Reducing Inter-Vector-Conflicts in Complex Memory Systems

A. M. del Corral & J. M. Llabería
Department d'Arquitectura de Computadors
Universitat Politècnica de Catalunya. Barcelona (Spain)
e-mail: anna@ac.upc.es

Abstract

In vector processors, the concurrent memory access of several vector streams causes inter-conflicts between references. In a complex memory system where several memory modules are mapped in every bus the number of conflicts increases because the bus must be shared by vector streams. In addition to the memory module access conflicts, bus access conflicts can appear, and also couplings of both type of conflicts. This paper proposes an access order to the vector elements and a mapping model of memory modules in buses that reduce the inter-conflicts in the steady-state.

Keywords: Complex Memory System, Vector Stream, Inter-Conflicts, Inter-Complex Conflicts, Stride.

1 Introduction

The memory module reservation time is, in general, much longer than the processor cycle time. Therefore, the performance of a vector processor is strongly related to the bandwidth of a memory subsystem (number of accesses per cycle). The most common memory system consists in multiple memory modules that are accessed independently in order to obtain the necessary bandwidth; after an initial latency, a datum can be obtained every cycle. However, memory module conflicts appear when two or more references are concurrently issued to the same memory module or, a busy memory module is referenced.

The most common data structures in vector processors are vectors stored in consecutive positions in memory. Vector instructions that access the memory specify the initial memory address of the vector and the distance between vector elements to be accessed; we call stride to this access pattern. In the present paper we consider regular access patterns; gather/scatter operations are not studied.

Memory vector instructions with a regular access pattern generate periodical conflicts as these type of instructions generate periodical streams of references (vector stream). In this paper, our interest is centered on the reduction of conflicts between references of different vector streams concurrently accessing the memory system (inter-conflicts).

In complex memory systems where several memory modules are mapped in every bus [1, 2], the interconnection network (between processor and memory modules) cost is decreased as the number of independent access paths to memory modules is reduced. This causes the appearance of inter-conflicts within the interconnection network, as well as producing memory module inter-conflicts. There are also situations where both conflicts are coupled. Because of these causes, the performance of the memory subsystem is lower than the highest achievable.

Our proposal to eliminate interconnection network and memory module conflicts consists of: a) defining a new order to access vector stream elements, and b) applying a mapping of memory modules in buses (sections). The relative order to access the memory modules and sections is access pattern independent; the stride and the first memory module referenced determine the set of memory modules and corresponding buses to be accessed.

As an indication of the benefit of the proposal, the simulation of the concurrent access of 4 odd strided vector streams using the classical order in a memory system like *CRAY X-MP* [2] *(16* memory modules, *4* sections and a memory cycle of *4* clock cycles*)* reaches 1.465 operations per cycle, while the simulation of the same 4 vector streams using the order and the memory module mapping proposed in this paper, reaches 3.908 operations per cycle.

In the present paper, section 2 outlines the architecture model, on which the study is based; the interaction between vector streams in a complex memory system is studied in section 3; section 4 presents the proposal we develop in this paper to avoid conflicts; and finally, section 5 deals with the performance of the proposal.

2 Architecture Model

The memory subsystem consists of $M = 2^m$ memory modules, with a memory cycle of $n_c = 2^c$ clock cycles. Modules are connected to *MP* memory ports through an interconnection network. Two buffers are provided for every memory module, one at the module input and the other at its output.

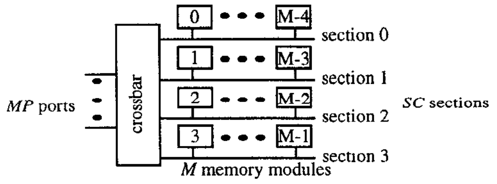


Figure 1. Complex Memory System. Interleaved Address Mapping.

The architecture model of a complex memory system like *CRAY X-MP*[2] is shown in figure 1. To reduce the number of access paths to the memory subsystem the memory modules are distributed into *SC* ($SC = 2^{sc}$) sections; we suppose the number of memory modules to be multiple of the number of sections and $SC \leq M$. A memory module request occupies during one cycle the section path where the module is located.

One way to organize the memory modules in a complex memory system is the interleaved mapping, as in *CRAY X*, where *module* = $A_i\, modM$, *section* = *module modSC* and *offset* = $\lfloor A_i/M \rfloor$.

