Optimal Module Distribution for Pipeline Digital Filter Analysis Algorithm

Marija Kacarska and Dragan Andonov

Faculty of Electrical Engineering, Skopje, R Macedonia

The development of general techniques for the analyses of digital filters with arbitrary topology is an area of interest in the process of digital filter design, especially in the educational area. A few analyses with a large number of frequency points are required until the desired response is achieved. Therefore, it is necessary to provide fast analysis algorithms. The pipeline implementation of the Crout's algorithm enables parallelized execution of the digital filter analysis. The process distribution must be optimized in order to achieve faster analysis execution and balanced processor performance. This paper presents a program package with a generalized approach to optimize the process distribution, based on an algorithm for element combinations for a set of size L into all subsets of size m arranged in lexicographic order. Two optimization criteria are used: the number of processors and their utilization. To avoid variable execution times on different processors, the number of operations executed at each processor is taken as a measure of processor execution time. Obtained results indicate that optimal distribution is achieved using smaller number of well balanced processors. With a larger N processor efficiency rises to more than 95%.

Keywords: digital filter analysis, pipeline architecture, process distribution

1. Introduction

A digital filter is defined as a network of interconnected elements for signal delay, signal summation and signal multiplication. A suitable representation for such a network is its signal-flow graph. The signal-flow graph consists of a set of nodes and a set of directed branches showing the flow of the signals through the network. It is also a graphical representation of the existing relationships among the network elements (Crochiere et al., 1975). The characteristic of digital filters is the linearity of these relationships. Figure 1 represents a typical digital filter signal-flow graph (a) and the existing interelement relationships (b). Due to the linearity of element relationship, mathematical representation of the digital filter is a set of linear equations. Signals of the digital filter are discrete in time, with a time step T. The signal-flow graph with N nodes is completely described with a set of N linear equations.

An area of special interest for digital filter theory application is the development of general techniques for the analyses of networks with arbitrary topology. This demand is a consequence of various network configurations that can be used for digital filter implementation, as well as of various characteristics offered by these networks. In this paper we propose a program package that provides fast analysis of digital filters with arbitrary topology, without limit in their size (including wave digital filters).

General aim of the filter analysis is to determine the frequency response of the network. A single frequency response requires a solution of the system of N linear equations, using some algorithm such as Gauss or Crout solvers (Chua et al., 1975, Kacarska, 1988). In order to obtain a complete frequency response, the system solver must be applied for up to 500 separate frequencies within the range from 0 to π radians (Thede,1996).

Chapter 2 presents a matrix representation of the system of N linear equations, suitable for computer implementation. The solution based on Crout's method is particularly interesting due to its parallelization potential. This algorithm is presented in Chapter 3, where the distinction between its two basic processes is outlined. The paralellization potential of the algorithm is

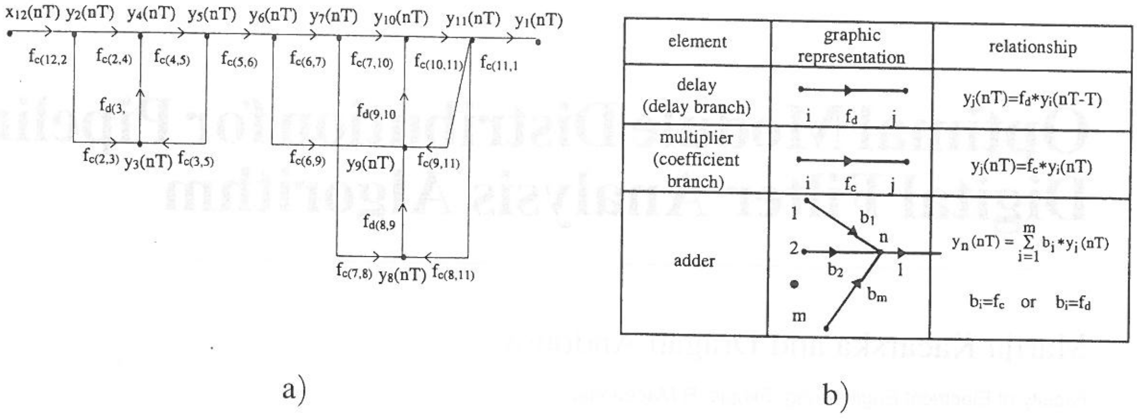


Fig. 1. (a) Principal signal-flow graph of a digital filter and (b) its relationship representation.

explored in Chapter 4 and a pipelined parallel implementation of the algorithm is described in Chapter 5.

