Program-Based Reduction of Memory Bank Conflicts: A Software Tool

Y.-C. Wu and E. L. Leiss*
Department of Computer Science
University of Houston
Houston Texas 77204-3475
coscel@cs.uh.edu

Abstract

Because the CPU of a computer usually processes instructions and data faster than they can be fetched into register from main memory, the memory cycle time is frequently the bottleneck of a computing system. Memory interleaving can be used to alleviate this problem. If references can be distributed over the memory banks, then successive accessing can be overlapped in a pipelined fashion, and a speed up of up to N can be achieved in an N bank system. Memory bank conflicts are caused if memory bank access requests are issued to a still busy bank; they can seriously degrade system performance. We outline a software tool, the Bank Conflict Reducer (BCR). BCR analyzes a given program and modifies the code automatically with the objective of reducing memory bank conflicts, thereby improving program performance.

1. Introduction

Main memory can be viewed as a collection of a large number of cells which in turn are organized into words. The information stored in a word can be moved into or out of the memory in one memory operation. Ordinarily, only one access to a word can be in progress at any one time.

The CPU of a computer usually processes instructions and data faster than they can be read from or written to main memory. Thus, the memory cycle time, the minimal time delay between the initiation of two memory operations, is often a major bottleneck of a system. In memory interleaving, the memory is divided into several independent memory modules or banks ([BK71], [D89], [H93], [K81], [LW86]); this technique can be used to close the gap between the speeds of the CPU and the main memory.

Ideally, in a system with N banks, a speed-up of N is achieved. Unfortunately, this peak performance often does not occur; in fact, several studies have shown that significant performance degradation occurs as the result of Bank busy conflicts ([D89], [H93], [RH93]). Bank busy conflicts are caused when a memory access request is issued to a still busy bank.

2. Interleaved Memory Architectures

If the main memory of a computer is structured as a collection of physically independent memory modules, memory access operations may proceed in more than one module at the same time. Thus, the average rate of transmission of words (bandwidth) from and to the memory system can be increased. This is called memory interleaving; it can be used both in uniprocessor systems and multiprocessing systems. The well known CRAY X-MP family is an example for a system with interleaved memory ([CS86], [EE93], [L86]).

* Research supported by Sandia National Laboratories under contract AJ-1289.

The distribution of storage addresses across separate modules of memory can be low-order or high-order ([D89], [H93], [HL91]). Different ways of assigning linear addresses result in different memory organizations. Assume that an address consists of $K=k_1+k_2$ bits.

Low-order (fine) interleaving: The low-order k_2 bits are used to identify the bank number, and the high-order k_1 bits denote the address in a bank. Low-order interleaving spreads contiguous memory locations across consecutive memory banks.

High-order (coarse) interleaving: The high-order k_1 bits are used to identify the bank number, and the low-order k_2 bits imply the address in a bank. Therefore, contiguous memory locations are assigned to the same memory banks.

Many programs reference consecutive elements of vectors in memory. In this case, low-order interleaving is more suitable than high-order interleaving. Thus, in the following discussions, we only assume low-order memory interleaving.

3. Theory Underlying the Bank Conflict Reducer

Increasing the bandwidth of the memory system is probably the most attractive property of interleaved memory systems. However, poor program design or an inefficient compiler can dramatically degrade performance. To simplify our analysis, we make some assumptions and define some notation.

System Assumptions

- Only the DO loops are analyzed.
  In most scientific code, DO loops consume the most CPU time. This is in particular true for memory accesses. The DO loop body is repeated several times while increasing or decreasing the loop index.
- Only in-core programming is considered ([L95]).
  This implies that the available main memory is large enough to hold all data sets associated with the execution of the program at the same time. If this condition is violated, other considerations (in particular, page fetching from disk) may become more important than bank conflicts.
- As the vectors are declared, all data are loaded into main memory only one time.
  Since we are only interested in the relative address, we may assume that the first element of the first declared vector is mapped to memory bank 0.
- Low-order interleaving is assumed.

