Memoirs of the Faculty of Engineering, Okayama University, Vol.24, No. 2, pp.99-112, March 1990

Parallel Skyline Method using Two Dimensional Array

Takeo Taniguchi* & Kohji Fujiwara**

(Received January 14 , 1990)

SYNOPSIS

This paper presents an effective solver for a large sparse set of linear algebraic equations, which appears at the application of the finite element and the finite difference methods in engineering field. Proposed method is a family of SKYLINE METHOD, and for faster computation on the vector processors the original skyline is modified with respect to following three items; the use of inner products of matrix operations, the removal of unnecessary numerical operations and the introduction of two-dimensional array for storing the data of coefficient matrix.

1. INTRODUCTION

In accordance with the development of supercomputer, the size of problems to be treated becomes larger and larger, and better solvers have been always sought by many mathematicians and engineers. Especially, the requirement became very strong for the solver of large sparse set of linear algebraic equations which necessarily apppear at solving 3-dimensional problems.

The conditions of "a good solver" are less memory size and fast computation, and at present we find two selections, i.e. PRECONDITIONED CONJUGATE GRADIENT METHOD and SKYLINE METHOD. The former belongs to the iterative one, and the latter the direct method.

* : Department of Engineering Science
** : Department of Civil Engineering

PCG (Preconditioned Conjugate Gradient) method apparently requires less memory-size comparing to the Skyline Method, but we are anxious on CPU-time and also on the infeasible computation due to the preconditioner used in the solver. On the contrary, the skyline method is reliable on its computation, and it has been one of the most popular solvers in the last decade.

Skyline method introduces one-dimensional array for storing data of coefficient matrix of linear equations in order to remove unnecessary zero entries which are not used for the elimination calculation. This method is modified as Parallel Skyline Method for improving the applicability to supercomputers. The aim of this modified method is to utilize in the matrix computations the property of the degree-of-freedom (DOF) appeared in many engineering problems. That is, in order to use the fast computation facility of the supercomputer in inner products for matrices, the elimination computation is done by treating submatrix whose size is set to be DOF as a unit.[1,2,3] But, the data of the parallel skyline method is still stored in one-dimensional array as the one for the original skyline method.

Another effective solver belonging to the skyline method is COLSOL.[4] This solver can remove unnecessary numerical operations during the elimination process, and as a result it can save CPU-time.

The purpose of this investigation is to show the influence of factors (the data storing method, the matrix operations, and the removal of unnecessary numerical operations during the elimination process) to CPU-time. Finally, we propose a kind of skyline methods which can introduce all of these factors. The listing of new solver is attached in Appendix A, which is valid for symmetric coefficient matrices. A sample of the use of the proposed skyline method is also given in Appendix B.

2. PARALLEL SKYLINE METHOD

Let

$$Ax = b \qquad (1)$$

be a set of linear algebraic equations, where A is a $n*n$ coefficient matrix, x is the solution vector, and b is a given vector. Here, we assume A is sparse, symmetric and possitive definite. Since the coef-

ficient matrix A is symmetric, the matrix can be factorized into the matrix product of the lower triangular (L), diagonal (D), and upper triangular matrices (U) by using Modified Cholesky factorization. Moreover, using the nature that U is equal to the transpose of the lower triangular matrix, we obtain

$$A = LDL^t \tag{2}$$

Firstly we explain original skyline method. This method consists of two processes; the forward elimination and the backward substitution. From the characteriscics of the direct method the forward elimination process mainly governs CPU-time of the modified Cholesky factorization method, and, then, the decreasing of the execution time can be firstly achieved when CPU-time of the forward elimination is saved. Then, our discussion is concentrated in the treatment of the forward elimination process.

This elimination process is expressed in eq.3, and the numerical operation is proceeded downward from the uppermost nonzero entry to the main diagonal columnwisely as shown in Fig.1-a.

$$\begin{aligned} d_{ii} &= a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 d_{kk} \qquad (1 \leqq i \leqq n) \\ l_{ij} &= (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} d_{kk}) / d_{jj} \\ l_{ii} &= 1 \qquad (1 \leqq i \leqq n, j < i) \end{aligned} \tag{3}$$

From this computational ordering for off-diagonal entries the data of the coefficient matrix are stored columwisely in one-dimensional array. Since all data are stored in one dimensional array, this method is suitable to the vector processor.

