmatrix}^T \cdot \begin{bmatrix} t_2 \\ \vdots \\ t_{m+1} \end{bmatrix}, \quad R_m = C_{m+2}^{m+1}$$

$$\rho_m = \Delta_m / R_m \ , \quad C_2^m = \rho_m \ , \quad C_1^{m-1} = \Delta_m$$

$$\begin{bmatrix} C_3^m \\ \vdots \\ C_{m+3}^m \end{bmatrix} = \begin{bmatrix} C_2^{m-1} \\ \vdots \\ C_{m+2}^{m-1} \end{bmatrix} - \rho_m \cdot \begin{bmatrix} C_{m+1}^{m-1} \\ \vdots \\ C_1^{m-1} \end{bmatrix}$$

$$0 \le m \le n-2$$

$$\bar{x}_{m+2}^{m+1} = y_{m+2} - \begin{bmatrix} x_1^m \\ \vdots \\ x_{m+1}^m \end{bmatrix}^T \cdot \begin{bmatrix} t_{m+2} \\ \vdots \\ t_2 \end{bmatrix}, \quad R_{m+1} = C_{m+3}^m$$

$$x_{m+2}^{m+1} = \bar{x}_{m+2}^{m+1} / R_{m+1}$$

$$\begin{bmatrix} x_1^{m+1} \\ \vdots \\ x_{m+1}^{m+1} \end{bmatrix} = \begin{bmatrix} x_1^m \\ \vdots \\ x_{m+1}^m \end{bmatrix} - x_{m+2}^{m+1} \cdot \begin{bmatrix} C_2^m \\ \vdots \\ C_{m+2}^m \end{bmatrix}$$

Solution: $\begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix}^T = \begin{pmatrix} x_1^{n-1} & \cdots & x_n^{n-1} \end{pmatrix}^T$

Figure 2. Usual writing of the Levinson recursions.

Initialization:

$R_0 = t_1$, $\Delta_0 = t_2$, $\rho_0 = \Delta_0 / R_0$, $S_{2,1}^0 = \rho_0$, $S_{2,0}^0 = y_1 / R_0$, $S_{1,1}^0 = R_0 - \rho_0 \cdot \Delta_0$

Iteration m: $\quad 0 \le m \le n-3$

$$\bar{S}_{m+3,j}^{m+1} = \begin{bmatrix} S_{2,j}^m \\ \vdots \\ S_{m+2,j}^m \end{bmatrix}^T \cdot \begin{bmatrix} t_{m+2} \\ \vdots \\ t_2 \end{bmatrix} \quad \Big\{ \text{concur. } j = 0,1$$

$$R_{m+1} = S_{1,1}^m \ , \ S_{m+3,0}^{m+1} = \left(y_{m+2} - \bar{S}_{m+3,0}^{m+1} \right) / R_{m+1} \ , \ \Delta_{m+1} = t_{m+3} - \bar{S}_{m+3,1}^{m+1}$$

$$\rho_{m+1} = \Delta_{m+1} / R_{m+1} \ , \ S_{m+3,1}^{m+1} = \rho_{m+1} \ , \ S_{m+3,1}^m = \Delta_{m+1}$$

$$\begin{bmatrix} S_{1,j}^{m+1} \\ \vdots \\ S_{m+2,j}^{m+1} \end{bmatrix} = \begin{bmatrix} S_{1,j}^m \\ \vdots \\ S_{m+2,j}^m \end{bmatrix} - S_{m+3,j}^{m+1} \cdot \begin{bmatrix} S_{m+3,1}^m \\ \vdots \\ S_{2,1}^m \end{bmatrix} \quad \Big\{ \text{concur. } j = 0,1$$

Last iteration: $\quad (\ m = n-2\ )$

$$x_n = y_n - \begin{bmatrix} S_{2,0}^{n-2} \\ \vdots \\ S_{n,0}^{n-2} \end{bmatrix}^T \cdot \begin{bmatrix} t_n \\ \vdots \\ t_2 \end{bmatrix} / S_{1,1}^{n-2} \ , \quad \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} S_{2,0}^{n-2} \\ \vdots \\ S_{n,0}^{n-2} \end{bmatrix} - x_n \cdot \begin{bmatrix} S_{n,1}^{n-2} \\ \vdots \\ S_{2,2}^{n-2} \end{bmatrix}$$

Note: $S_{1,0}^{m+1}$ is of no use but makes the two vector-multiply-and-add operations identical.