Notation

- ARS: The ordered list of array references in a DO loop.
- |ARS|: The number of arrays in ARS.
- D_{array} or $D_{sequence}$ in ARS: The stride of an array. It is the number of words separating consecutive accesses to the array.
- S_{array} name or $S_{sequence}$ in ARS: The start bank of an array. It is the memory bank number of the first referenced element of that array.
- MB: The number of memory banks; they are numbered 0, 1, 2, ..., MB-1.
- CC: The memory clock cycle time, the minimum time delay required between two successive memory operations on the same bank.
- $\{S\}_{MB}$: The elements of the set S (of integers) are calculated modulo MB.

3.1 Modified Conflict-Free Conditions

Our goal is to reduce the memory bank conflicts of a given program. Ideally, we will never encounter any bank conflict in DO loops; that situation is called conflict free (CF). There are two types of CF situations, disjoint access sequences and non-disjoint access sequences, as stated in ([OL85], [RH93]) for two or more vector stream accesses. We adapt the definitions to our analytic model.

Definition 3.1: When a DO loop body is executed, the CPU continuously issues memory access requests for accessing the elements of some array. We call these consecutive memory access requests the access stream.

Definition 3.2: The return number R of the access stream of some array with stride D is the number of banks accessed by the access stream before it accesses the same bank again; it is given by [OL85]

$$R = MB / \gcd(MB, D).$$

Definition 3.3: A self conflict is caused by accessing a busy bank which is still in the memory cycle time of the previous access of the same access stream. An array will encounter a self conflict iff $R*|ARS| < CC$.

In a disjoint access sequence, the access stream of each array will never visit the same memory bank again. Conflicts are caused by accessing busy banks; thus, disjoint access sequences always yield a CF situation.

Proposition 3.4 ([OL85]): Assume $|ARS| = n$ and that no array encounters a self conflict. By giving consecutive bank numbers to the start bank of the n arrays, disjoint access sequences are achieved if $\gcd(M, D_1, D_2, ..., D_n) \geq n$.

Note: The condition $\gcd(M, D_1, D_2, ..., D_n) \geq n$ in Proposition 3.4 is only sufficient but not necessary to achieve disjoint access sequences. To see this, consider the following situation: MB=30, CC=4, and ARS=(A,B,C). The strides of A, B, and C are 6, 10, and 25, respectively. Assume $S_A=0$, $S_B=1$, and $S_C=2$. Then $\gcd(30,6,10,25)=1<3$, but the access sequences are disjoint.

Nondisjoint access sequences connote that the access streams will visit the same memory bank at least once.

Theorem 3.5: Let the arrays A and B correspond to two nondisjoint sequences and set $f=\gcd(MB,D_A,D_B)$. Then defining $S_A=0$ and $S_B=[(\lfloor CC/2 \rfloor + 1)*D_A] \bmod MB$ will lead to conflict free access if and only if $\gcd(MB / f, (D_B-D_A) / f) \geq 2*(\lfloor CC/2 \rfloor + 1)$.

Proof: It follows that of Theorem 3 in [OL85]. In view of the page limit, we refer the reader to [W95].

The up-shot of Theorem 3.5 is that under certain conditions, a CF situation is achievable. Unfortunately, the probability that the conditions in Proposition 3.4 and Theorem 3.5 are satisfied is low. As the number of referenced array increases, this probability decreases ([RH93]).

3.2 Common Methods Used to Reduce Bank Conflict

As mentioned in [L95], there are several ways which can be used to avoid certain bank conflicts in conjunction with vector strides. These include the following.

(1) Using a prime number as the number of memory banks. The interleaved scheme is optimal whenever the stride D of the access is relatively prime to the number of memory banks. If MB is prime, all access strides, except multiples of MB, are relatively prime to MB. However, almost everything in modern computers is a power of 2 ([HJ87], [R85]).

(2) Having a very large number of memory banks, at least as large as the product of the number of cycles per memory access times the largest stride. This is a very expensive solution.

(3) Using different mapping schemes. Several mapping schemes, such as the dynamic storage scheme ([HL91]), the skewed scheme ([HJ87]), and the XOR scheme ([H91]), differ from the conventional low-order or high-order and may reduce conflicts ([LSYT93]). Unfortunately, none of these methods is under the control of the programmer.