Equation 3 expressing the elimination process suggests that any off-diagonal nonzero element can be columnwisely factorized, and, therefore, each column can be factorized independently. That is, the elimination process of the skyline solver can be treated in parallel. Consideration on these characteristics of the factorization process leads to two types of skyline methods as shown in Fig.1.[1,2,3] The column or row indicated by arrows in the figure shows the elements to be factorized, and the shaded zone indicates the entries which are

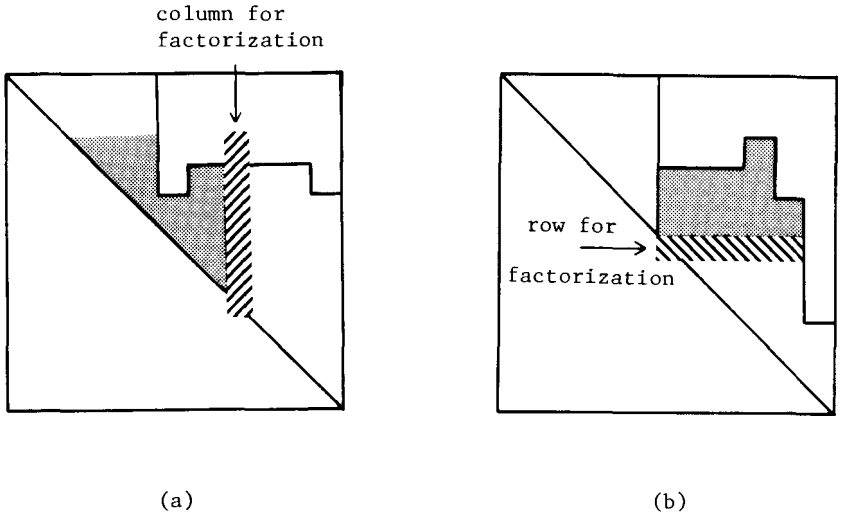


Fig.1 Two types of Modified cholesky Factorization

used for the factorization.

In case of m-dimensional problems the degree-of-freedom of any node is m, and the coefficient matrix can be divided into a gathering of m*m submatrices. Then, their factorization can be proceeded by treating m*m submatrix as a unit and, m columns of each submatrix can be treated in parall. These characteristics can be effectively introduced into the procedure of the forward elimination. These methods are generally called the parallel skyline method.

Eq.3 suggests that the forward elimination procedure can be improved still more for saving CPU-time. For example, any factorized lower triangular element, l(i,j), requires the multiplication and also the subdivision by the same main diagonal entries. If these unnecessary operations can be removed from the elimination process by changing the ordering of the numerical operations, then the execution time can be largely saved. A solver named COLSOL is the one which can remove these unnecessary operations, and the method is, at present, the fastest solver among the direct methods. [4]

In general, the skyline method uses one-dimensional array for storing the data of the coefficient matrix. But, it is obvious that the factorization process in parallel computaion requires the searching of data which can be treated in parallel. This indicates that CPU-time for searching these data can be saved, if all data are stored in a two-dimensional array of (m*mN) for the problem with N nodes and m

degree-of-freedom.

As explained already, skyline-type solvers presently in use are the parallel skyline method using one-dimensional array and COLSOL. But, above considerations clarify that we can propose several new solvers by introducing and combining items explained already. Successive section is used for the proposal of these new solvers and their examination.

3. FAMILY OF PARALLEL SKYLINE METHODS AND THEIR EFFICIENCY

The family of the skyline method can be classified using following items:

a. The difference of the elimination processes (See Fig.1.)
b. The data storing method (One- or two-dimensional array)
c. Treatment of the diagonal entries (Refer to COLSOL.)

By considering these items following solvers can be proposed:

(1) SKY (Original skyline method)
(2) COLSOL
(3) 1PARSKYa : Parallel skyline method using 1-dim. array and the elimination process of Fig.1-a
(4) 1PARSKYb : Parallel skyline method using 1-dim. array and the elimination process of Fig.1-b
(5) 2PARSKYa : Parallel skyline method using 2-dim. array and the elimination process of Fig.1-a
(6) 2PARSKYb : Parallel skyline method using 2-dim. array and the elimination process of Fog.1-b
(7) 2PARSKYc : Parallel skyline method using 2-dim. array and the technique of COLSOL

SKY and COLSOL are the original solvers presently in use, 1PARSKYa and 1PARSKYb are developed by Miyoshi et al (see [1,2]), and the residuals (2PARSKYa, 2PARSKYb, and 2PARSKYc) are newly developed in this investigation which are based on the solvers proposed by Miyoshi et al, include the techniques of COLSOL and use 2-dimensional array for data storing.