The interleaving function ($module = A_i modM$) which maps the address into memory modules has a period M. The interleaving function ($section = module\ modSC$) which maps memory modules in sections has a period SC.

Definition 1: A vector stream $A = (A_0, S, VL)$ is a set of references to memory modules such as $\{A_i | A_i = A_0 + i \times S, (0 \le i < VL)\}$. It is characterized by the address of the first element (A_0), the distance between two consecutive elements (*stride=S*), and the vector length (*VL*). When the length is not relevant a vector stream is specified as $A = (A_0, S)$.

The strides are classified into different families.

Definition 2: The stride family s includes vector streams with strides $S = \sigma \times 2^s$ with an odd σ, as in [3].

For a vector stream with a stride $S = \sigma \times 2^s$ its period is $P_s = M/gcd(M, S)$, and its references are distributed among P_s memory modules.

Definition 3: The memory module set (*MMS*) of the vector stream $A = (A_0, S)$ is the set of all the memory modules accessed by the vector stream $A=(A_0, S, P_s)$,

$$MMS = \{m_i | m_i = (A_0 + i \times S) modM,\ 0 \le i < P_s\}\ .$$

Definition 4: We define stride subfamily (SF_s^{m0}) as a subset of the stride family s:

$$SF_s^{m0} = \{A = (A_0, S); S = \sigma \times 2^s | A_0 modgcd(M, S) = m0\}\ .$$

That is, a stride subfamily is the set of all the vector streams of a stride family that have the same *MMS*.

Definition 5: The classical order employed to access the memory uses the expression $A_i = A_{i-1} + S$ to reference the vector stream element i after the element i-1. Thus, using the classical order, the sequence of memory references is:

$$SR = (A_0, \ldots, A_0 + k \times S, \ldots, A_0 + (VL - 1) \times S)\ .$$

Vector streams compete dynamically to access memory modules and it is necessary to solve conflicts. Two types of access conflicts can be distinguished in complex memory systems: a) memory module conflict when two or more ports request access to the same memory module or when a busy memory module is requested, and b) section conflict (interconnection network conflict) when two or more ports request access to memory modules (the same memory module or different) within the same section. A priority rule determines which port will proceed and which ones must wait.

An element of a vector stream is requested in each cycle on every port except when a conflict exists in the interconnection network or in a memory module. In both cases, the request must wait.

In the simulation of the classical order (section 5), we use the arbitration used in the *CRAY X-MP* [4]: a port with an odd stride always has a higher priority than a port with an even stride, regardless of their issued sequence. For ports with same parity of strides, the priority is fixed.

Since the vector length is usually greater than the vector register length, the compiler is required to transform the code using strip-mining. Under this condition, a great proportion of memory accesses from vector streams are issued by vector instructions *load* and *store*, which are of a fixed length equal to the vector register length. Let us assume that, in order to simplify the explanation of the proposed method, the vector stream length ($VL = 2^{vl}$) is a multiple of the vector register length $MVL = 2^{mvl}$ which is assumed to be a multiple of the number of memory modules $M = 2^m$.

3 Interaction between vector streams in complex memory systems

For complex memory systems, T. Cheung and J.E. Smith characterize in [1] the linked conflict between two vector streams, and they use the term complex linked conflict (complex conflict) when three or more vector streams interfere with each other in a less precise way.

A linked conflict occurs if the memory module to be referenced by a higher priority port is already busy with a lower priority port. The higher priority port will then wait until the lower priority port has finished using the memory module. Then, when the higher priority port gains access to the memory module, it will require the same section the lower priority port needs to use (section conflict). This blocks the lower priority port for one clock period; and so, the higher priority port will be blocked again later by the lower priority port due to a memory module conflict.

Figure 2 shows the linked conflict situation reached for vector streams $A = (1,1)$ and $B=(0,1)$, when the arbitration of *CRAY X-MP* is applied in a system with $M = 8$, $n_c = 4$ and $SC = 2$. We suppose that vector stream A has a bigger priority than the vector stream B. In the figure, when a memory module or a section is occupied, its number appears at the cycle in which it begins to be occupied. The memory modules remain busy during n_c memory cycles. As an example, memory module 1 is occupied by the vector stream A at cycle 0, and it will remain busy until cycle 3. A symbol "-" points out a memory module conflict (cycle 1, vector stream B wants to access memory module 1, which is busy servicing a request of the vector stream A); a '*' points out a section conflict (cycle 4, vector stream B wants to access memory module 1 -in section 1-, but the vector stream A -that has higher priority- accesses memory module 5 -in the same section-). We can observe that a memory module conflict produces a section conflict in the next cycle, and this section conflict causes a memory module conflict n_c-1 cycles after.