Parallel implementation is optimized with respect of the execution time and processor efficiency, as described in Chapter 6. The two cases considered here relate to a fixed number of available processors and to situations with unlimited number of processors. The optimization analysis is modeled with several programs and tested on various filter designs. The results are presented in Chapter 7. Chapter 8 is the conclusion and recommendation for practical application and further development.

2. Matrix Representation of Digital Filters

The digital filter elements are represented as nodes and branches in a signal flow graph. A typical digital filter signal-flow graph presented in Figure 1.a. is used as a basis for the explanation of the mathematical transformation suitable for the computer filter analysis. The filter consists of 12 nodes and 17 branches, with one input node and one output node. The node numbered 12 is the entry point for the unit pulse input signal ($x_{12}(nT)$). The network response is represented by the signal output at node numbered 1 ($y_1(nT)$). The two types of branches (delay — f_d and coefficient — f_c) are used to interconnect node pairs. Existing interelement relationships at discrete time steps are represented in Figure 1.b. The output signal $y_k(nT)$ of each node is a sum of the signals entering the node, as described by the equation (1).

$$y_k(nT) = x_k(nT) + \sum_{j=1}^{N}[f_{c_{jk}} y_j(nT) + f_{d_{jk}} y_j(nT-T)],\ k = 1,\dots,N. \quad (1)$$

Equation (2) represents the z-transform of the equation (1).

$$y_k(z) = x_k(z) + \sum_{j=1}^{N}[f_{c_{jk}} + f_{d_{jk}} z^{-1}]Y_j(z), k = 1,\dots,N. \quad (2)$$

The above system of N linear equations fully represents the digital filter network with N nodes. The equation (2) can be represented in a matrix form as in equation (3).

$$[I - f_c^t - f_d^t \cdot z^{-1}]Y(z) = X(z). \quad (3)$$

Taking $[I - f_c^t - f_d^t \cdot z^{-1}] = A(z)$, the equation (3) becomes

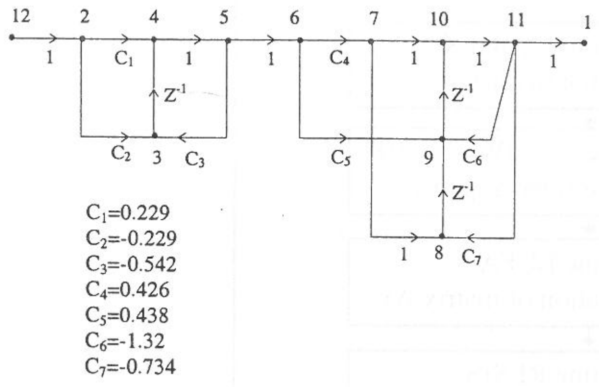
$$A(z) * Y(z) = X(z) \quad (4)$$

where:

A – is the $N * N$ system matrix, unsymmetrical, positive, definite and very sparse, whose elements are complex numbers,

Y – is $N * 1$ column matrix of the node signals $Y_k(z)$ ($k = 1, 2, \dots, N$),

X – is $N * 1$ column matrix of the node input signals $X_k(z)$ ($k = 1, 2, \dots, N$).



a)

$$A=\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
0 & C_2 & 1 & 0 & C_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & C_1 & z^{-1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & C_4 & 1 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & C_7 & 0\\
0 & 0 & 0 & 0 & 0 & C_5 & 0 & z^{-1} & 1 & 0 & C_6 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & z^{-1} & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

b)

Fig. 2. (a) Actual signal-flow graph of a digital filter and (b) its system matrix A.

The principal signal-flow graph of the digital filter presented in Figure 1.a is redrawn in Figure 2.a with the actual parameters. Its system matrix A is presented in Figure 2.b.

Equation (4) is used to calculate the frequency response of the digital filter for a single frequency. Frequency response of the digital filter is a periodic function with a period $2pi/T$, where T is the time between two consecutive signal samples. The sampling frequency is usually normalized in the range from 0 to π radians.

To obtain the complete filter response, it is necessary to solve the equation (4) for each discrete frequency in the range from 0 to π radians (at least 100 discrete frequencies, normally working with 500 for better resolution (Thede, 1996).