The efficiency of these solvers are examined through the numerical experiments for following test problems. The first one is the actual 3-dimensional structural problem as shown in Fig.2. A cube subjecting distributed load on its upper surface and fixed at its bottom surface is analyzed using the displacement-type finite element method. The

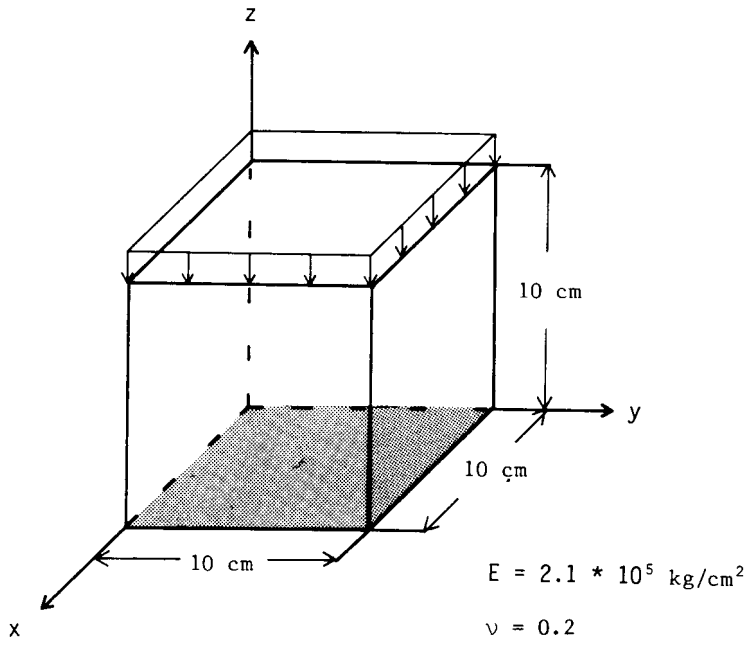


Fig.2 Problem 1

cube is regularily divided by 8-point isoparametric elements.

This cube is divided into several types of finite element models, and they are solved using seven solvers mentioned above. The execution-time required for these solvers is summarized in Table 1. In the table N shows the number of equations, and HBW shows the half bandwidth of coefficient matrices. These problems are 3-dimensional ones, and the 2-dimensional array storing the data are (3,3*NODE), where NODE shows the number of nodes which are set in the model. This table shows CPU-time in seconds required for solving these problems. The values in parentheses show the ratio of CPU-time between SKY and other solvers.

The results suggest that COLSOL is the fastest solver among them, and the 2PARSKYc comes next. That is, the removal of unnecessary numerical operations through the factorization process is most effective for decreasing the execution-time.

The row of V-ratio shows the vectorization ratio of these solvers, and we know that more than 90% of the procedure can be vectorized for all of them. These values suggest that the family of the skyline method is suitable for the vector machine.

N	HBW	SKY	1PARSKYa	1PARSKYb	2PARSKYa	2PARSKYb	2PARSKYc	COLSOL
1152	33	0.1497682 (1)	0.1028552 (1.46)	0.0822869 (1.82)	0.0862447 (1.74)	0.0765069 (1.96)	0.0762478 (1.96)	0.0757617 (1.98)
1620		0.2270194 (1)	0.1567595 (1.45)	0.1235989 (1.84)	0.1293761 (1.75)	0.1152391 (1.97)	0.1146158 (1.98)	0.1135362 (2.00)
1944		0.2819681 (1)	0.1960989 (1.44)	0.1573838 (1.79)	0.1646881 (1.71)	0.1469377 (1.92)	0.1460794 (1.93)	0.1440782 (1.96)
1080	96	0.6269479 (1)	0.3459422 (1.81)	0.2901599 (2.16)	0.3005155 (2.09)	0.2752899 (2.28)	0.2687449 (2.33)	0.2635265 (2.38)
1620		0.9746556 (1)	0.5346052 (1.82)	0.4485185 (2.17)	0.4645214 (2.10)	0.4274530 (2.28)	0.4151756 (2.35)	0.4073789 (2.39)
1944		1.1906687 (1)	0.6320499 (1.88)	0.5435475 (2.19)	0.5629893 (2.11)	0.5158620 (2.31)	0.5031210 (2.37)	0.4936866 (2.41)
V-RATIO(%)		98.90	95.70	98.31	96.32	94.84	98.84	98.59

