Comparative Performance of the Burg Algorithm Implemented in Parallel FORTRAN and the Applicative Language SISAL

A.L. Cricenti
G.K. Egan

Technical Report 31-026

Version 1.0 March 1991

Abstract

This paper describes the implementation of a signal processing algorithm, specifically the Burg Algorithm, using both a high level parallel language SISAL and Encore Parallel FORTRAN. The Burg Algorithm is an estimation technique for fitting an autoregressive model to a time series data set. This algorithm contains a time shift / inner product operation which is used in a number of other important signal processing algorithms such as convolution.

COMPARATIVE PERFORMANCE OF THE BURG ALGORITHM IMPLEMENTED IN PARALLEL FORTRAN AND THE APPLICATIVE LANGUAGE SISAL

A.L. Cricenti and G.K. Egan

1. Introduction

Signal Processing Algorithms are widely used and of vital importance in areas such as biomedical engineering, seismic data analysis, speech analysis and spectral estimation. The demand that signal processing algorithms place on computing system performance is increasing as more complicated algorithms are made to function in real time. As the limitations of current uniprocessor systems are being reached, many computer manufacturers are turning to multiprocessor configurations to obtain increased performance. In addition to hardware limitations, current computer languages must be evaluated for performance and ease of use with reference to their suitability for parallel machines.

The purpose of this paper is to describe the implementation of a signal processing algorithm, specifically the Burg Algorithm[1], using both a high level parallel language SISAL (Stream and Iteration in a Single Assignment Language)[2] and EPF (Encore Parallel FORTRAN). The Burg Algorithm is an estimation technique for fitting an autoregressive model to a time series data set. This algorithm contains a time shift / inner product operation which is used in a number of other important signal processing algorithms such as Convolution.

The results obtained using an optimising SISAL compiler are compared to those obtained using the EPF (parallel FORTRAN) annotator. The comparison is made both on a 4 XPC processor (4 Mips per processor) based Encore Multimax Multiprocessor machine and a single processor IBM RS6000/530 (30 Mips) system. The performance of the IBM processor is representative of processors in next generation medium cost multiprocessors.

2. The Burg Algorithm

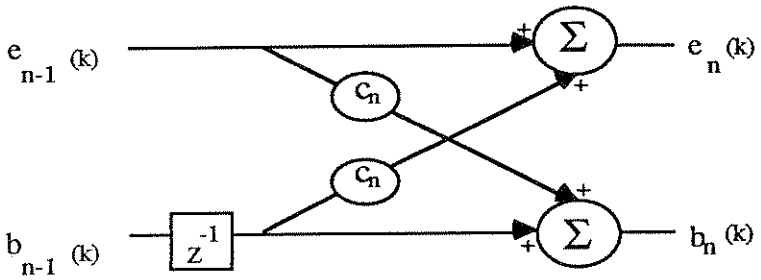
The Burg Algorithm is a method of generating an autoregressive model from a set of data samples, that is it gives estimates for A(z) in:

$$H(z)=\frac{1}{A(z)}$$

There are several ways of obtaining an AR model, the Burg algorithm is based on minimising the forward and backward prediction errors, assuming a lattice filter structure as shown in figure 1.

A.L. Cricenti is a member of Faculty in the School of Electrical Engineering and a Researcher in the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology, John Street, Hawthorn 3122, Australia, Phone:+61 3 819 8516, E-mail: alc@stan.xx.swin.oz.au.

G.K. Egan is Professor of Computer Systems Engineering and Director of the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology, John Street, Hawthorn 3122, Australia, Phone:+61 3 819 8516, E-mail: gke@stan.xx.swin.oz.au.



where

$e_n(k) = e_{n-1}(k) + c_n\, b_{n-1}(k-1)$ forward prediction error (1)

$b_n(k) = c_n\, e_{n-1}(k) + b_{n-1}(k-1)$ backward prediction error (2)

and c_n are called the reflection coefficients.

Figure 1 lattice filter structure

The Burg Algorithm involves the choice of reflection coefficients such that the error energy is minimised, when only a finite number of data samples is available.