cycles	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
A section	1	0	1	0	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1
B section	0	1	*	1	*	1	0	1	0	*	1	0	1	0	*	1	0	1	0	*
A module	1	2	3	4	5	6	7	0	-	1	2	3	4	-	5	6	7	0	-	1
D module	0	-	*	-	*	1	2	3	4	*	5	6	7	0	*	1	2	3	4	*

Figure 2. 8-way interleaved memory system with $n_c = 4$ and SC = 2. Linked conflicts when two vector streams access the memory: (A0, S1) = (1,1) and (B0, S2) = (0, 1).

Figure 3 shows complex conflicts generated by vector streams $A=(0,1)$, $B=(4,1)$, $C=(8,1)$ and $D=(12,1)$ in a memory system with $M = 16$, $n_c = 4$ and $SC = 4$. The priorities are $A > B > C > D$. We can observe that conflicts are periodically repeated every 5 cycles.

cycles	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
A section	0	1	2	3	0	0	1	2	3	0	0	1	2	3	0	0	1	2	3	0
B section	*	0	1	2	3	*	0	1	2	3	*	0	1	2	3	*	0	1	2	3
C section	*	*	0	1	2	3	*	0	1	2	3	*	0	1	2	3	*	0	1	2
D section	*	*	*	0	1	2	3	*	0	1	2	3	*	0	1	2	3	*	0	1
A module	0	1	2	3	-	4	5	6	7	-	8	9	10	11	-	12	13	14	15	-
B module	*	4	5	6	7	*	8	9	10	11	*	12	13	14	15	*	0	1	2	3
C module	*	*	8	9	10	11	*	12	13	14	15	*	0	1	2	3	*	4	5	6
D module	*	*	*	12	13	14	15	*	0	1	2	3	*	4	5	6	7	*	8	9

Figure 3. 16-way interleaved memory system with $n_c = 4$ and SC=4. Conflicts when four vector stream access the memory: A=(0, 1), B=(4, 1), C=(8,1) and D=(12, 1).

T. Cheung and J.E. Smith in [1], in a memory system like *CRAY X-MP* with 32 memory modules, 4 sections and a memory module latency of 4 cycles, prove that steady-state linked conflicts and complex conflicts reduce the effective bandwidth. They show that the concurrent accesses of three vector streams with 64 elements and stride one, in 34% of the cases (combinations of initial memory modules) linked conflicts appear, and in 0.7% of the cases complex conflicts are generated. Furthermore, in some specific cases performance is degraded by 20%, and in rare cases by as much as 33%. Obviously, it is important to reduce conflicts in complex memory systems.

When vector streams have different strides, even if they belong to the same family, the number of conflicts dramatically increases and performance decreases. The simulation of the concurrent access of vector streams *A=(0,1,64)*, *B=(4,3,64)* and *C=(10,7,64)* in the memory system used by T. Cheung and J.E. Smith in [1], performs 1.47 memory operations per cycle, half of the expected number of memory operations. The results are worst if the number of concurrent vector streams increases to four; for the simulation of *A=(0,1,64)*, *B=(4,3,64)*, *C=(10,7,64)* and *D=(15,9,64)* the number of memory operations obtained per cycle is 1.7, when the desired number is 4.

4 Proposal to reduce conflicts in complex memory systems

Not only memory module conflicts must be eliminated, but also section conflicts. To avoid conflicts it is necessary that, at the steady-state phase, concurrent vector streams reference different sections on a cycle, and also it is necessary that references to the same memory module are distanced, at least, n_c cycles.

Figure 4 shows the conflicts generated by vector streams *A = (0, 1)*, and *B = (12, 3)* using the classical order in a complex memory system with *M = 8, SC = 2* and $n_c = 4$. The sequences of memory references are SR_A = *(0, 1, 2, 3, 4, 5, 6, 7...)* and SR_B = *(12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42,...)*, and the sequences of memory modules are *(0, 1, 2, 3, 4, 5, 6, 7)* and *(4, 7, 2, 5, 0, 3, 6, 1)* respectively.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A section	0	1	0	1	0	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1
B section	*	0	1	0	*	*	0	1	*	1	*	1	0	*	0	*	1	0	*	0
A module	0	1	2	3	-	4	5	6	7	0	1	2	3	4	-	5	6	7	0	1
B module	*	4	7	-	*	*	2	-	*	-	*	5	-	*	0	*	3	-	*	-

Figure 4. 8-way interleaved memory system with n_c = 4 and SC = 2. Conflicts when two vector stream access the memory: (A0, S1) = (0,1) and (B0, S2) = (12, 3).