Table 1 Results of Problem 1

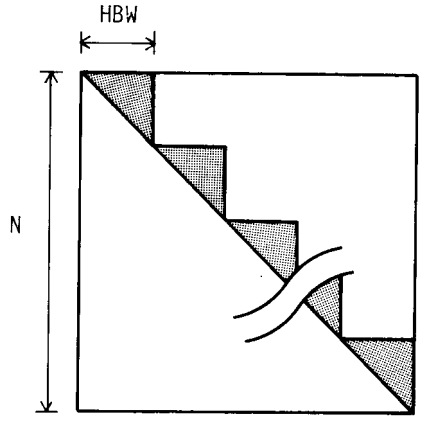


Fig.3 Problem 2

The second problem is an artificial one whose coefficient matrix shows a skyline type of nonzero zone as shown in Fig.3. This problem is used for the examination of the influence of the use of two-dimensional array to the execution-time, and by considering DOF of submatrices the number of rows of the two-dimensional array is set 3, 5 and 7.

All data of the computations are summarized in Table 2 which show only the ratio of the execution time between SKY and other solvers.

The results show that the fastest solver is 2PARSKYc, and that it becomes fastest when DOF is set to be 5. The reason can be explained as following: In accordance with the increase of m the computation of the inner product of matrices becomes effective, but on the contrary, since total number of variables is fixed, the vector length becomes short and its computation becomes relatively long. As a result, CPU-time becomes shortest, when DOF is equal to 5.

Moreover, we can recognize that solvers of the skyline method become effective for problems with bigger bandwidth. Its reason is that the vector length becomes longer in accordance with the increase of HBW.

All solvers can be vectorized more than 90%, and it suggests that the family of the skyline method is very suitable for the vector processors.

N	HBW	SKY	1PARSKYa	1PARSKYb	2PARSKYa			2PARSKYb			2PARSKYc			COLSOL
					DOF=3	DOF=5	DOF=7	DOF=3	DOF=5	DOF=7	DOF=3	DOF=5	DOF=7	
1050	105	1	2.11	2.25	2.32	2.39	2.36	2.36	2.49	2.45	2.45	2.59	2.46	2.49
	210	1	2.85	3.04	3.12	3.21	2.92	3.15	3.32	3.02	3.34	3.49	3.09	3.28
1470	105	1	2.12	2.26	2.32	2.39	2.36	2.37	2.50	2.46	2.45	2.60	2.47	2.50
	210	1	2.80	2.98	3.07	3.15	2.87	3.09	3.27	2.96	3.28	3.43	3.04	3.22
1995	105	1	2.12	2.25	2.32	2.40	2.36	2.37	2.50	2.46	2.46	2.60	2.47	2.50
	210	1	2.80	2.99	3.07	3.16	2.87	3.10	3.28	2.97	3.29	3.44	3.04	3.22
V-RATIO(%)		98.90	95.70	98.31	96.32	94.26	96.18	94.84	97.10	96.15	98.84	98.57	98.18	98.59

Table 2 Results of Problem 2

4. CONCLUDING REMARKS

Several variations of the skyline method are newly proposed in this paper, and their efficiency was investigated through a number of test problems. The numerical experiments can lead to following conclusions:
(1) The skyline method is originally suitable for the vector processor, and its family also fits to the machine.
(2) In order to fasten the computation of skyline methods the introduction of the inner products for matrices, 2-dimensional array for data storing, and the removal of unnecessary numerical operations through the factorization are necessary.
(3) In accordance with the increase of the number of equations and the half bandwidth, the efficiency of the proposed methods become higher, and we can fasten more than three times of the original skyline method.
(4) Skyline methods newly proposed in this paper are expected to be much faster when they are used on parallel computers.

All the numerical experiments in this paper are done using the supercomputer, SX-1E, of Data Processing Center of Okayama University.

