NASA Contractor Report 181982
ICASE Report No. 90-8

ICASE

FACTORING SYMMETRIC INDEFINITE MATRICES ON HIGH-PERFORMANCE ARCHITECTURES

Mark T. Jones
Merrell L. Patrick

Contract Nos. NAS1-18107, NAS1-18605
January 1990

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Factoring Symmetric Indefinite Matrices on High-Performance Architectures

Mark T. Jones*and Merrell L. Patrick*†

Abstract

The Bunch-Kaufman algorithm is the method of choice for factoring symmetric indefinite matrices in many applications. However, the Bunch-Kaufman algorithm does not take advantage of high-performance architectures such as the Cray Y-MP. Three new algorithms, based on Bunch-Kaufman factorization, that take advantage of such architectures are described. Results from an implementation of the third algorithm are presented.

*Department of Computer Science, Duke University, Durham, NC 27706

†This research was supported by the National Aeronautics and Space Administration under NASA contract Nos. NAS1-18107 and NAS1-18605 and the Air Force Office of Scientific Research under AFOSR grant No. 88-0117 while the authors were in residence at ICASE. Additional support was provided by NASA grant No. NAG-1-466.

1 Introduction

The Bunch-Kaufman algorithm is considered one of the best methods for factoring full, symmetric, indefinite matrices [1], [2]. A modified version has been successfully used to factor sparse, indefinite matrices [3]. Recently, Bunch-Kaufman factorization has been shown to be the method of choice for a subset of banded, symmetric indefinite matrices [4].

The Bunch-Kaufman algorithm maintains the symmetry of the matrix during factorization and yields the inertia of the matrix essentially for free, an important consideration for eigenvalue calculations [1]. A drawback to the Bunch-Kaufman algorithm is its unsuitability for high-performance architectures. Herein, three new algorithms, based on Bunch-Kaufman factorization, are given for architectures such as the Cray Y-MP.

The technique of loop unrolling for vector architectures is discussed in section 2. In section 3, one of several variations of the Bunch-Kaufman algorithm is reviewed and the reason for its unsuitability for high-performance architectures is given. Three new algorithms for high-performance architectures are developed in section 4. Results showing the benefits of the third algorithm are given in section 5. Finally, a summary and description of future work is given in section 6.

2 Loop-Unrolling

Loop unrolling is a well known technique for improving performance on vector architectures. A loop is unrolled by restructuring it to allow more computation to take place at each step. A simple example of loop unrolling from [5] is given in Figure 1. The outer DO-loop has been unrolled to a depth of four. In the original program segment, three vector memory references were required for every two vector floating point operations. The ratio for the unrolled program segment is six vector memory references for every eight vector floating point operations. A significant decrease in the number of memory references has been achieved.

The reduction in the number of vector memory operations reduces the probability of delays due to memory latency times as well as the possibility of memory contention in a parallel computer [6].

Three other benefits of loop unrolling are described in [7]. The first is a

```
C Original program segment
      DO 20 J = 1, N2
        DO 10 I = 1, N1
          Y(I) = Y(I) + X(J) * M(I,J)
10      CONTINUE
20    CONTINUE
C In this example, the end condition if N2 isn't a multiple of four is ignored
      DO 20 J = 4, N2, 4
        DO 10 I = 1, N1
          Y(I) = Y(I) + X(J-3) * M(I,J-3) + X(J-2) * M(I,J-2) +
     c          X(J-1) * M(I,J-1) + X(J) * M(I,J)
10      CONTINUE
20    CONTINUE
```

Figure 1: Simple loop unrolling example

reduction in loop overhead because fewer incrementing and testing operations are required. This benefit can be reaped by any computer architecture.

For computers with segmented functional units, such as the CDC 7600, the higher ratio of floating point operations to overhead operations will allow a higher level of concurrency within a functional unit.

Computers with independent functional units, such as the Cray-1, benefit from greater concurrency between the functional units.