The optimum value of the reflection coefficients, can be easily derived[7] and is given by:

$$c_n = -2\frac{\sum_{k=n}^{M} e_{n-1}(k) \cdot b_{n-1}(k-1)}{\sum_{k=n}^{M} e_{n-1}^{2}(k) + b_{n-1}^{2}(k-1)} \qquad (3)$$

where

M is the number of data samples.

and the AR parameters can then be estimated from:

$$a_n = c_n \qquad (4)$$

$$a_n(j) = a_{n-1}(j) + c_n a_{n-1}(n-j) \quad \text{for } j = 1....n-1 \qquad (5)$$

A sequential implementation of the Burg algorithm is outlined in [1] and is reproduced in figure 2a with individual tasks labelled $T_{in}(1)$, $T_n(1)$, $T_{in}(2)$, $T_{nn}(2)$, $T_{in}(3)$. Tasks $T_{in}(1)$ are computations of the inner products in both the numerator and denominator of (3) above. $T_n(1)$ is the calculation of the division needed to compute c_n. This task cannot proceed in parallel, since it depends on the results of task $T_{in}(1)$. Also reproduced in figure 2b is the maximally parallel graph for M=5 and MAX=3. $T_{in}(2)$ updates the autoregressive coefficients and task $T_{in}(3)$ updates the forward and backward prediction errors.

```
1. INITIALIZATION
   FOR i=1 TO M DO
       e(i)=x(i)
       b(i)=x(i)
2. THE MAIN LOOP
   FOR n=1  TO MAX DO
       s1=0.0; s2=0.0
       FOR i=n+1 TO M DO
           s1=s1+e(i)*b(i-n)              Tin(1)
           s2=s2+e(i)**2+b(i-n)**2
       c(n)=-2.0*s1/s2                    Tn(i)
       IF n>1 THEN DO
           FOR i=1 TO n-1 DO
               a1(i)=a(i)+c(n)*a(n-i)     Tin(2)
           FOR i=1 TO n-1 DO
               a(i)=a1(i)
       a(n)=c(n)                          Tnn(2)
       FOR i=n+1 TO M DO
           temp=e(i)+c(n)*b(i-n)          Tin(3)
           b(i-n)=b(i-n)c(n)*e(i)
           e(i)=temp
```

Figure 2a sequential Burg Algorithm for m data points and max reflection coefficients

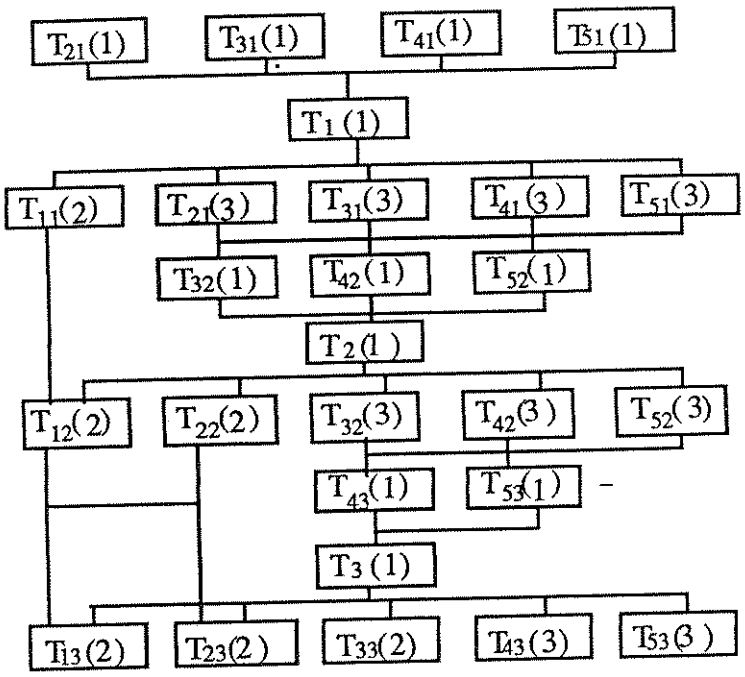


Figure 2b maximally parallel graph for m=5 and max=3 from [1]

3. Language Considerations

3.1 Encore Parallel FORTRAN

The Encore Parallel FORTRAN compiler (EPF) is the UMAX f77 implementation of FORTRAN with enhancements which allow parts of a program to be executed in parallel. These statements are PARALLEL, DO ALL, CRITICAL SECTION, BARRIER, LOCK WAIT, LOCK SEND, and EVENT.