According to many digital filter examples the matrix A in system (4) is a sparse matrix with at least 80% zero elements. The Crout's algorithm is especially suitable for the solution of this type of matrix equations (Chua et al., 1975, Stott et al., 1986), due to its pipeline potential, especially for the matrix factorization process (Kacarska et al., 1993, 1995). At the same time, this algorithm preserves the sparseness structure of the system matrix A. The original Crout's algorithm is modified to provide pipeline data parallel processing on distributed memory multiprocessor, or some other type of MIMD (multiple instruction multiple data) architecture. Algorithm for the digital filter analysis is presented in the following chapter.

3. Algorithm for Solving the System $AY = X$

The solution for system (4) using the Crout's algorithm consists of three processes:

1. Matrix A factorization into three matrices $A = L * D * U$, where L and U are triangular matrices with 1s in the diagonal, D is diagonal matrix. There, elements are calculated from the following relations

$$\begin{aligned}
&d_{rr}=a_{rr}^{(r-1)} && r=1,\dots,N\\
&l_{ir}=a_{ir}^{(r-1)} && i=r+1,\dots,N\\
&u_{rj}=a_{rj}^{(r-1)}d_{rr} && j=r+1,\dots,N(5)\\
&a_{ij}=a_{ij}^{(r-1)}-l_{ir}\cdot u_{rj} && \left\{\begin{array}{l} i=r+1,\dots,N\\ j=r+1,\dots,N\end{array}\right. \quad (6)
\end{aligned}$$

2. Step by step forward replacement in the equation $L * D * Y' = X$

3. Step by step backward replacement for the equation $U * Y = Y'$ to get the vector Y

$$y_j = y_j' - \sum_{k=j+1}^{N} u_{jk}\cdot y_k, \quad j = N,\dots,1 \quad (7)$$

The sequential program for digital filter analysis MADF was implemented in FORTRAN.

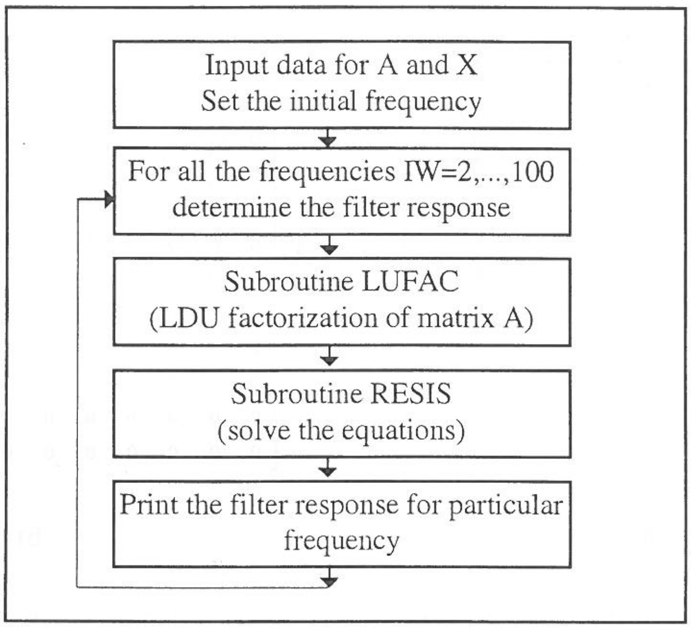

Input data for A and X
Set the initial frequency
For all the frequencies IW=2,...,100 determine the filter response
Subroutine LUFAC (LDU factorization of matrix A)
Subroutine RESIS (solve the equations)
Print the filter response for particular frequency

Fig. 3. The MADF program block diagram.

The block diagram is presented in Figure 3. Obvious characteristic of MADF is formation of the matrix A at the beginning of the algorithm. For any subsequent frequency it is only necessary to change the value of only a few matrix elements (those that represent the delay branches of the filter). So, a network of fixed sparseness structure must be solved repeatedly, with different numeric values.

The program manipulates only the non-zero elements of matrix A, as it is a very sparse matrix. The implementation is based on linear vectors holding only the nonzero elements of the A, L, D and U matrices. This results in memory savings of as much as 90% and the number of operations for the factorization and solution of $AY = X$ is highly reduced (Kacarska, 1988).

With appropriate numeration of filter nodes, it is possible to avoid the process 2 in the solution of system (4), thus speedingup the calculations (Kacarska, 1988).

As the filter response is calculated for K frequency values in the range from 0 to π, every particular frequency represents a single point in the global solution. Structure of the global solution of the filter response is presented in Figure 4.

Another important feature of the sequential algorithm is the recursive nature of the two inherent processes for forward and backward computations.