The optimal depth of loop unrolling is largely dependent on the target architecture. For example, if the independent functional units of a computer are kept busy with loop unrolling of depth p, then increasing the depth to $p+1$ will not result in increased concurrency among functional units.

In the simple example in Figure 1, the results of iteration j of the outer loop did not depend on results of previous iterations. Therefore, the outer loop was easily unrolled. If LDL^T decomposition is considered, however, each iteration of the outer loop depends on the previous iterations (see Figure 2). Unrolling the outer loop causes several pivot columns to be used simultaneously to update the remaining non-zeroes. For the algorithm to be correct, the first pivot column must be used to update the other pivot columns, then the second pivot column used to update the remaining pivot columns, and so forth. After all the pivot columns are updated, they are used to update the remaining non-zeroes. Conceptually, loop unrolling in LDL^T allows the use

```
1)      DO 10 I = 1, N
  C       Compute the pivot column
2)        DO 20 J = I+1, N
3)          V(J) = A(J,I)
4)          A(J,I) = A(J,I)/A(I,I)
5) 20     CONTINUE
  C       Update the remaining non-zeroes
6)        DO 30 J = I+1, N
7)          DO 40 K = J, N
8)            A(J,K) = A(J,K) - V(K)*A(J,I)
9) 40       CONTINUE
10) 30    CONTINUE
11) 10  CONTINUE
```

Figure 2: The LDL^T algorithm

of matrix-matrix operations rather than matrix-vector operations. A version of LDL^T unrolled to a depth of three is given in Figure 3.

Because three pivot columns are used to update the remaining non-zeroes in step 20, each time an element, $a_{j,k}$, is fetched six floating point computations are done, rather than just two as in step 8 of the original algorithm.

3 The Bunch-Kaufman Algorithm

The Bunch-Kaufman algorithm factors A, an $n \times n$ real symmetric indefinite matrix, into LDL^T while doing symmetric permutations on A to maintain stability, resulting in the following equation:

$$PAP^T = LDL^T. \tag{1}$$

Although several variations of the algorithm exist, the focus here is on algorithm D from [1] because it is the simplest to discuss. The methods described in section 4 are also applicable to Algorithm A described in [1], but not to Algorithm C (Algorithm B is mentioned in [1], but it is not described in detail).

The Bunch-Kaufman algorithm maintains stability by using 2x2 pivots combined with symmetric permutations to A when a 1x1 pivot is not stable.

```
   C Loop unrolled version
   C The end condition for handling N not divisible by 3 has been ignored
 1)      DO 10 I = 1, N, 3
   C        Compute the upper 3x3 triangle of the pivot columns
 2)        V1(I+1) = A(I+1,I)
 3)        A(I+1,I) = A(I+1,I)/A(I,I)
 4)        V1(I+2) = A(I+2,I)
 5)        A(I+2,I) = A(I+2,I)/A(I,I)
 6)        A(I+1,I+1) = A(I+1,I+1) - V1(I+1)*A(I+1,I)
 7)        V2(I+2) = A(I+2,I+1) - V1(I+1)*A(I+2,I)
 8)        A(I+2,I+1) = V2(I+2)/A(I+1,I+1)
 9)        A(I+2,I+2) = A(I+2,I+2) - V1(I+2)*A(I+2,I) - V2(I+2)*A(I+2,I+1)
   C        Update and compute all three pivot columns
10)        DO 20 J = I+3, N
11)          V1(J) = A(J,I)
12)          A(J,I) = A(J,I)/A(I,I)
13)          V2(J) = A(J,I+1) - V1(I+1)*A(J,I)
14)          A(J,I+1) = V2(J)/A(I+1,I+1)
15)          V3(J) = A(J,I+2) - V1(I+2)*A(J,I) - V2(I+2)*A(J,I+1)
16)          A(J,I+2) = A(J,I+2)/A(I+2,I+2)
17) 20     CONTINUE
   C        Use the 3 pivots columns to update the remaining non-zeroes
18)        DO 30 J = I+3, N
 9)          DO 40 K = J, N
20)            A(J,K) = A(J,K) - V1(K)*A(J,I) - V2(K)*A(J,I+1) -
     C                 V3(K)*A(J,I+2)
21) 40       CONTINUE
22) 30     CONTINUE
23) 10   CONTINUE
```