Figure 3. 2 x 2 concurrent Levinson recursions.

Initialization:

$R_0 = t_1$, $\Delta_0 = t_2$, $\rho_0 = \Delta_0 / R_0$, $S_{2,1}^0 = \rho_0$, $S_{2,0}^0 = y_1 / R_0$, $S_{1,1}^0 = R_0 - \rho_0 \cdot \Delta_0$

$\bar{S}_{3,j}^1 = S_{2,j}^0 \cdot t_2 \quad j = 1,2 \ , \ \alpha_1^{-1} = 0 \ , \ \beta^{-1} = 0$

Iteration m: $\quad 0 \le m \le n-3$

$R_{m+1} = S_{1,1}^m \ , \ S_{m+3,0}^{m+1} = \left(y_{m+2} - \bar{S}_{m+3,0}^{m+1} \right) / R_{m+1} \ , \ \Delta_{m+1} = t_{m+3} - \bar{S}_{m+3,1}^{m+1}$

$\rho_{m+1} = \Delta_{m+1} / R_{m+1} \ , \ S_{m+3,1}^{m+1} = \rho_{m+1} \ , \ S_{m+3,1}^m = \Delta_{m+1}$

$$\begin{cases} \begin{bmatrix} S_{1,j}^{m+1} \\ \vdots \\ S_{m+2,j}^{m+1} \end{bmatrix} = \begin{bmatrix} S_{1,j}^m \\ \vdots \\ S_{m+2,j}^m \end{bmatrix} - S_{m+3,j}^{m+1} \cdot \begin{bmatrix} S_{m+3,1}^m \\ \vdots \\ S_{2,1}^m \end{bmatrix} & \Big\{ \text{concur. } j = 0,1 \\ \text{concur.} \\ \alpha_j^m = \begin{bmatrix} S_{2,j}^m \\ \vdots \\ S_{m+2,j}^m \end{bmatrix}^T \cdot \begin{bmatrix} t_{m+3} \\ \vdots \\ t_2 \end{bmatrix} & \Big\{ \text{concur. } j = 0,1 \end{cases}$$

$$\beta^m = \beta^{m-1} + S_{m+2,1}^m \cdot \left(t_{m+3} - \alpha_1^{m-1} \right) \ , \quad \bar{S}_{m+4,j}^{m+2} = \alpha_j^m - S_{m+3,j}^{m+1} \cdot \left(\beta^m - t_2 \right)$$

Last iteration: $\quad (\ m = n-2\ )$

$$x_n = \left(y_n - \bar{S}_{n+1,0}^{n-1} \right) / S_{1,1}^{n-2} \ , \quad \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} S_{2,0}^{n-2} \\ \vdots \\ S_{n,0}^{n-2} \end{bmatrix} - x_n \cdot \begin{bmatrix} S_{n,1}^{n-2} \\ \vdots \\ S_{2,2}^{n-2} \end{bmatrix}$$

Note: $\bar{S}_{n+1,1}$ hence α_1^{n-3} may be dropped

Figure 4. New 1 x 4 concurrent Levinson recursions.

Step 1:

$1 \le i \le n$, $r_{i,0} = t_i$

$2 \le i \le n$, $s_{i,0} = t_i$

$1 \le j \le n-1$, $\rho_j = s_{j+1,j-1} / r_{j,j-1}$

$j+1 \le i \le n$, $r_{i,j} = r_{i-1,j-1} - \rho_{j_{i,j-1}}$

$s_{i,j} = -\rho_j r_{i-1,j-1} + s_{i,j-1}$

Step 2:

$1 \le i \le n$, $y_{i,0} = y_i$, $z_{i,0} = y_i$

$1 \le j \le n-1$,

$j+1 \le i \le n$, $z_{i,j} = z_{i-1,j-1} - \rho_j y_{i,j-1}$

$y_{i,j} = -\rho_j z_{i-1,j-1} + y_{i,j-1}$

Step 3:

$1 \le j \le n$, $f_{j,n-j} = y_{j,j-1} / r_{j,j-1}$, $g_{n+1,n-j} = 0$

$1 \le j \le n-1$,

$n-j+1 \le i \le n$, $f_{i,j-1} = f_{i,j-1} - \rho_{n-j} g_{i+1,j-1}$

$g_{i,j} = -\rho_{n-j} f_{i,j-1} + g_{i+1,j-1}$

$1 \le i \le n$, $x_i = f_{i,n-1} + g_{i+1,n-1}$

Figure 5. THCS equations.