4. The Parallelization Potential of the Algorithm

Parallel processing techniques can be used in order to speed up the filter response calculations. Several parallelization approaches are considered, depending on the available number of processors.

The first area of parallelization is to use the algorithm implicit global parallel computation processes at the system level (**point level parallelization**). This is the fact that the equation (4) should be used K times for the complete filter frequency response. This approach, although trivial, would require K independent processors. K is minimum 100, and usually 500. The second area of parallelization (process level parallelization) is to use the nature of the LU factorization of the matrix A and the backward replacement process (process 1 and process 3 of the system solution). These processes are part of the calculation of a single frequency digital filter response.

The process for LU factorization of the matrix A starts with an $N * N$ initial matrix $A_0 = A$.

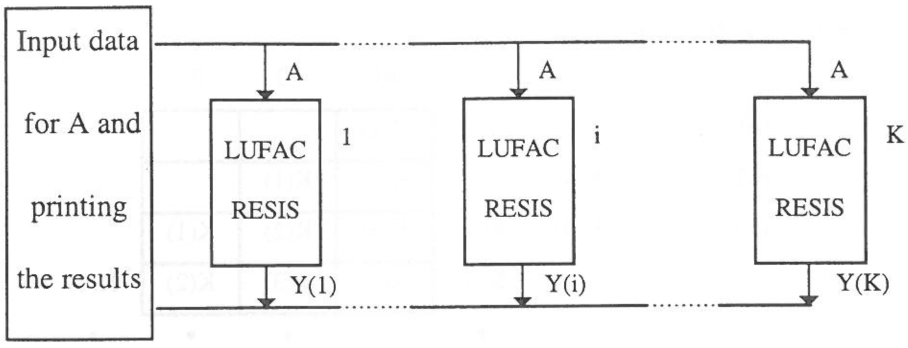


Fig. 4. The global solution block diagram.

Each following step uses a lower order matrix $A_1, A_2, \ldots, A_i, \ldots, A_{N-1}$. The order of the matrix A_i is $(N-i)*(N-i)$, the last matrix A_{N-1} consisting of a single element d_{NN}. There are several efficient parallel algorithms for the LU factorization process on triangular systolic arrays (Kacarska et al., 1993, 1995). For sparse matrices these algorithms result in unbalanced processor load. The backward replacement process uses only the U matrix (upper triangular). In this case, the solution starts with a single element matrix and ends with $N*N$ matrix. As above, the same systolic array implementation characteristics apply for this process, too.

Both, the point-level and process-level parallelization potentials require a large number of processors. For efficient parallelization, a pipelined implementation of the Crout's algorithm for digital filter analysis is proposed. This method integrates the two parallelization potentials of the algorithm, providing balanced performance.

5. Pipeline Implementation

The computation process analysis for a single point has extracted two important computational phases that can be further parallelized in a pipelined fashion:

• LDU factorization (embedded in the process LUFAC), and

• equation solver (embedded in the process RESIS).

Each of the LUFAC and RESIS processes is based on recursive computations, as is indicated in processes 1 and 3 of the system solution. The recursive steps are expanded for both, LUFAC and RESIS, so that the single point solution process can be represented as in Figure 6. The resulting process is composed of $L = N - 1$ pipelined processes, each being distributed to a separate processor. In order to balance the processor utilization, according to the modified algorithm described in the Chapter 7, several processes are grouped within a single processor, resulting in a pipeline of the $m < L$. The idea is to keep all the processors busy for about an equal amount of time. On the other hand, high sparsity of operational matrices depends on the particular filter configuration whose response is to be determined. The number of filter nodes is not a good estimate for the sparsity of the operational matrices, so filters with the same node complexity require various computational complexity. The number of computational steps cannot be estimated in advance.

It is therefore suggested that the system (4) for the first frequency is solved on a sequential processor determining the number of operations required for particular computation steps. Afterwards, the program subroutine grouped processes for the available number of pipelined processors ($m < L$), with balanced computational complexity. The program is then executed on the pipeline in order to compute filter responses for the rest of $K - 1$ points. As soon as one part of the computation for a single process is completed at a processor, the results are sent to the next processor in the pipeline. Such organization has all the positive characteristics of pipelined computations. Figure 5 represents an example of a pipeline operation of a system with $K = 5$ solutions (K different frequencies) for $L = 9$ processes, distributed to a pipeline with $m = 3$ processors. The solution is executed