(4) When the dimensions of matrices are declared, the programmer can add a row or column to avoid an effective stride that is a power of two. The new matrices are slightly larger than the original ones, but may yield much better performance for the memory system.

Example 1: Let MB = 8, CC = 4, and assume that the data are mapped to main memory by the row-major mapping scheme. Assume the matrix is originally declared as A(0:7,0:7). Here, consecutive accesses along a column always encounter memory bank conflicts. If instead the matrix is declared as A(0:7, 0:8), the situation is very different as there does not exist any conflict any longer.

(5) Choosing better start banks for each array. The start bank of an array can seriously affect the access streams. This is because the stride of an array is a constant.

Example 2: Given MB=16, CC=4, and ARS=(A,B) with D_A=D_B=1. For S_A=S_B=0, a conflict occurs each time we access an element of B. However, if S_A=0 and S_B=3, the situation is very different: no conflicts occur. Thus, the second assignment is about four times faster than the first.

We developed a software tool that will automatically reduce memory bank conflicts; it is called the Bank Conflict Reducer (BCR) and consists of two parts, the BCR-counter and the BCR-reducer. The BCR-counter is a compile time counter which determines the number of conflicts and the delay time caused by the memory bank conflicts (called conflict information) of a target file (program). The BCR-reducer uses the fourth and fifth methods described above to reduce the memory bank conflicts. The fourth method can reduce the self conflict problem of some array by adding a row or a column to that array, and the fifth method can be used in general types of conflicts.

3.3 How to Find the Cycle?

The basic assumption of the BCR-counter is as follows: Since the strides of referenced arrays are constant, the access streams are not random. More specifically, it is possible to find a smallest interval in the access streams so that the conflict information of the entire DO loop is represented by the conflict information within that interval. Moreover, the length of the interval is independent of the number of times the loop is iterated; thus, the length of the interval is independent of the execution time.

Definition 3.6: The integer cycle is the minimum number of iterations of a DO loop which have to be executed to get the full conflict information.

The access stream of an array can be divided into several consecutive subaccess streams. The first subaccess stream consists of the accessed bank numbers of the first to the cycleth iteration, the second subaccess stream consists of the accessed bank numbers of the (cycle+1)th to the (2*cycle)th iteration. The remaining subaccess streams are formed in a similar way. The relative bank numbers of each subaccess stream will be equal.

Theorem 3.7: Let |ARS|=n. Then

$$\text{cycle} = MB / \gcd(MB, D_2-D_1, D_3-D_2, \ldots, D_n-D_1).$$

Proof: In view of the page limit, we refer the reader to [W95].

Example 3: Given MB = 16, CC = 4 and ARS = (A, B, C). Let S_A=0, S_B=1, S_C=2.

```
DO I = 1, 100, 2
   C(5*I) = A(I) + B(3*I)
END DO I
```

By Theorem 3.7, cycle is equal to 4. Fig. 3.1 shows the access streams. We find that the relative bank numbers of every subaccess stream are the same.

Every cycle iterations, the subaccess streams have the same relative bank numbers. We first obtain the sub-conflict information by executing the DO loop body cycle times. Then we determine the full conflict information, without executing the entire DO loop, by multiplying the sub-conflict number and the sub-delay time by the total number of iterations, divided by cycle. That is why the BCR-counter is a compile time counter.

Determining the Bank Repeat Number

We use the fourth and fifth methods to reduce the memory bank conflicts. The fourth method can be used to solve the self conflict problem and avoid potentially conflict-generating factors, namely powers of two. On the other hand, the fifth method works well after we eliminate the potential for conflicts, strides of powers of two, from the DO loop. When applying the fifth method, we first add 1 to the start bank of some array, and keep the others unchanged; second, we use the BCR-counter to determine the conflict information of this new start bank assignment. Finally, we compare the conflict information of each combination. The assignment that generates the least number of conflicts (i. e., the least delay) will be chosen.

There is an obvious problem of efficiency with this approach: the number of combinations of start banks could be very large. For example, if MB=64 and ARS= (A,B,C), there are 262,144 kinds of combination for all of S_A, S_B, and S_C ranging from 0 to 63. The number of combinations can be reduced by assuming that the start bank of the array first referenced is always equal to 0. In other words, the relative bank numbers of the access streams for arrays A, B, and C with $S_A=k_1$, $S_B = k_2$ and $S_C = k_3$ are exactly those for $S_A=0$, $S_B = k_2-k_1$, and $S_C = k_3-k_1$.