Figure 3: LDL^T unrolled to a depth of three

Because this paper will concentrate on 1x1 pivots, only stability for these pivots will be discussed in detail. A 1x1 pivot for element $a_{i,j}$ at step k takes the form

$$a_{i,j} = a_{i,j} - a_{i,k}a_{j,k}/a_{k,k}. \tag{2}$$

Let μ_k be the maximum of the absolute values of the uneliminated elements at step k. Step 2 of the algorithm (shown in Figure 4) finds the maximum element, λ, in the pivot column. By substituting μ and λ into equation 2, the bound on μ_{k+1} becomes

$$\mu_{k+1} \leq \mu_k + \lambda^2/ \mid a_{k,k} \mid \leq \mu_k(1 + \lambda/ \mid a_{k,k} \mid). \tag{3}$$

Step 4 ensures that a 1x1 pivot occurs if $\alpha < \mid a_{i,i} \mid /\lambda$, where the parameter α has been chosen to be 0.525 to maximize stability for Algorithm D [1]. By substituting α into equation 3,

$$\mu_{k+1} \leq \mu_k(1 + 1/\alpha). \tag{4}$$

Therefore, the bound on the growth of an element due to a 1x1 pivot is 2.905.

If the test in step 4 is failed, a subsequent row search and another stability test determines if a 2x2 pivot and a permutation are necessary.

The stability checks and possible permutations at each step of the Bunch-Kaufman algorithm prevent the use of the same type of loop unrolling that is used for LDL^T decomposition. Because the stability checks and permutations must be completed before a pivot column is computed, pivot columns cannot be grouped as they were in Figure 2 without invalidating the bounds on element growth.

4 New Algorithm

This section will develop three ways in which the Bunch-Kaufman algorithm can be modified to allow pivot columns to be grouped together in one step. Because each 2x2 pivot involves a permutation of A, it is not possible to group 2x2 pivots together. However, 1x1 pivots can be grouped with a 2x2 pivot if they follow the 2x2 pivot, allowing the permutation to precede the updating and computing of pivot columns. Each 2x2 pivot can be implemented using loop unrolling of depth 2. The general strategy in this section will be to try to group several 1x1 pivots into a single step in a stable fashion. The

```
1)  for i = 1, n
      begin
2)    λ = max_{j=i+1,n} | a_{j,i} |
3)    set r to the row number of λ
4)    if λ α < | a_{i,i} | then
      begin
5)      perform a 1x1 pivot
      end
      else
      begin
6)      σ = max_{j=i+1,n} | a_{r,j} |
7)      if α λ^2 < σ | a_{i,i} | then
        begin
8)        perform a 1x1 pivot
        end
        else
        begin
9)        exchange rows and columns r and i + 1
10)       perform a 2x2 pivot
11)       i = i + 1
        end
      end
12) end
13) if inertia is desired, then scan the D matrix
```

Figure 4: The Bunch-Kaufman Factorization Algorithm

strategies described in this section are applicable to Algorithms A and D, but not C from [1]. Algorithm C cannot be unrolled using these strategies because a permutation occurs at every step.

The first approach uses *pxp* pivots in much the same way as the 2x2 pivots in the Bunch-Kaufman algorithm. The update of a single element of A using a 2x2 pivot at step k is

$$a_{i,j} = a_{i,j} - \frac{(a_{i,k}a_{k+1,k+1} - a_{i,k+1}a_{k+1,k})a_{j,k} + (a_{i,k+1}a_{k,k} - a_{i,k}a_{k+1,k})a_{j,k+1}}{a_{i,i}a_{i+1,i+1} - a_{i+1,i}^2}. \tag{5}$$