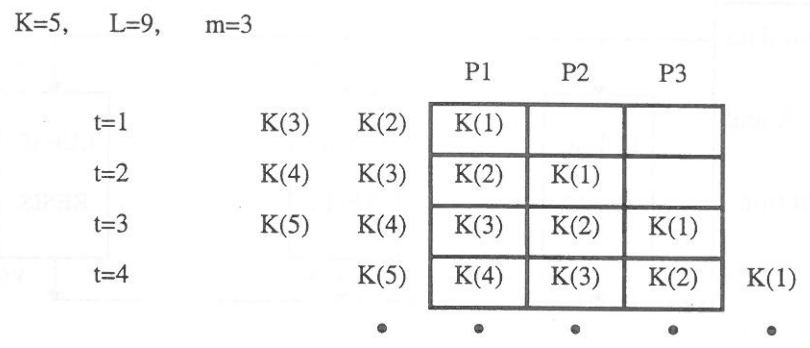


Fig. 5. Pipeline operation example.

in $t = 8$ time steps. One must keep in mind that the pipeline size is usually less than 20 processors and the number of computed points is bigger than 200. Except for the initial delay in data distribution, the rest of the computations flow smoothly through the pipeline. If the processor load is balanced, this gives high computational efficiency is accomplished. Balancing the pipeline performance is therefore very important.

6. Optimization of Process Distribution

The basis for optimal process distribution is the pipelined architecture of the LUFAC and RESIS processes. This enables efficient implementation of the sequential Crout's algorithm. The factorization phase (LUFAC) and the solver (RESIS) are split into $L = N - 1$ pipelined processes (LUP and REP respectively) to be executed on a pipeline of the $m < L$ length.

The LUP and REP processes are pipelined for the following three cases:

1. The step i of the factorization and the step $(N - i)$ of the solver are executed at one processor (Figure 6). Execution time for the processor i is then

$$t(i) = tl(i) + tr(N - i),$$

where $tl(i)$ is the execution time for the LUP process and $tr(N - i)$ is the execution time for the REP process at step i. This organization requires the data transfer only for the elements of the submatrix $A(i - 1)$, while the processor P_i keeps the elements belonging to row i and column i of the matrix. The optimization process is therefore performed on a vector of $(N - 1)$ elements, as presented in Figure 6.

2. The LUP and REP processes are considered independent of each other, and optimization is performed for two vectors independently. In this case two groups consisting of different number of processors are formed and the LUP and REP process distribution within the two groups is different. In general, processing time of the LUP process is much longer than processing time of the REP process, the time $tl(i)$ is dominant. This means that distribution of the LUP processes dominates distribution of the REP processes and the REP processes can be distributed accordingly within the same group of processors. Each processor executes different combination of LUP and REP processes. An example of this kind of distribution is presented in Figure 7 (for the digital filter presented in Figure 2). Figure 7a presents an optimal distribution of the LUP and REP processes separately, while Figure 7b presents an actual distribution of the processes in $m = 2$ processors. The data transfer in this case is similar to the one described for case 1, but there is no transfer for the processes within the same processor.

3. This case takes a sequence of the LUP processes followed by a sequence of the REP processes. A single sequence of independent processes is considered, whose length is the doubled length of individual sequences, $2 * (N - 1)$. The whole matrix A is transferred between the processors. This data transfer slightly overloads the processors, mainly because of the high sparsity of matrix A (for filters with $N > 100$ nodes only $N_A < 4\%$ elements are non-zero). The vector $t(i)$ with length $2 * (N - 1)$ represents both, the LUP and REP execution times. This situation is represented in Figure 8.

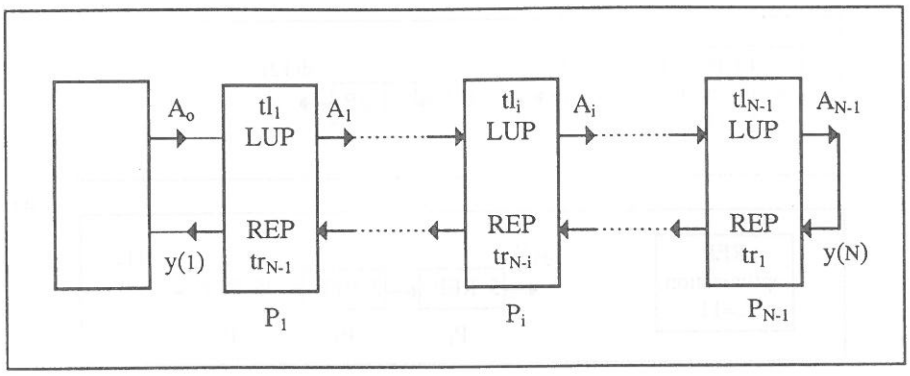


Fig. 6. The distribution of interdependent LUP and REP.