Assume that the array A is referenced prior to the array B. If we keep the start bank of array A constant and add 1 to the start bank of array B each time, the conflict pattern of the access streams will be repeated after adding some number to the start bank of array B. The conflict patterns of different combinations are considered equal if, within cycle iterations, the conflicts in the access streams of the start banks combination (S_A, S_B) are just shifted up or down as they occur in the access streams of the start bank combinations (S_A, S_B+k). The bank repeat number, or BRN for short, of the array B is the minimum value of k for which this holds.

Theorem 3.8: If the array A is referenced prior to the array B and |ARS| = 2, the bank repeat number of B is gcd(MB, D_A-D_B).
Proof: In view of the page limit, we refer the reader to [W95].

Example 4: Given MB=12 and CC=4. Let D_A=5 and D_B=2. The graphical representation of the access streams of each combination is shown in Fig. 3.2. The conflict patterns for $S_A=S_B=0$ and $S_A=0$, $S_B=3$ are the same, since the conflict in the access streams for $S_A=S_B=0$ is just shifted one row down. However, the conflict patterns are not the same for $S_A=S_B=0$ and $S_A=0$, $S_B=2$. Although they have the same number of conflicts and the same delay time, the direction of the conflicts (0 -> 0; S_A -> S_B) and (8 -> 8; S_B -> S_A) is not the same. The BRN in this example is equal to 3.

The conflict pattern will be repeated after adding the number BRN to the start bank of the later referenced array. By definition of the conflict pattern, the same conflict pattern can have the same number of conflicts and the same delay time; thus, we can reduce the number of combinations by reducing the interval of the later referenced array from MB to BRN.

Unfortunately, Theorem 3.8 cannot be generalized to |ARS| > 2, because some conflicts may be avoided by the delay time of the previous conflicts, as the following example shows.
Example 5: Given MB=12 and CC=4, let D_A=1, D_B=2, and D_C=5. By Theorem 3.8, the BRN of array C is 4. We can see this from Fig. 3.3; the conflicts numbered 3 and 4 are shifted 2 rows down from Fig. 3.3a to Fig. 3.3b. However, note the conflict numbered 4 in Fig. 3.3a; this conflict does not really occur because of the delay of the conflict numbered as 1. On the other hand, in Fig. 3.3b, previous conflicts do not affect conflict 4 and conflict 4 occurs.

4. Implementation of the Bank Conflict Reducer

4.1 Assumptions and Restrictions

About the Target File

(1) A valid target file contains only declarations and DO loops.
(2) Statements are present only within the innermost DO loops.
(3) At most 10 arrays can be declared in the file.
(4) At most 4 DO loops are nested in a group. Furthermore, at most 3 groups of nested DO loops are allowed in the file.
(5) Expressions representing array indices contain only the following operations: addition, subtraction, or multiplication. The expressions may involve integers together with variables and must be linear functions of the indices.

Note: Restrictions (3) and (4) can be changed by modifying the definition part of programs counter.c and reducer.c.

4.2 Mapping Functions of Row- and Column-Major Schemes

An array A is declared by $A(L_1:U_1; L_2:U_2; ...; L_n:U_n)$ and $S_A = k$.
For both row- and column-major mapping, the bank number for the element $A(I_1,I_2,...,I_n)$ is given by

$$\{addrA(I_1,I_2,...,I_n)\}_{MB} \qquad (4.2.1)$$

If the accesses to array elements are not random (by assumption, the strides of the referenced arrays are constants), the bank number can be determined in a much easier way. Define Dim-stride(k) to be the difference of the bank numbers of two consecutive elements in dimension k of the array. For our n-dimensional array A, Dim-stride(k) is given by

$$\left\{ \prod_{i=k+1}^{n} (U_i-L_i-1) \right\}_{MB}$$

if the row-major mapping scheme is performed; for column-major mapping, Dimstride(k) is given by

$$\left\{ \prod_{i=1}^{k-1} (U_i-L_i-1) \right\}_{MB}$$

Real-stride(I) is the difference of the bank numbers of two consecutive accesses as the loop variable I increases by its stride; for the array A it is determined by

$$\{ \text{Dim-stride(k)*loop\_I\_stride*array\_A\_stride} \}_{MB}$$

where k corresponds to the sequence of I in the referenced array A.