Φ_1^m :

t_1	$f_{1,4}$		$f_{1,4}$		$f_{1,4}$
y_1	ρ_1	$f_{2,3}$	ρ_1	$f_{2,3}$	ρ_1
t_2	$r_{2,1}$	ρ_2	$f_{3,2}$	ρ_2	$f_{3,2}$
y_2	$z_{2,1}$		ρ_3	$f_{4,1}$	ρ_3
t_3	$r_{3,1}$	$r_{3,2}$		ρ_4	$f_{5,0}$
y_3	$z_{3,1}$	$z_{3,2}$			
t_4	$r_{4,1}$	$r_{4,2}$	$r_{4,3}$		
y_4	$z_{4,1}$	$z_{4,2}$	$z_{4,3}$		
t_5	$r_{5,1}$	$r_{5,2}$	$r_{5,3}$	$r_{5,4}$	
y_5	$z_{5,1}$	$z_{5,2}$	$z_{5,3}$	$z_{5,4}$	

Φ_2^m :

t_1					
y_1					
t_2	$s_{2,1}$				
y_2	$y_{2,1}$				
t_3	$s_{3,1}$	$s_{3,2}$			
y_3	$y_{3,1}$	$y_{3,2}$			
t_4	$s_{4,1}$	$s_{4,2}$	$s_{4,3}$		
y_4	$y_{4,1}$	$y_{4,2}$	$y_{4,3}$		
t_5	$s_{5,1}$	$s_{5,2}$	$s_{5,3}$	$s_{5,4}$	
y_5	$y_{5,1}$	$y_{5,2}$	$y_{5,3}$	$y_{5,4}$	

$m=0$	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$

$$\Phi_1^m(2m+1:2n) = \Phi_1^{m-1}(2m-1:2n-2) - \rho_m \Phi_2^{m-1}(2m+1:2n)$$
$$\Phi_2^m(2m+1:2n) = \Phi_2^{m-1}(2m+1:2n) - \rho_m \Phi_1^{m-1}(2m-1:2n-2)$$

Figure 6. THCS Steps 1 & 2 (for $n = 5$).

Φ_1^m :

$(f_{1,4})$	.	$(f_{1,4})$	.	$(f_{1,4})$
(ρ_1)	$(f_{2,3})$	(ρ_1)	$(f_{2,3})$	$(\rho_1)\, f_{2,4}$
$(f_{3,2})$	(ρ_2)	$(f_{3,2})$	$(\rho_2)\, f_{3,3}$	$f_{3,4}$
(ρ_3)	$(f_{4,1})$	$(\rho_3)\, f_{4,2}$	$f_{4,3}$	$f_{4,4}$
$(f_{5,0})$	$(\rho_4)\, f_{5,1}$	$f_{5,2}$	$f_{5,3}$	$f_{5,4}$

Φ_2^m :

0	$g_{5,1}$	$g_{4,2}$	$g_{3,3}$	$g_{2,4}$	z_1
.	0	$g_{5,2}$	$g_{4,3}$	$g_{3,4}$	z_2
.	.	0	$g_{5,3}$	$g_{4,4}$	z_3
.	.	.	0	$g_{5,4}$	z_4
.	.	.	.	0	z_5

$m=6$	$m=5$	$m=4$	$m=3$	$m=2$
	$k=1$	$k=2$	$k=3$	$k=4$

$$\Phi_1^m(m:n) = \Phi_1^{m+1}(m:n) - \rho_{m-1}\Phi_2^{m+1}(1:k)$$
$$\Phi_2^m(1:k) = \Phi_2^{m+1}(1:k) - \rho_{m-1}\Phi_1^{m+1}(m:n)$$
$$X(1:n) = \Phi_1^2(1:n) + \Phi_2^2(1:n)$$