W. Oed and O. Lange present in [5] the conditions two vector streams must fulfill to produce neither memory module conflicts nor section conflicts in their concurrent access. The first condition determines when memory module conflicts are not produced; if this first condition is fulfilled, a second condition determines when there are no section conflicts.

In this paper we are going to avoid conflicts in a similar way. First, we will make a proposal for eliminating conflicts in a crossbar memory system where memory modules are directly connected to ports through a crossbar, this system only presents memory module conflicts. After that, we will complete the proposal by eliminating section conflicts in a complex memory system, using a new memory module mapping in sections.

For the first step, we can deduce from [5] and [6] that, for obtaining a memory module conflict-free access, it is sufficient (but not necessary) that vector streams access the memory modules using the same order.

The development of this access order is the objective in the next subsection as a mean to reduce memory module conflicts. We propose that the order vector streams must follow to access memory modules should be σ-independent, that is, all the vector streams of a subfamily reference the *MMS* with the same order. Under these conditions the memory subsystem sees only one stride for all the vector streams of a subfamily. As an example, for the vector stream *B = (12, 3)* the order of the memory references, we call Ordered Sequence of memory References, is OSR_B = *(12, 21, 30, 15, 24, 33, 18, 27,...)* which produces the sequence of memory module references *(4, 5, 6, 7, 0, 1, 2, 3,...)*. We can observe that the memory modules are accessed in sequential order, the memory system sees a stride 1.

Figure 5.a illustrates the concurrent access of vector streams *A=(0,1)* and *B = (12, 3)* in a crossbar memory system with *M=8* and n_c = *4* if the classic access is used. Figure 5.b shows the concurrent access of the same vector streams if they follow their Ordered Sequence of memory References. We can observe that conflicts are avoided.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A module	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
B module	4	7	-	-	-	-	2	-	-	5	-	-	0	-	-	3	-	-	6	-
A m.ref	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B m ref.	12	15	-	-	-	-	18	-	-	21	-	-	24	-	-	27	-	-	30	-

a) Concurrent Access using the Classical Order.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A module	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
B module	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
A m ref	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B m ref.	12	21	30	15	24	33	18	27	36	45	54	39	48	57	42	51	60	69	78	63

b) Concurrent Access using the OSR.

Figure 5. 8-way interleaved memory system with n_c = 4 and M=8. Conflicts when two vector stream access the memory: (A0, S1) = (0,1) and (B0, S2) = (12, 3).

Figure 6 shows conflicts generated by vector streams *A = (0, 1)*, and *B = (12, 3)* in a complex memory system with an interleaved mapping of memory modules in sections, *M = 8, SC = 2* and n_c = *4* using the Ordered Sequence of memory References to access the vector stream elements. The *OSR* are OSR_A = *(0, 1, 2, 3, 4, 5, 6, 7...)* and OSR_B = *(12, 21, 30, 15, 24, 33, 18, 27,...)*, and the ordered sequences of memory modules are *(0, 1, 2, 3, 4, 5, 6, 7)* and *(4, 5, 6, 7, 0, 1, 2, 3)* respectively. We can observe that the mapping of memory modules in sections, periodically produces a memory module conflict (cycle 4) after a section conflict (cycle 0).

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A section	0	1	0	1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1	0
B section	*	0	1	0	1	*	0	1	0	1	*	0	1	0	1	*	0	1	0	1
A module	0	1	2	3	-	4	5	6	7	-	0	1	2	3	-	4	5	6	7	-
B module	*	4	5	6	7	*	0	1	2	3	*	4	5	6	7	*	0	1	2	3

Figure 6. 8-way interleaved memory system with n_c = 4, M=8 and SC = 2. Conflicts when vector streams (A0, S1) = (0,1) and (B0, S2) = (12, 3) access a complex memory system.

The start addresses 0 and 4 for vector streams *A* and *B*, that are appropriate if the network is a crossbar, are mapped in the same section in the complex memory system and a section conflict appears at cycle 0, and due to n_c = *4*, a memory module conflict appears at cycle 4.