Taking the execution time of each process t_i $(i = 1, \ldots, N-1)$, the total execution time is

$$T = \sum_{i=1}^{N-1} t(i).$$

If there are m processors available, the ideal distribution would keep each processor busy for a period $T_m = T/m$. This provides a balanced performance with small communication overhead.

Nature of the matrix A requires LUP and REP processes with variable execution times, so ideal distribution is impossible. Therefore we try to load the processors as evenly as possible, with small execution times and reduced communication overheads to lowest possible level.

Several program implementations were tested on various filter designs. The digital filter presented in Figure 2 was used in the experiments as example 1, increasing the filter complexity for examples 2 (cascade), 3 (wave) and 4 (wave).

The case with $m = N-1$ processors (each processor for individual LUP) gave fastest execution, but it was trivial and the balance was very bad. For N large, a large number of processors was required and the implementation was very inefficient.

The optimization process considers execution times, communication overheads and the number of processors available. Two general cases were considered:

1. There is a fixed number of processors available (fixed length pipe), and

2. The number of available processors is unlimited (variable).

7. Experimental Results

Two computer programs for process distribution optimization were implemented:

1. MADFPF — for the case with a fixed number of available processors. This program tests the distribution processor loads for the various combinations of neighbor processes. It is based on an algorithm to find element combinations for a set of size N into all subsets of size m arranged in lexicographic order (Reingold et al., 1977). The algorithm is modified for particular program implementations. The modification reduces general computations of the algorithm for the restriction of neighboring processes. The execution time differences are compared for each processor pair, $T = Ti - Tj$ $(i,j = 1, \ldots, m)$. The best distribution has minimal average waiting time.

2. MADFPU — for the case with unlimited number of processors. This program tests process distributions for variable number of processors m $(m = 2, \ldots, N)$. An optimal distribution is selected for minimal average waiting time for each processor in the pipe. For a filter with $N > 50$ nodes, the time to analyze all possible combinations can easily grow higher than the time necessary to complete the filter analysis, so only the cases for up to m=10 processors are analyzed. A measure for optimal timing is developed during the evaluation of process execution times. This measure enables further

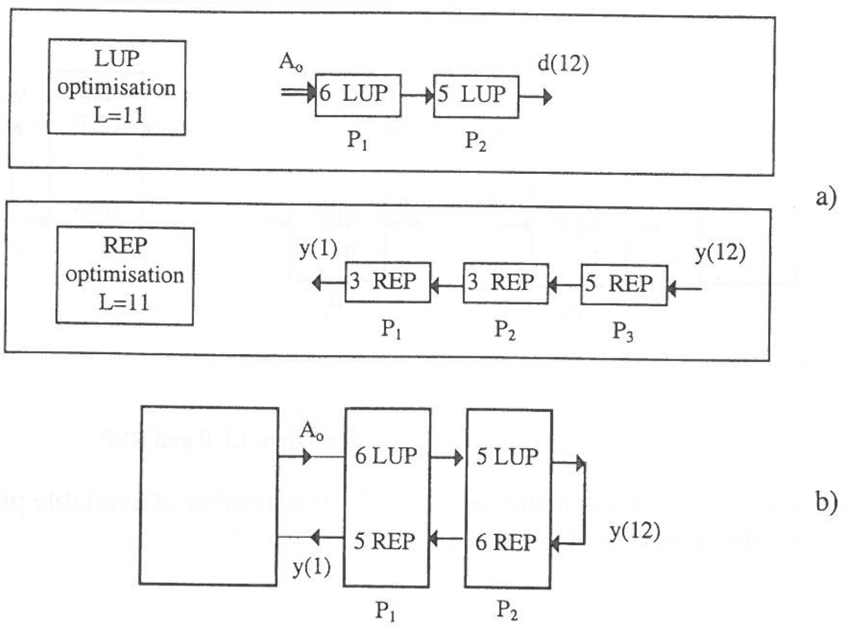


Fig. 7. (a) Process distribution for folded LUP and REP (b) and processor grouping in $m = 2$ processors for digital filter presented in Figure 2.

improvement of this implementation by elimination of unproductive combinations.