With the functions Dim-stride(k) and Real-stride(I) defined as above, the bank number of the next access of array A can be computed as the bank number of the previous access plus Real-stride(I) of array A as the loop I increases by its stride. Thus, we just need to compute the bank number of the first access by (4.2.1), then the bank numbers of subsequent accesses can be obtained by adding a constant.

4.3 Special Modifications

4.3.1 If (Cycle*|ARS|) Is Less Than cycle

Consider the following example.
Example 6: Given MB=16, CC=4, and S_A=0.

```
DO I = 1, 100
   A(I) = A(I+1)
END DO I
```

By Theorem 3.8, cycle is 1. But in fact, the cycle iterations are not enough to reflect the entire conflict information of the DO loop.

If we consider cycle=1, the Total Conflict Number is 100*1=100. However, for cycle=2, we see that memory bank 2 will not encounter conflicts. It is reasonable to keep the product cycle*|ARS| greater than or equal to clock cycle time. Therefore, we make this adjustment if cycle*|ARS| is less than the clock cycle time.

4.3.2 Different Conflicts Within Two Special Intervals

Consider the following example.
Example 7: Given MB = 12, CC = 4.

```
A(1:105); B(1:300)
DO I = 2,100
   A(I) = A(I+1) + A(I-1) + B(3*I)
END DO I
```

 The access streams of this DO loop are shown in Fig. 4.1. We find that the conflict number in the first cycle iterations is 1 more than that of the second cycle iterations, as shown in Figures

4.1a and 4.1b. The reason for this is that the previous conflict delay, such as 6 -> 6, shown in Fig. 4.1a, frees the next conflict, 8->8, shown in Fig. 4.1a. Except for the first cycle iterations, the first conflict of the following cycle iterations is freed by the delay of the previous cycle iterations.

Therefore, we have to compare the conflict of the first CC cycle time with the last CC cycle time. If the numbers of conflicts are different in this two intervals, we have to make adjustments of the number of conflicts for the following cycle iterations.

4.3.3 About BRN

Theorem 3.8 is true only for |ARS| = 2 and if the total interval is divisible by the cycle of the innermost DO loop. Consider Example 4 again. We know already that the conflict pattern is the same for S_A=S_B=0 and S_A=0, S_B=3. However, if (end - begin + 1) modulo the cycle of the innermost DO loop is 1, as is the case for S_A=S_B=0, the conflict (0->0) will occur. But if S_A=0, S_B=3, the conflict (5->5) will not occur. Therefore, even though the conflict patterns of S_A=S_B=0 should be equal to that of S_A=0, S_B=3, the conflict numbers can be different.

Thus, we consider it as a special case when these two conditions are satisfied. If they are satisfied, Theorem 3.8 will be used to determine BRN; otherwise, BRN is taken to be MB.

5. Experimental Results

We present some of our experimental results in this section. For additional results, see [W95]. We will vary the number of memory banks MB and the clock cycle CC to observe their influence on the total number of conflicts (CN) and the percentage of delay time (DP) caused by the memory bank conflicts.

Example 8:

```
A(1:160,1:160)
B(1:160,1:160)
C(1:160,1:160)
DO I = 1,160
   DO J = 1,160
      DO K = 1,160
         C(I,J) = A(I,K)*B(K,J)
      END DO K
   END DO J
END DO I
```

Matrix multiplication is one of the most frequently used array algorithms in computer programming. It is obvious that the array C will always encounter the self conflict problem if the clock cycle is equal to 4. Therefore, we use CC=4, as well as CC=3 which will not suffer from the self conflict problem, to check the conflict results.

We draw the following conclusions from Figs. 5.1.1 and 5.1.2.

(1) It is reasonable that the number of conflict and delay time percentage for both before and after BCR for any MB with CC=3 are always lower than those with CC=4. The average of the delay percentage with CC=3 is 12.4 % and that with CC=4 is 46.67 %.