REFERENCES

[1] Toshiro Miyoshi & Yuichiro Yoshida,"Finite element analysis of surface cracks by the supercomputer", Proc. of JSME, Vol.53 (1989), pp.255-260
[2] Toshiro Miyoshi,"The bench mark test for the three dimensional finite element codes for a supercomputer", Proc. of Symp. on Computational Methods in Structural Engineering and Related Fields, Vol.13, (1989), pp.129-132
[3] Toshiro Miyoshi & Naoki Takano,"General purpose high speed solver for supercomputers", Proc. of Symp. on Computational Methods in Structural Engineering and Related Fields, Vol.13, (1989), pp.145-148
[4] K.J. Bathe & E.L. Wilson,'Numerical Methods in Finite Element Analysis', Prentice-Hall (1976), Chapter 7

APPENDIX A : PARALLEL SKYLINE METHOD

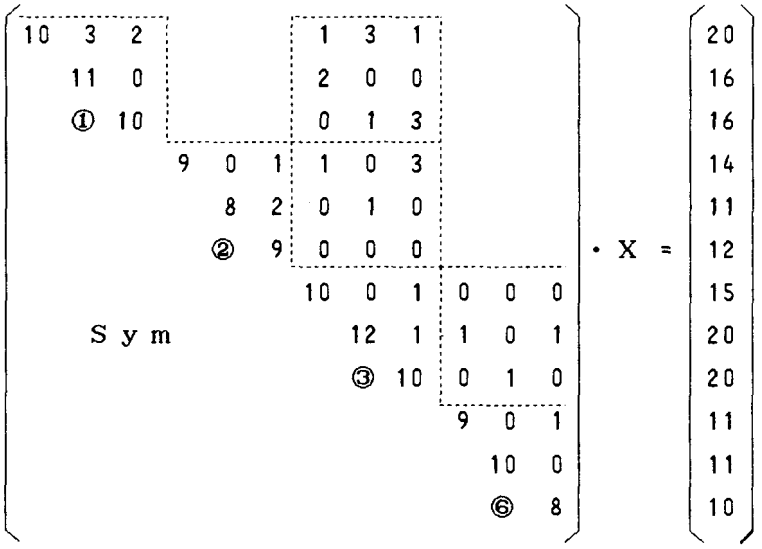
```
C*********************************************************************
C*     ----  Input  ----                                             *
C*                                                                   *
C*      A(3,KK1) : Coefficient Matrix for Parallel Skyline           *
C*     IADR(NN1) : Addresses of Diagonal Element Blocks              *
C*        B(N)   : Right - Hand - Side Vector                        *
C*                                                                   *
C*          N    : Number of Equations                               *
C*        KK1    : Total Number of Elements below Block Skyline     *
C*                                                                   *
C*     ----  Output  ----                                           *
C*                                                                   *
C*        X(N)   : Vector of Solutions                               *
C*                                                                   *
C*********************************************************************
      SUBROUTINE PARSKY(A,N,IADR,B,X,KK1,IHBW)
       IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(3,KK1),IADR(N+1),D(N),X(N),B(N)
C
      IHBW3=IHBW/3
      N3=N/3
C
      DO 1000 I=1,N3
      IDT=IADR(I+1)-IADR(I)-1
      IDT=IDT*3
      J1=3*IADR(I)
      ID=3*I-2
C
      S2=0.D0
      S3=0.D0
      S6=0.D0
      B1=0.D0
      B2=0.D0
      B3=0.D0
      K=ID
      DO 10 L=1,IDT
      K=K-1
      C1=A(1,J1+L)/D(K)
      C2=A(2,J1+L)/D(K)
      C3=A(3,J1+L)/D(K)
      S2=S2+C1*A(2,J1+L)
      S3=S3+C1*A(3,J1+L)
      S6=S6+C2*A(3,J1+L)
      B1=B1+C1*A(1,J1+L)
      B2=B2+C2*A(2,J1+L)
      B3=B3+C3*A(3,J1+L)
      A(1,J1+L)=C1
      A(2,J1+L)=C2
   10 A(3,J1+L)=C3
CC
      D(ID)=A(1,J1)-B1
      A(2,J1)=A(2,J1)-S2
      A(3,J1)=A(3,J1)-S3