Each program is tested with several digital filter examples. Experimental results for 4 representative filter designs are presented in Table 1. The MADFPF program is tested with the number of processors fixed to m=5. The MADFPU program is tested for m<10. Both programs are tested for each of the three cases described in Chapter 3.

Results obtained for the MADFPF program indicate that optimal distribution is achieved by

example	N	sp %	case	L	MADFPF: $m = 5$	MADFPU: $m < 100$
1	12	20.14	1	11	$tm = 135$; $p = 15.74\%$	$m = 4$; $tm = 160$; $p = 10.42\%$
			2	11	$tm = 105$; $p = 17.86\%$	$m = 4$; $tm = 120$; $p = 8.33\%$
			3	22	$tm = 130$; $p = 11.54\%$	$m = 5$; $tm = 130$; $p = 11.54\%$
2	20	12.75	1	19	$tm = 240$; $p = 17.71\%$	$m = 2$; $tm = 525$; $p = 3.81\%$
			2	19	$tm = 180$; $p = 16.67\%$	$m = 2$; $tm = 405$; $p = 7.41\%$
			3	38	$tm = 220$; $p = 7.95\%$	$m = 2$; $tm = 525$; $p = 3.81\%$
3	35	6.94	1	34	$tm = 345$; $p = 5.07\%$	$m = 2$; $tm = 830$; $p = 0.61\%$
			2	34	$tm = 255$; $p = 2.94\%$	$m = 3$; $tm = 420$; $p = 1.78\%$
			3	68	$tm = 345$; $p = 5.07\%$	$m = 4$; $tm = 420$; $p = 1.98\%$
4	50	4.63	1	49	$tm = 510$; $p = 5.64\%$	$m = 2$; $tm = 1235$; $p = 2.83\%$
			2	49	$tm = 390$; $p = 6.73\%$	$m = 2$; $tm = 945$; $p = 4.76\%$
			3	98	$tm = 495$; $p = 2.02\%$	$m = 5$; $tm = 495$; $p = 2.02\%$

N – number of filter nodes and size of matrix A

sp – percentage of non-zero matrix elements

L – number of processes executed in m processors

m – number of processors in pipeline

p – average processor idle time in percents

tm – maximum processor execution time in time units

Table 1. Optimization results for 4 filter designs.

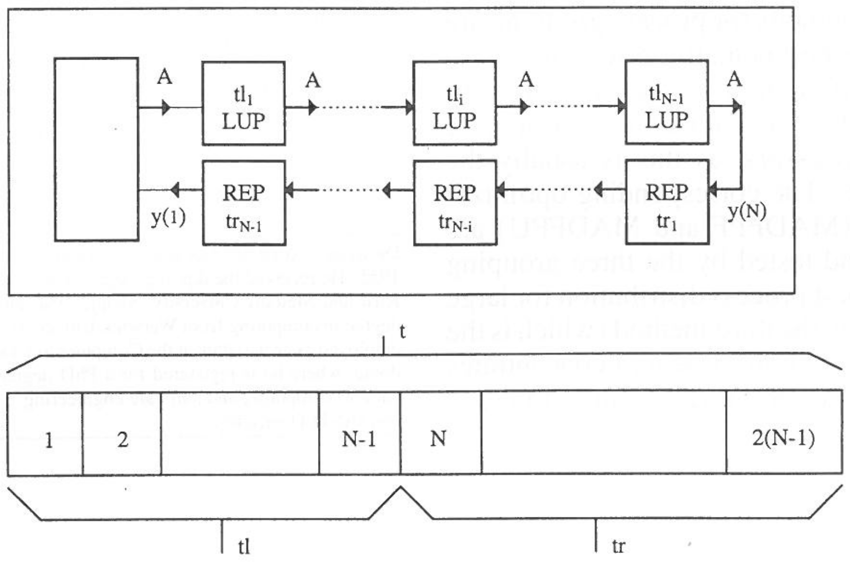


Fig. 8. Process distribution for independent LUP and REP.

the third method of process grouping. The second method is neglected because it is based only on the execution of LUP processes. Due to the LUP processes domination over the REP processes, inclusion of the REP execution times results in further loss of performance. Each of the other two methods produces similar maximum processor execution times, but the critical differences are in processor utilization. For the third method, the parameter p is minimal in all four examples, which is more obvious for larger$N$ (processor efficiency rises to more than 95%). As this method operates with vector twice as long, this optimization naturally takes much longer time than the other two methods.