To obtain a bound for element growth, first define at step k

$$\lambda_1 = max_{l=k+1,n} \mid a_{l,k} \mid \tag{6}$$

and,

$$\lambda_2 = max_{l=k+2,n} \mid a_{l,k+1} \mid . \tag{7}$$

If μ_k, λ_1, and λ_2 are substituted into equation 5, then

$$\mu_{k+2} \leq \mu_k(1 + \frac{\lambda_1^2 + \lambda_2^2 + 2\lambda_1\lambda_2}{\mid a_{i,i}a_{i+1,i+1} - a_{i+1,i}^2 \mid}). \tag{8}$$

The bound on growth of μ_{k+2} for a 2x2 pivot in Algorithm D is 8.526 [1]. Therefore, a 2x2 pivot can be performed if

$$1 + \frac{(\lambda_1 + \lambda_2)^2}{\mid a_{i,i}a_{i+1,i+1} - a_{i+1,i}^2 \mid} \leq 8.526. \tag{9}$$

This derivation is similar to the 2x2 pivot stability analysis in [1]. This test differs from the Bunch-Kaufman 2x2 test because it groups two potential 1x1 pivots into one step. The Bunch-Kaufman 2x2 test is used when a 1x1 pivot is not stable. The new test precedes step 4 of the Bunch-Kaufman algorithm in Figure 4. This approach has two drawbacks: 1) the formulae for bounding the growth due to a *pxp* pivot become increasingly complicated as p increases, and 2) the inertia calculation is no longer just a search down the diagonal, it requires the solution of many *pxp* eigenproblems.

The second approach updates the potential pivot columns one at a time and, after each update, applies the 1x1 pivot stability check to determine if

```
4a) if λ α < | a_{i,i} | then
    begin
4b)   compute the ith pivot column and use it to update column i + 1
4c)   λ_2 = max_{j=i+2,n} | a_{j,i+1} |
4d)   if λ_2 α > | a_{i+1,i+1} | then
      begin
4e)     compute the (i + 1)th pivot column and use it and column i to
        update column i + 2
4f)     λ_3 = max_{j=i+3,n} | a_{j,i+2} |
4g)     if λ_3 α > | a_{i+2,i+2} | then
        begin
4h)       use columns i, i + 1, and i + 2 to update the remaining
          non-zeroes
        end
        else
        begin
4i)       use columns i and i + 1 to update the remaining non-zeroes
        end
      end
      else
      begin
4j)     perform a 1x1 pivot
      end
    end
```

Figure 5: Approach Two for Loop Unrolling

further unrolling is possible. This test replaces steps 4 and 5 of the Bunch-Kaufman algorithm. An example of this test for up to three columns is given in *Figure 5*.

The third strategy uses an *a priori* approach to predict stability. The strategy predicts the stability of grouping p 1x1 pivots without updating each potential pivot column. Only the upper pxp triangle must be updated to bound element growth. From equation 4, for p successive 1x1 pivots to maintain stability, the maximum element growth must be bounded by

$$(1 + 1/\alpha)^p. \tag{10}$$

At step k, let

$$\lambda_2 = max_{j=k+2,n} \mid a_{j,k+1} \mid . \tag{11}$$

The bound on element growth for a 1x1 pivot is $(1 + \lambda/a_{k,k})$, therefore the bound on λ_2 after a 1x1 pivot is

$$\hat{\lambda_2} \leq \lambda_2(1 + \lambda/a_{k,k}). \tag{12}$$

Because the upper pxp triangle has been updated, a bound on μ_{k+2} for a second 1x1 pivot using column $k + 1$ is

$$\mu_{k+2} \leq \mu_{k+1}(1 + \hat{\lambda_2}/a_{k+1,k+1}). \tag{13}$$

By substituting the bound for μ_{k+1} into equation 13

$$\mu_{k+2} \leq \mu_k(1 + \lambda/a_{k,k})(1 + \hat{\lambda_2}/a_{k+1,k+1}). \tag{14}$$