(2) The solid curves are always lower than or equal to the dash curves in Figs, 5.1.1 and 5.1.2. Furthermore, the solid curves are much smooterh than the dashed curves. This implies that the BCR can keep the performance of a program stable, even in the worst case, such as MB=10 or 16.

6. Conclusions

A software tool, the Bank Conflict Reducer (BCR), which automatically reduces bank busy conflicts was developed and implemented. BCR frees the programmers from the burden of thoroughly understanding complicated memory architecture and data mapping issues. Other

methods can be used to reduce memory bank conflicts. In past years, much research concentrated on different mapping schemes, such as dynamic storage schemes, the skewed scheme, and the XOR scheme, with less emphasis on software issues in resolving the problem of bank conflicts. Our method can be combined with these approaches.

Bibliography

[BK71] P. Budnik and D. J. Kuck, "The Organization and Use of Parallel Memories", IEEE Trans. Comput., vol C-20, 1566-1569, Dec. 1971.

[BM68] G. Birkhoff and S. MacLane, "A Survey of Modern Algebra", New York: Macmillan, 1968.

[CS86] T. Cheung and J. E. Smith, "A Simulation Study of the CRAY X-MP Memory System", IEEE Trans. Comput., vol C-35, 613-622, July. 1986.

[D89] A. L. DeCegang, "Parallel Processing Architectures and VLSI Hardware", Volume 1, Prentice-Hall, Inc., 1989.

[EE93] S. Edirisooriya and G. Edirisooriya, "Enhancing Vector Access Performance in CRAY X-MP memory System", IEEE Comput. Soc. COMPCON Spring 93, 569-576, Feb. 1993.

[H91] D. T. Harper III, "Block, Multistride Vector, and FFT Accesses in Parallel Memory Systems", IEEE Trans. Comput., vol C-2, 43-51, Jan. 1991.

[H93] K. Hwang, "Advanced Computer Architecture", McGraw-Hill, Inc., 1993.

[HJ87] D. T. Harper III and J. R. Jump, "Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme", IEEE Trans. Comput., vol C-36, 1440-1449, 1987.

[HL91] D. T. Harper III and D. A. Linebarger, "Conflict-Free Vector Access Using a Dynamic Storage Scheme", IEEE Trans. Comput., vol C-40, 276-283, Mar. 1991.

[HVZ90] V. C. Hamacher, Z. G. Vranesic and S. G. Zaky, "Computer Organization", McGraw-Hill , Inc., 1990.

[K81] P. M. Kogge, "The Architecture of Pipelined Computers", McGraw-Hill, Inc., 1981.

[L86] C. Lazou, "Supercomputers and Their Use", Oxford University Press, New York, 1986.

[L95] E. L. Leiss, "Parallel and Vector Computing: A Practical Introduction", McGraw-Hill, Inc., 1995.

[LMB92] J. R. Levine, T. Mason, and D. Brown, "Lex & Yacc", O'Reilly & Associates, Inc., 1992.

[LSYT93] K. Lam, W. Siu, J. Yin and K. Tse, "A New Storage Scheme for Conflict-Free Vector Access", IEEE, vol. 1, 32-35, Oct. 1993.

[LW86] A. G. Lippiatt and G. G. Wright, "The Architecture of Small Computer Systems", Prentice-Hall International UK LTD., 1986.

[OL85] W. Oed and O. Lange, "On the Effective Bandwidth of Interleaved Memories in Vector Processor Systems", IEEE Trans. Comput., vol C-34, 949-957, Oct. 1985.

[R85] A. Ranade, "Interconnection Networks and Parallel Memory Organizations for Array Processing", Proc. Int. Conf. Parallel Processing, 1985, 41-47.

[RH93] R. Ram and J. P. Hayes, "Reducing Interference Among Vector Accesses in Interleaved Memories", IEEE Trans. Comput., vol. C-42, 471-483, Apr. 1993.

[W95] Y.-C. Wu, Automatic Reduction of Memory Bank Conflicts, M. S. Thesis, Dept. of Computer Science, Univ. of Houston, Houston, Texas, Dec. 1995.