The MADFPU program takes much longer time than the MADFPF program. In most cases (for smaller filter sizes) it generates optimal solutions for smaller number of processors (m<5), as its optimization criterion is processor load balance and not the minimal execution time. Naturally, if the optimal number of processors obtained is $m = 5$, the results are the same as for the MADFPF program.

8. Conclusion

A digital filter designer has a choice of various filter structures and methods in order to achieve the desired frequency response. In general, the obtained filter structure is analyzed (from 100 to 500 frequencies) in order to check the disagreement limits between the desired and obtained responses. Therefore, many analysis steps might be required and fast analysis algorithms will speedup this process. The pipeline implementation of the Crout's algorithm enables parallelized execution of the digital filter analysis stage. The process distribution must be optimized in order to achieve faster analysis execution and balanced processor performance. This paper presents a fast algorithm for analysis of digital filters with arbitrary topology (including wave filters). There are no limits to digital filter size. Besides, this paper presents a generalized approach to the process distribution optimization, based on an algorithm to find element combinations for a set of size L into all subsets of size m in lexicographic order. The optimization criteria are the number of processors and their utilization.

The program implementation of digital filter analysis gives different execution times on different processors. Therefore, to avoid variable execution times on different processors, the number of operations executed at each processor is taken as a measure of processor execution time. It is assumed that the division operation (5) and update operation (6) have the same execution times. This supports the proposed filter analysis procedure: find sequential algorithm timings for one frequency, optimize it for the particular processor and then execute the parallelized version on the pipeline of optimal length. Three methods for process grouping are proposed and experimentally tested on several digital filter configurations. Two cases are considered, for a fixed and for unlimited number of available processors, as this is usually the case in practice. The corresponding optimization programs (MADFPF and MADFPU) are implemented and tested by the three grouping methods. The best process distribution for large *N* are achieved by the third method (which is the case of interest). Future research concentrates on better and faster optimization algorithms.

References

R. E. CROCHIERE, A. V. OPPENHEIM, Analysis of Linear Digital Networks, *Proc. of the IEEE*, vol. 63, No. 4, (1975), pp. 581–595.

M. KACARSKA, Efficient methods for digital filter analysis in frequency domain, MSc thesis, University St. Kiril I Metodij, Skopje (in Macedonian language), 1988.

M. KACARSKA, D. ANDONOV, Pipeline implementation of a digital filter analysis algorithm, *Proc of the IV Theme ETAI Symposium*, Ohrid, Macedonia, 1993.

M. KACARSKA, D. ANDONOV, Optimal module distribution for pipeline digital filter analysis algorithm *Proc of the 17th International Conference ITI'95*, Pula, Hrvatska, 1995.

B. STOTT, O. ALSAC, An Overview of Sparse Matrix Techniques for On-Line Network Applications, *Proc of the IFAC Symposium Computer Applications in Large Power Systems*, China, 1986.

LEON CHUA, PEN–MIN LIN, *Computer-aided Analysis of Electronic Circuits*, Prentice Hall, N. Jersey, 1975.

E. M. REINGOLD, J. NIEVEGELT, N. DEO, *Combinatorial Algorithms: Theory and practice*, Prentice–Hall, N. Jersey, 1977.

LES THEDE, *Analog and Digital Filter Design Using C*, Prentice–Hall, N. Jersey, 1996.

Received: July, 1995
Accepted: October, 1997

Contact address:

Marija Kacarska, Dragan Andonov
Elektrotehnički fakultet
Karpos II bb.
91 000 Skopje
R Macedonia
tel: 389–91–363566
fax: 389–91–364262

MARIJA KACARSKA was born in Skopje, Macedonia, on June 16, 1954. She received the dipl.ing. and M.Sc. degrees in electrical engineering from St. Kiril and Metodij University, Skopje, Macedonia in 1978 and 1988, respectively. Her current position is of an assistant at the ETF Skopje, Macedonia, where she is registered for a PhD degree. Her interests are in circuit theory, especially in digital filters and computer aided network analysis. She is a member of IEEE and IASTED societes.

DRAGAN ANDONOV was born in Skopje, Macedonia, on August 20, 1955. He received the dipl.ing. degre in electrical engineering from St. Kiril and Metodij University, Skopje, Macedonia in 1979 and M.Sc. degree in computing from Warwick University, England in 1990. He is employed as an assistant at the Computer Sci. Dept. ETF Skopje, Macedonia, where he is registered for a PhD degree. His interests include parallel computing and software engineering. He is a member of IEEE and IASTED societes.