In general, the bound on μ_{k+p-1} for p 1x1 pivots is

$$\mu_{k+p-1} \leq \mu_{k+p-2}(1 + \frac{\mu_{k+p-2}\lambda_{k+p-1}}{a_{k+p-1,k+p-1}}), \tag{15}$$

where

$$\lambda_{k+p-1} = max_{j=k+p,n} \mid a_{j,k+p-1} \mid . \tag{16}$$

Given the bound for $p - 1$ 1x1 pivots, the only new information necessary is the updated value of $a_{k+p-1,k+p-1}$ and λ_{k+p-1}. If the bound for $\mu_{k+(p-1)}$ is small enough then the p pivot columns are computed at the same time and all used at the same time to update the remaining non-zeroes. The a

priori strategy has two advantages over the second strategy. First, it allows all the pivot columns to be updated and computed in one loop. The second strategy requires the pivot columns to be updated and computed one after the other. Therefore, the *a priori* method reaps the benefits of loop unrolling in the pivot column calculation. Second, the *a priori* method can combine a pivot that fails the 1x1 pivot test with a 1x1 pivot that is very stable if the combination of the two meets the stability criterion. The second method is unable to combine the two pivots if one of the pivots fails the 1x1 stability test. Another benefit the *a priori* method reaps from this combination is avoiding the search for σ in step 6 of the Bunch-Kaufman algorithm. A disadvantage of the *a priori* method when compared with the second method, is the use of estimated bounds. In some cases these bounds are too pessimistic and thereby prevent the combining of 1x1 pivots when the combination is stable. Because the second method actually computes the pivot columns it does not have to use estimated bounds.

5 Results

A version of the *a priori* algorithm described in section 4 suitable for variable banded systems was implemented on a Cray Y-MP. The uniprocessor implementation allows loop unrolling up to depth six to take place. When the maximum depth of loop unrolling is fixed at one, this algorithm is identical to the Bunch-Kaufman algorithm. The Cray Y-MP is a register-to-register parallel/vector computer with up to eight processors. Each processor has independent, segmented functional units. An indefinite matrix that arose from an application in structural engineering was factored and the resulting triangular matrices were solved. The order of the matrix was 10,785 and the average bandwidth was 416. A significant reduction in factorization time and solution time for the resulting triangular systems, due to increasing depths of loop unrolling is shown in Figure 6. For this matrix, the combination of six pivot columns never met the stability criterion.

The benefits of the algorithm will vary depending on the architecture and on the matrix being solved. For example, the same implementation was executed on the Convex C-220, an architecture with independent, segmented functional units. An indefinite matrix arising from the same application was factored, but with an order of 6734 and an average bandwidth of 336. The

Depth	Factorization Time (sec.)	Speedup over Bunch-Kaufman	Triangular Solution Time (sec.)	Speedup over Bunch-Kaufman
1	16.3	1.00	0.32	1.00
2	13.4	1.22	0.23	1.39
3	12.9	1.26	0.21	1.52
4	12.6	1.29	0.21	1.52
5	12.8	1.27	0.20	1.60

Figure 6: Effects of different depths of loop unrolling on the Cray Y-MP

Depth	Factorization Time (sec.)	Speedup over Bunch-Kaufman	Triangular Solution Time (sec.)	Speedup over Bunch-Kaufman
1	71.77	1.00	1.30	1.00
2	50.99	1.41	1.05	1.24
3	46.85	1.53	0.98	1.33
4	45.37	1.58	0.96	1.35
5	45.15	1.59	0.97	1.34

Figure 7: Effects of different depths of loop unrolling on the Convex C-220

maximum speedup due to the *a priori* algorithm is shown in Figure 7 to be significantly better for this combination of architecture and matrix than for the combination in Figure 6.

Observing the number of times each depth of loop unrolling is utilized can be useful in determining the best maximum depth of loop unrolling for a particular application. Shown in Figure 8 is the number of times that each depth of loop unrolling was utilized when factoring the same matrix used in the Convex C-220 test. These results will, of course, be different for every matrix factored, but may be similar for matrices arising from the same application.