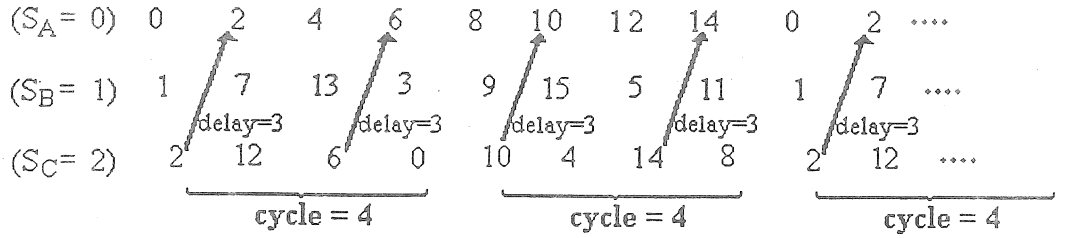

Fig. 3.1: A subaccess stream consists of 4 times iterations.

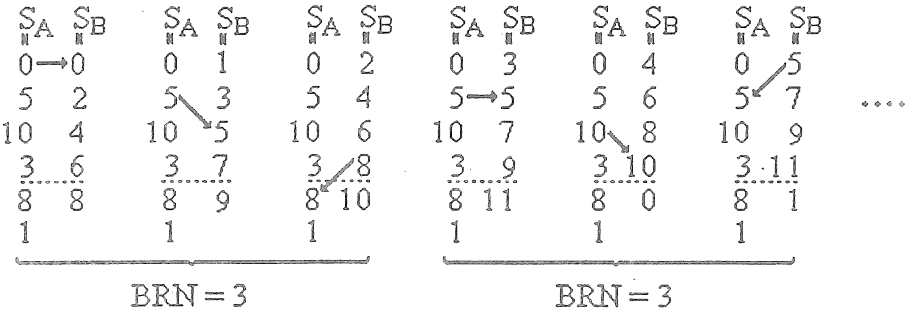

Fig. 3.2: For D_A=5 and D_B=2, BRN=gcd(12,2-5)=3.

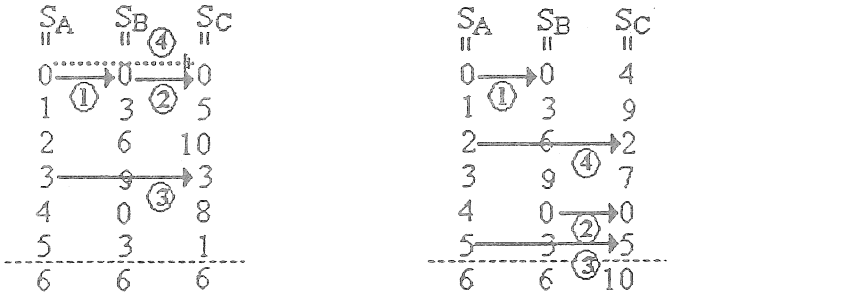

Fig. 3.3 a: S_A=S_B=S_C=0. b: S_A=S_B=0 and S_C=4.

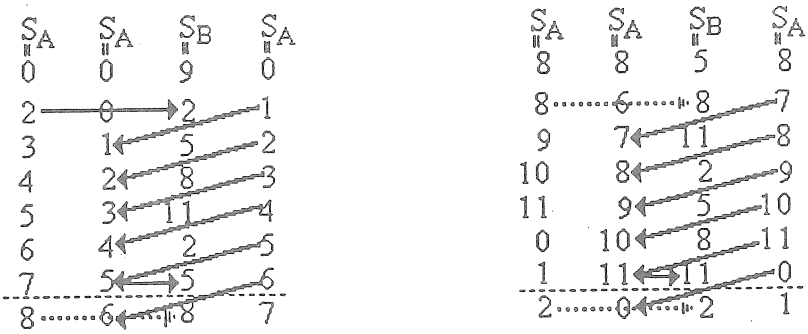

Fig. 4.1 a: The first cycle iterations. b: The second cycle iterations.