Thus the main idea to eliminate section conflicts is to map memory modules 0 and 4 in different sections. Then it is necessary to modify the mapping of memory modules into sections. We propose the use of a skewed mapping of memory modules in sections, instead of an interleaved mapping (section 2). Figure 7 shows the mapping of 8 memory modules in a complex memory system with 2 sections and a n_c of 4 cycles if the skewed mapping *section* = $(m_i + \lfloor m_i/4 \rfloor) mod 2$ is applied.

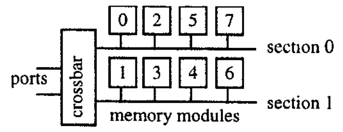


Figure 7. Complex Memory System. Skewed Memory Module Mapping.

In figure 8 we can see that there are no conflicts in the concurrent access of vector streams *A = (0, 1)*, and *B = (12, 3)* to the memory system showed in figure 7 when the *OSR* is used to access the vector stream elements.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A section	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1
B section	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0
A module	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
B module	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7

Figure 8. 8-way skewed memory system with $n_c = 4$ and $SC = 2$. Avoiding conflicts when vector streams (A0, S1) = (0,1) and (B0, S2) = (12, 3) access the memory.

Even so, when the start addresses of the vector streams are not the appropriate ones, it is necessary to use arbitration algorithms at the start-up transition phase to reach the synchronization of vector streams to obtain, independently of the initial memory address, a conflict-free phase or reduce inter-conflicts. Figure 9.a shows in a memory system with $M=8$, $n_c=4$ and $SC=2$, the spontaneous synchronization reached by vector streams $A = (0, 1)$, and $B = (1, 3)$ if the arbitration algorithm gives priority to vector stream A. Figure 9.b shows in the same memory system (with the same arbitration) that the synchronization is not reached if vector stream B starts its accesses at memory module 2. In section 4.4 we present the arbitration algorithm we consider appropriate.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A section	0	1	1	1	1	0	1	1	1	1	1	0	1	0	0	1	0	1	1	0
B section	1	0	*	*	*	1	*	*	*	*	*	1	*	1	1	0	1	0	0	1
A module	0	-	-	-	1	2	-	-	-	3	4	5	6	7	0	1	2	3	4	5
B module	1	2	*	*	*	3	*	*	*	*	*	-	*	-	4	5	6	7	0	1

a) Concurrent access of (A0, S1) = (0,1) and (B0, S2) = (1, 3)

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A section	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	1	0	0
B section	*	0	1	1	*	*	0	*	1	*	*	0	0	1	*	*	*	0	1	*
A module	0	1	-	-	-	2	3	4	-	-	5	-	6	-	-	7	0	1	-	-
B module	*	2	3	4	*	*	5	*	6	*	*	7	0	1	*	*	*	2	3	*

b) Concurrent access of (A0, S1) = (0,1) and (B0, S2) = (2, 3)

Figure 9. 8-way skewed memory system with $n_c = 4$ and $SC = 2$.

4.1 Ordered sequence of references

In the present subsection we describe how to order the memory references sequence of a vector stream $A=(A0, S)$ to access the memory modules in a σ-independent order [6, 7]; that is, all vector streams of a subfamily reference the *MMS* with the same order. The objective is to access the memory modules using the sequential order of memory modules for odd strided vector streams, and monotonic sequences of memory modules for other stride families. Under these conditions the memory subsystem sees only one stride for all the vector streams of a subfamily.

Definition 6: For a vector stream $A=(A_0, S=\sigma\times 2^s)$, we define as σ-independent order, the predetermined order used by vector streams of a stride subfamily to reference memory modules.

In this article, the chosen order to reference the memory modules of *MMS* for a stride subfamily, is the order that the stride $S = 2^s$ uses. The sequence of memory modules referenced by the vector stream $A = (A_0, S)$ is $m_{i+1} = (m_i + gcd(M, S))\ mod\ M$, where $m_0 = A_0\ mod\ M$.

Definition 7: For a vector stream $A=(A_0, S)$, we name Ordered Sequence of memory Modules (*OSM*) the sequence of memory modules $OSM = (m_0,\ldots, m_i,\ldots)$ where $m_0 = A_0 modM$, and $m_{i+1} = (m_i + gcd(M, S))\,modM$.

Definition 8: We name Ordered Sequence of References (*OSR*) to the order of memory references that fulfil the *OSM*. $OSR = \left(A_0^{OSR}, \ldots A_i^{OSR}, A_{i+1}^{OSR}, \ldots\right)$.