The EPF compiler consists of analysis and transformation tools, a parallelising compiler, parallel runtime libraries, and a code generator. Whilst programs can be written directly in EPF, EPF can also be used to convert a standard FORTRAN program into a source which is annotated with the parallel statements outlined above. During compilation, EPF first detects possible concurrent parts of the source programs, these are shown in a .LST file. The annotator then generates the EPF program , .E file, by inserting the appropriate EPF statements.

The EPF annotator may require user intervention to produce the most efficient code for a particular program; however, useful speedup can be achieved by relying on the annotator alone.

3.2 SISAL

SISAL is a functional language which has been targeted at a wide variety of systems including current generation multiprocessors such as the Encore Multimax and research dataflow machines [2][3][4]. The textual form of SISAL, in terms of control structures and array representations, provides a relatively easy transition for those familiar with imperative languages. The optimising SISAL compiler (OSC) from Colorado yields performance competitive with FORTRAN [5].

4. Parallel Implementation

The simplest parallel implementation of the Burg algorithm is obtained by coding the sequential algorithm in FORTRAN and then using the Encore Parallel FORTRAN compiler (EPF) to produce the parallel code suitable for the Encore Multimax Multiprocessor. This process requires that the programmer to know very little about the underlying architecture of the machine, thus code may be generated very easily. This method is also attractive since it allows existing software, written in FORTRAN, to be implemented on parallel machines without any translation. The disadvantages of this method are that: optimum speedup is usually not obtained; and that the annotated code produced by the EPF compiler is machine dependent. Appendix 1 shows the EPF annotated version of the Burg Algorithm. The annotator has identified that all loops except the outer loop can be parallelised. Thus the annotator can successfully identify the parallel loops. The maximally parallel graph suggests that tasks $T_{in}(2)$, $T_{nn}(2)$ and $T_{in}(3)$ could be performed at the same time; unfortunately EPF can only slice loops, and since the outer loop is sequential, due to task $T_n(1)$, EPF cannot make these tasks parallel.

The second implementation is to code the algorithm in SISAL. The disadvantage of a SISAL implementation is that existing codes need to be rewritten in the SISAL language. The SISAL implementation of the Burg Algorithm as presented in [1], was directly transliterated from the FORTRAN version, excepting the loop which updates the autoregressive coefficients shown in figure 3a, which was transformed into a parallel format to ease the coding.

```
a:=     %calculate auto regressive coefficient
        for k in 1,old n
              returns array of
                  if k = old n then
                        c
                  else
                              old a[k]+c*old a[old n-k]
                  end if
        end for;
```

Figure 3a implementation of the calculation of the autoregressive coefficients in SISAL.

The loop which which updates the forward and backward errors was also changed. The original FORTRAN loop has been split into two loops. This was done so that the indexes of b change in manner which is suitable for the parallel SISAL *'for'* loop, refer fig 3b.

```
e:=
    for l in old n+1,m
     returns array of
           old e[l]+c* old b[l-old n]
    end for;
b:=
    for j in 1,m - old n
        returns array of
            old b[j]+c*old e[j+old n]
    end for;
```

Figure 3a implementation of the calculation of the autoregressive coefficients in SISAL.

5. Results

The SISAL and FORTRAN versions of the program were run on both an Encore Multimax and IBM RS6000/530 system using the standard f77 FORTRAN compiler, the EPF compiler and the optimising SISAL compiler.

For comparison purposes the number of data points was set to m=10000 and the model size to max=100.

The run times obtained for both the FORTRAN and SISAL implementations of the algorithm on the Encore Multimax multiprocessor with four XPC processors are summarised in table 1.

Processors	Real	User	System	Speedup	Efficiency
1	30	30.0	0.2	1.00	1.00
2	16	32.9	0.4	1.81	0.91
3	11	33.8	0.4	2.65	0.88
4	9	35.7	0.5	3.34	0.84

Encore Parallel FORTRAN

Processors	Real	User	System	Speedup	Efficiency
1	39.31	38.35	0.92	1.00	1.00
2	19.43	18.92	0.47	2.02	1.00
3	14.51	14.06	0.36	2.71	0.90
4	11.94	11.55	0.28	3.29	0.82

SISAL

Note: Speedup = T 1 / T n Efficiency = Speedup(n) / n

Table 1 times, speedup and efficiency for the Encore Multimax

As can be seen from the run times, speedup is achieved with the EPF compiler without significant programming effort. The EPF compiler converts DO LOOPS to parallel code. However, the annotator is fairly conservative, and further speedup may be obtained, in some instances, by manually annotating programs.