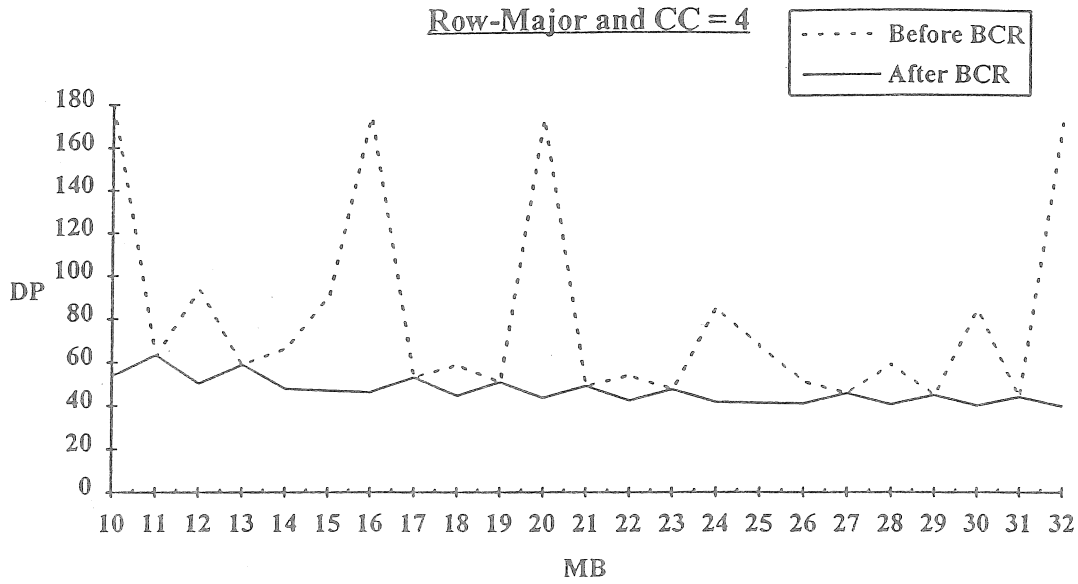

Fig. 5.1.1: The delay percentage curves for CC = 4.

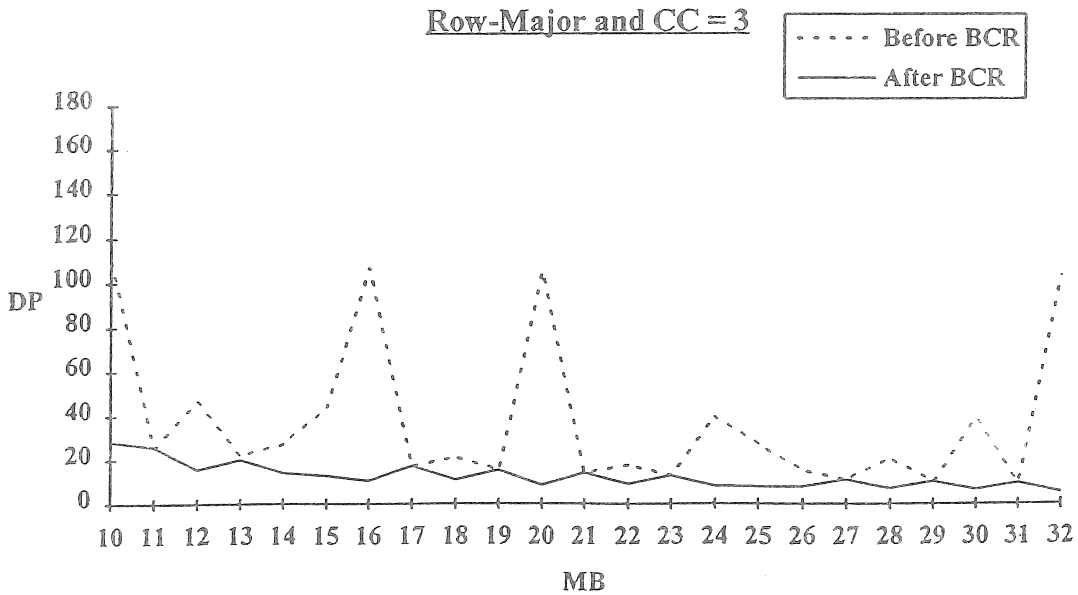

Fig. 5.1.2: The delay percentage curves for CC = 3.