      C=A(2,J1)/D(ID)
      B2=B2+C*A(2,J1)
      S6=S6+C*A(3,J1)
      A(2,J1)=C
```

```
CC
      D(ID+1)=A(2,J1-1)-B2
      A(3,J1-1)=A(3,J1-1)-S6
      C1=A(3,J1)/D(ID)
      C2=A(3,J1-1)/D(ID+1)
      D(ID+2)=A(3,J1-2)-B3-C1*A(3,J1)-C2*A(3,J1-1)
      A(3,J1)=C1
      A(3,J1-1)=C2
C
      M=MIN0(N3-I+1,IHBW3)
      DO 1100 J=2,M
      K1=I+J-1
      IJ=IADR(K1)+J
      ISTA=I-IADR(K1+1)+IJ
      IF(I.LT.ISTA) GO TO 1100
      IH=I-ISTA
      IH=3*IH
      IH=MIN0(IDT,IH)
      J2=3*IJ-3
      S1=0.D0
      S2=0.D0
      S3=0.D0
      S4=0.D0
      S5=0.D0
      S6=0.D0
      S7=0.D0
      S8=0.D0
      S9=0.D0
      DO 20 K=1,IH
C
      S1=S1+A(1,J1+K)*A(1,J2+K)
      S2=S2+A(1,J1+K)*A(2,J2+K)
      S3=S3+A(1,J1+K)*A(3,J2+K)
      S4=S4+A(2,J1+K)*A(1,J2+K)
      S5=S5+A(2,J1+K)*A(2,J2+K)
      S6=S6+A(2,J1+K)*A(3,J2+K)
      S7=S7+A(3,J1+K)*A(1,J2+K)
      S8=S8+A(3,J1+K)*A(2,J2+K)
   20 S9=S9+A(3,J1+K)*A(3,J2+K)
      A(1,J2)=A(1,J2)-S1
      A(2,J2)=A(2,J2)-S2
      A(3,J2)=A(3,J2)-S3
C
      AA2=A(2,J1)
      A(1,J2-1)=A(1,J2-1)-S4-AA2*A(1,J2)
      A(2,J2-1)=A(2,J2-1)-S5-AA2*A(2,J2)
      A(3,J2-1)=A(3,J2-1)-S6-AA2*A(3,J2)
C
      AA3=A(3,J1-1)
      AB3=A(3,J1)
      A(1,J2-2)=A(1,J2-2)-S7-AA3*A(1,J2-1)
     &          -AB3*A(1,J2)
      A(2,J2-2)=A(2,J2-2)-S8-AA3*A(2,J2-1)
     &          -AB3*A(2,J2)
      A(3,J2-2)=A(3,J2-2)-S9-AA3*A(3,J2-1)
     &          -AB3*A(3,J2)
 1100 CONTINUE
 1000 CONTINUE
C
```

```
C
      DO 1200 I=1,N3
      IH=IADR(I+1)-IADR(I)-1
      IH=3*IH
      J1=3*IADR(I)
      ID=3*(I-1)+1
      DO 80 K=1,IH
      B(ID)=B(ID)-A(1,J1+K)*B(ID-K)
      B(ID+1)=B(ID+1)-A(2,J1+K)*B(ID-K)
   80 B(ID+2)=B(ID+2)-A(3,J1+K)*B(ID-K)
      B(ID+1)=B(ID+1)-A(2,J1)*B(ID)
      B(ID+2)=B(ID+2)-A(3,J1)*B(ID)-A(3,J1-1)*B(ID+1)
 1200 CONTINUE
C
      DO 1250 I=1,N
      B(I)=B(I)/D(I)
 1250 CONTINUE
C
      DO 1300 I=N3,1,-1
      IH=IADR(I+1)-IADR(I)-1
      IH=3*IH
      J1=3*IADR(I)
      ID=3*(I-1)+1
      X(ID+2)=B(ID+2)+X(ID+2)
      X(ID+1)=B(ID+1)-A(3,J1-1)*X(ID+2)+X(ID+1)
      X(ID)=B(ID)-A(2,J1)*X(ID+1)-A(3,J1)*X(ID+2)+X(ID)
      DO 90 K=1,IH
      I1=ID-K
   90 X(I1)=X(I1)-A(1,J1+K)*X(ID)-A(2,J1+K)*X(ID+1)
     &     -A(3,J1+K)*X(ID+2)
 1300 CONTINUE
C
      RETURN
      END
```

APPENDIX B : AN EXAMPLE OF INPUT AND OUTPUT DATA FOR "PARSKY"

Equation



<INPUT DATA>

N = 12

A =

①			②			③									⑥					
0	0	10	0	0	9	0	0	10	0	0	1	0	2	1	0	0	9	0	1	0
0	11	3	0	8	0	0	12	0	0	1	0	1	0	3	0	10	0	1	0	0
10	0	2	9	2	1	10	1	1	0	0	3	3	0	1	8	0	1	0	1	0

IADR = (1 2 3 6 8)

B = (20 16 16 14 11 12 15 20 20 11 11 10)

<OUTPUT DATA>

X = (1 1 1 1 1 1 1 1 1 1 1 1)