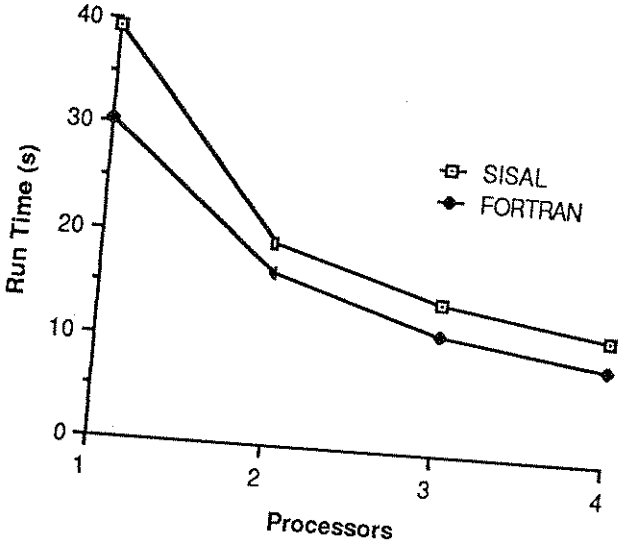


Figure 4a run time vs processors

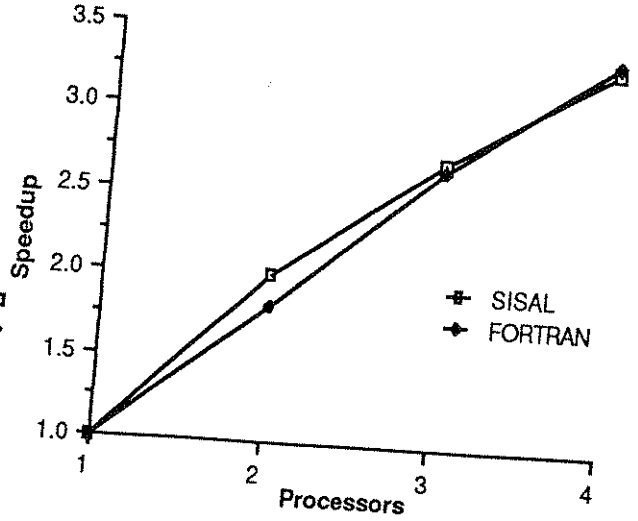


Figure 4b speedup vs processors

As can be seen, from the results, the FORTRAN and SISAL implementations achieve similar speedup, but more importantly the FORTRAN implementation has a lower run time.

It should be noted that initially no speedup was achieved with the SISAL implementation of the Burg Algorithm. Speedup was achieved by forcing the SISAL compiler to slice all FOR loops, by setting the -H pragma to 500; the cost estimator in SISAL had deemed the low complexity loops not worth slicing. The value of 500 was arrived at by trial and error. As this pragma is applied globally in the current version of the SISAL compiler there may be a risk of over parallelisation of some loops [6].

Speedup for the SISAL implementation increases beyond four processors as can be seen from the speedup curve shown in figure 5. These results were obtained on a slower 20 APC processor Encore Multimax. The graph shows that the speedup tends to asymptote to a value of about 10. This limitation is due to the sequential part of the algorithm. Note, only 16 processors were used so that interference from other users was minimised.
The parallel FORTRAN (EPF) compiler was not available on this system.

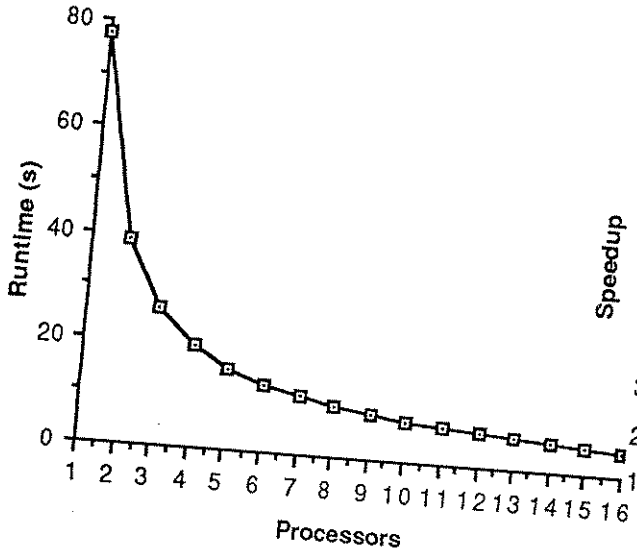


Figure 5a runtime vs processors (16 processors)