Maximum Possible Depth	No. of Depth 1	No. of Depth 2	No. of Depth 3	No. of Depth 4	No. of Depth 5	No. of Depth 6
1	6732	1	-	-	-	-
2	658	3038	-	-	-	-
3	516	1195	1276	-	-	-
4	543	992	429	730	-	-
5	561	874	439	432	276	-
6	561	874	439	432	276	0

Figure 8: Depths of loop unrolled achieved

6 Summary and Future Work

Three algorithms, each based on the Bunch-Kaufman algorithm, suitable for factoring symmetric indefinite matrices on high-performance architectures were given. The third algorithm, called the *a priori* strategy, was determined to be superior to the other two and was implemented on two high-performance architectures, the Cray Y-MP and the Convex C-220. The *a priori* algorithm was shown to be significantly faster than the Bunch-Kaufman algorithm.

The *a priori* algorithm is also suitable for implementation on parallel architectures because it allows the use of matrix-matrix operations, rather than the matrix-vector operations used in the Bunch-Kaufman algorithm. The use of the *a priori* algorithm will increase the ratio of computation to synchronization. The authors are currently implementing this algorithm on a parallel architecture.

Another possible application for the *a priori* strategy is the factorization of sparse matrices. A variant of the Bunch-Kaufman algorithm that includes pivoting to preserve sparsity is given in [3]. A modified version of the *a priori* strategy may improve the performance of this algorithm on high-performance architectures.

References

[1] J. R. Bunch and L. Kaufman, "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems," *Mathematics of Computation*, vol. 31, pp. 163–179, January 1977.

[2] V. Barwell and A. George, "A Comparison of Algorithms for Solving Symmetric Indefinite Systems of Linear Equations," *ACM Transactions on Mathematical Software*, vol. 2, pp. 242–251, September 1976.

[3] I. S. Duff, J. K. Reid, N. Munksgaard, and H. B. Nielsen, "Direct Solution of Sets of Linear Equations Whose Matrix is Sparse, Symmetric and Indefinite," *Journal of the Institute of Math and its Applications*, vol. 23, pp. 235–250, 1979.

[4] M. T. Jones and M. L. Patrick, "Bunch-Kaufman Factorization for Real Symmetric Indefinite Banded Matrices," Report No. 89-37, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA, 1989.

[5] J. J. Dongarra and S. C. Eisenstat, "Squeezing the Most out of an Algorithm in CRAY FORTRAN," *ACM Transactions on Mathematical Software*, vol. 10, pp. 219–230, September 1984.

[6] W. R. Cowell and C. P. Thompson, "Transforming Fortran DO Loops to Improve Performance on Vector Architectures," *ACM Transactions on Mathematical Software*, vol. 12, pp. 324–353, December 1986.

[7] J. J. Dongarra and A. R. Hinds, "Unrolling Loops in FORTRAN," *Software: Practice and Experience*, vol. 9, pp. 219–226, 1979.

NASA

Report Documentation Page

1. Report No	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181982 ICASE Report No. 90-8		

4. Title and Subtitle

FACTORING SYMMETRIC INDEFINITE MATRICES ON HIGH-PERFORMANCE ARCHITECTURES

5. Report Date

January 1990

6. Performing Organization Code

7. Author(s)

Mark T. Jones
Merrell L. Patrick

8. Performing Organization Report No.

90-8

10. Work Unit No.

505-90-21-01

9. Performing Organization Name and Address

Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665-5225

11. Contract or Grant No.

NAS1-18107
NAS1-18605

13. Type of Report and Period Covered

Contractor Report

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Final Report

16. Abstract

The Bunch-Kaufman algorithm is the method of choice for factoring symmetric indefinite matrices in many applications. However, the Bunch-Kaufman algorithm does not take advantage of high-performance architectures such as the Cray Y-MP. Three new algorithms, based on Bunch-Kaufman factorization, that take advantage of such architectures are described. Results from an implementation of the third algorithm are presented.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
symmetric indefinite matrices; Bunch-Kaufman algorithm; loop unrolling; Cray Y-MP	64 - Numerical Analysis 61 - Computer Programming and Software Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	15	A03

NASA FORM 1626 OCT 86