Figure 7. THCS Step 3 (for $n = 5$).

```
      SUBROUTINE TLSA(SF,SI,Y,T)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON /TLSCO/ N,M
      DIMENSION SF(N,0:1), SI(N,0:1), Y(1), T(1)
C     ITERATIONS OF TLS
      M2 = M+2
      M3 = M+3
      DO 1 J = 0,1
         SF(M3,J) = DOTPRODUCT(SI(2:M2,J),T(M2:2:-1))
1     CONTINUE
      R = SI(1,1)
      SF(M3,0) = (Y(M2) - SF(M3,0)) / R
      DELTA = T(M3) - SF(M3,1)
      RHO = DELTA / R
      SF(M3,1) = RHO
      SI(M3,1) = DELTA
      DO 2 J = 0,1
         SF(1:M2,J) = SI(1:M2,J) - SF(M3,J) * SI(M3:2:-1,1)
2     CONTINUE
      RETURN
      END
```

Figure 8. 2×2 concurrent Levinson recursions
Basic step written for COVI mode.

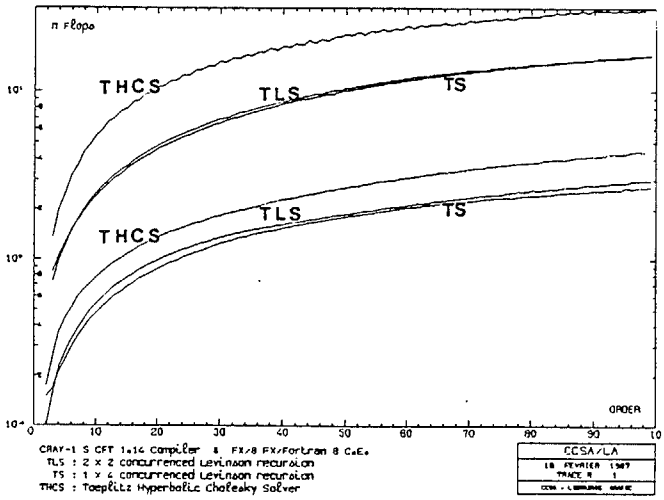

Figure 9.

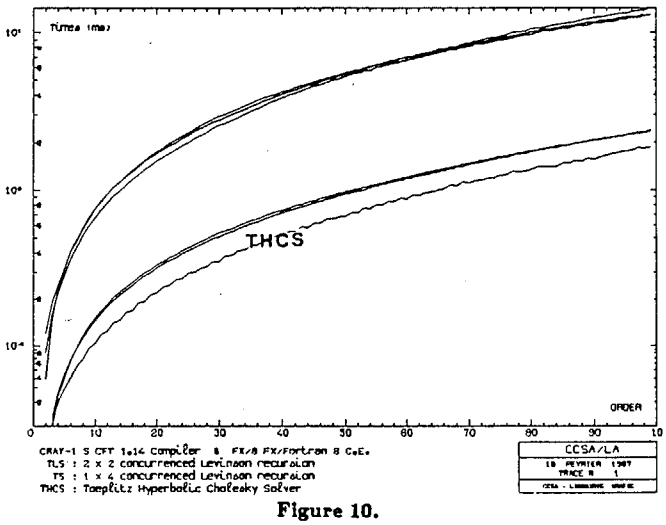

Figure 10.

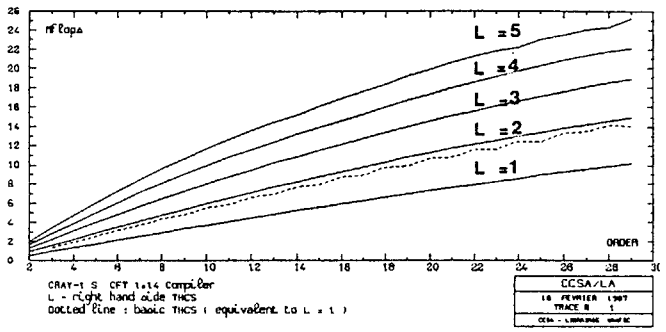

Figure 11.