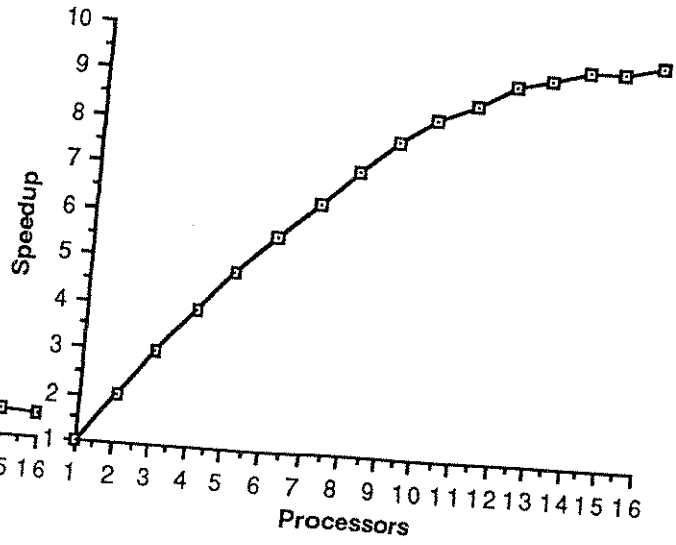


Figure 5b speedup vs processors

The run times for a single processor are summarised in table 2. These times are for a model size of m=10000 and max=100. The time for the SISAL implementation is comparable with the FORTRAN (XLF) implementation.

	User	System
SISAL	0.92	0.0
FORTRAN	4.24	0.7
FORTRAN - O	0.42	0.5

Table 2 times for the IBM RS6000/530 (m=10000, max=100)

The results for the IBM RS6000/530 and Cray Y-MP are shown in table 3 for comparison with the results from [1]. The parameters for this study were m=16384 data points and model size max=10.

Machine	*iPSC/2*	*MPP*	*X-MP/48*	Y-MP	RS6000
Execution Time (s)	*0.24*	*0.05522*	*0.016887*	0.009 (1p)	0.19

Times from [1]

Table 3 comparison of Burg Algorithm execution time (m=16384, max=10)

6. Conclusions

The Burg filter was implemented both in FORTRAN and SISAL. Significant speedup is achieved with the SISAL implementation, suggesting that SISAL may be a useful language for signal processing algorithms. FORTRAN annotators such as the EPF annotator are useful in that speedup is obtained for little effort, and existing FORTRAN implementations of algorithms need not be recoded. Run times on modern single processor machines such as the IBM RS6000/530, are comparable to some existing parallel architecture machines.

Acknowledgements

The authors thank and the members of the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology, in particular P.S. Chang, for their assistance in this research.

The Laboratory for Concurrent Computing Systems is funded under a special research infrastructure grant for parallel processing research by the Australian Commonwealth Government.

References

[1] N.M. Sammur and M.T. Hagan. "Mapping Signal Processing Algorithms on Parallel Architectures." Journal of Parallel and Distributed Computing, Issue no.8 1990 pp180-185.

[2] McGraw et al., "SISAL: Streams and Iteration in a Single Assignment Language." Language Reference Manual, M146, Lawrence Livermore National Laboratories.

[3] A.P.W. (Wim) Bohm and J. Sargeant, "Efficient Dataflow Code Generation of SISAL", Technical Report UMCS-85-10-2, Department of Computer Science, University of Manchester, 1985.

[4] G.K. Egan, N.J. Webb and A.P.W. (Wim) Bohm, "Some Features of the CSIRAC II Dataflow Machine Architecture", in Advanced Topics in Data-Flow Computing, Prentice-Hall 1990,

[5] D.C. Cann, "High Performance Parallel Applicative Computation", Technical Report CS-89-104, Colorado State University, Feb.1989.

[6] P.S. Chang, and G.K. Egan "Analysis of a Parallel Implementation of a Numerical Weather Model in the Functional Language SISAL" Technical Report 31-012, Laboratory for Concurrent Computing Systems, School of Electrical Engineering, Swinburne Institute of Technology, March 1990.

[7] R.A. Roberts and C.T. Mullis, "Digital Signal Processing" Addison - Wesley 1987

Appendix A

The following listing shows the EPF annotated version of the FORTRAN implementation of the Burg Algorithm.

```
                                      1          program burg
                                      2    c     G.K. Egan
                                      4          double precision x(10000), e(10000),
                                                                  b(10000),c(10000)
                                      5          double precision a1(10000),a(10000)
                                      6          double precision s1,s2,temp
                                      7          integer m,n,max
                                     10          m=10000
                                     11          max=100
                                     12    c
      C             +---------     13          do 100 i=1,m
                    *              14             x(i)=float(i)
                    *_________     15    100   continue
                                     16    c
                                     17    c     main code
                                     18    c
      C             +---------     19          do 200 i=1,m
                    *              20          e(i)=x(i)
                    *              21          b(i)=x(i)
                    *_________     22    200   continue
                                     23    c
     NC             +---------     24          do 300 n=1,max
                    !              25          s1=0.
                    !              26          s2=0.
      C             !+--------     27          do 400 i=(n+1),m
      C             !*             28            s1=s1+e(i)*b(i-n)
      C             !*             29            s2=s2+(e(i)*e(i))+(b(i-n)*b(i-n))
                    !*_________    30    400     continue
                    !              31          c(n)=-2.*s1/s2
                    !              32    c
                    !              33          if (n.gt.1) then
      C             !+--------     34            do 500 i=1,(n-1)
                    !*             35              a1(i)=a(i)+c(n)*a(n-i)
                    !*_________    36    500       continue
      C             !+--------     37            do 600 i=1,(n-1)
                    !*             38              a(i)=a1(i)
                    !*_________    39    600       continue
                    !              40          endif
                    !              41    c
                    !              42          a(n)=c(n)
      C             !+--------     43          do 700 i=(n+1),m
                    !*             44            temp=e(i)+c(n)*b(i-n)
                    !*             45            b(i-n)=b(i-n)+c(n)*e(i)
                    !*             46            e(i)=temp
                    !*_________    47    700     continue
     NC             !              48
                    !              49    c
                    !_________     50    300   continue
     NC             +---------     51          do 1000 i=1,nmax
     NC             !              52             write(6,*) a(i)
                    !_________     53    1000  continue
                                     54          stop
                                     55          end

Abbreviations Used
 NC       non-concurrent-stmt      C        concurrentize
  8 loops total
            1 loops left as DO loop
            1 preferred scalar mode
            6 loops concurrentized
```

Appendix B

The following listing shows the SISAL version of the Burg Algorithm.

```
% A.L. Cricenti 1990.
%Burg algorithm, procedure to fit an autoregressive model to  a time series data set.
define main

type    vector = array [double_real];

function burg(m, max:integer returns vector)
  for initial
    a,b,e:  vector;
    a, b, e :=  for i in 1, m
                    ii:=double_real(i)
                returns array of ii
                    array of ii
                    array of ii
                end for;
    n:=1;
  while (n <= max) repeat
    s1, s2 :=                                          %calculate sums s1,s2
      for i in old n+1,m
        returns value of sum old e[i]*old b[i-old n]
                value of sum old e[i]*old e[i]+old b[i-old n]*old b[i-old n]
      end for;
    c:= -2.0d0*s1/s2;                                  %calculate reflection coefficient
    a:=                                                %calculate auto regressive coefficient
      for k in 1,old n
      returns array of
            if k = old n then
              c
            else
              old a[k]+c*old a[old n-k]
            end if
      end for;
    e:=                                                %calculate forward prediction error
      for l in old n+1,m
      returns array of
            old e[l]+c* old b[l-old n]
      end for;
    b:=                                                %calculate backward prediction error
      for j in 1,m - old n
        returns array of
          old b[j]+c*old e[j+old n]
      end for;
    n:=old n +1
  returns value of a
  end for
end function
function main(returns vector)
  let
    m:=10000;                                          %Number of data sample
    max:=100;                                          %Number of coefficients (model size)
  in
    burg(m,max)
  end let
end function % main;
```