The vector stream is divided in sequences of P_s consecutive memory references. These sequences are issued to P_s different memory modules in *OSR* order. Definition 9 is the key to order P_s memory references in a σ-independent way.

If we lexicographically order the *MMS* of a vector stream, we can observe that the distance between adjacent memory modules is $gcd(M, S)$. Thus, there is a value that multiplied by the stride permits to obtain the address of the memory reference stored at the adjacent memory module. This value is the number of vector stream elements that are referenced by the classical order between two vector elements placed in adjacent memory modules.

Definition 9: Let C_s be the lowest natural number, greater than zero, that fulfills the relation $(C_S \times S)\ modM = (gcd(M, S))\ modM$.

Lemma 1: Let $A = (A_0, S, P_s)$ be a vector stream. The expression that generates *OSR* is: $A_i^{OSR} = A_0 + A_i^r$ where $A_0^r = 0$, $A_i^r = (A_{i-1}^r + C_S{\times}S)\ mod(P_s{\times}S)$ and $1 \le i < P_s$. $OSR = (A_0^{OSR}, \ldots A_i^{OSR}, A_{i+1}^{OSR}, \ldots)$.

Proof: In [6]. ■

Next lemma shows how to sequentially generate the memory references of the vector stream $A = (A_0, S, VL=Q{\times}P_s)$, in a σ-independent order by concatenating subsequences of P_s memory references already ordered (lemma 1). Two consecutive memory references placed in the same memory module are distanced $S{\times}P_s$ units.

Lemma 2: The sequence *OSR* for the vector stream $(A_0, S, VL = Q \times P_s)$ is calculated using the expression: $A_{i+q\times P_s}^{OSR} = A_0 + q{\times}P_s{\times}S + A_i^r$, where $0 \le q < Q$, $A_0^r = 0$, $A_i^r = (A_{i-1}^r + C_S{\times}S)\ mod(P_s{\times}S)$ and $1 \le i < P_s$.

Proof: In [6]. ■

Table 1 shows examples of *OSR* and *OSM* for different vector streams, for $M=16$ or $M=8$.

Vector stream	C_s	*OSM*	*OSR*
A=(0,3,16)	*1*	*0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15*	*0, 33, 18, 3, 36, 21, 6, 39, 24, 9, 42, 27, 12, 45, 30, 15*
B=(4,2,8)	*1*	*4, 6, 8, 10, 12, 14, 0, 2*	*4, 6, 8, 10, 12, 14, 16, 18*
C=(3,5,8)	*5*	*3, 4, 5, 6, 7, 0, 1, 2*	*3, 28, 13, 38, 23, 8, 33, 18*
D=(7,3,8)	*3*	*7, 0, 1, 2, 3, 4, 5, 6*	*7, 16, 25, 10, 19, 28, 13, 22*
E=(1,2,4)	*1*	*1, 3, 5, 7*	*1, 3, 5, 7*
F=(6,6,4)	*3*	*6, 0, 2, 4*	*6, 24, 18, 12*
G=(2,4,2)	*1*	*2, 6*	*2, 6*

Table 1. OSM and OSR sequences for some vector streams in a 16-way (A, B) and in a 8-way (C, D, E, F, G) memory systems.

Definition 10: Memory-Stride (*MS*) is the distance between two memory modules consecutively referenced in *OSM*.

$MS = gcd(M, S)\,modM$.

Corollary 1: Let $A = (A_0, S, VL=Q{\times}P_s)$ be a vector stream, where $P_s = M/gcd(M, S)$, the *OSM* sequence can be seen as a vector stream $OSM = (m_0, MS, Q{\times}P_s)$, where $m_0 = A_0 modM$, and it can be accessed using the classical order.

Proof: In [6]. ■

Figure 10 shows the algorithm and the data-path design that generates the *OSR* sequence. Value C_S can be calculated when the memory system is designed and it can be stored in a $(M/4){\times}m$ bits *ROM* memory [6]. In the algorithm, the variable *BDIR* stores the base address of every period $(A_0, A_0+P_s{\times}S, \ldots)$, and the variable A is A_i^r of lemma 2. The $mod(P_s \times S)$ operation in lemma 2 can be implemented using a conditional clause (to add $C_s{\times}S$ or $C_s{\times}S\text{-}P_s{\times}S$), as the value $A_{i-1}^r + C_s \times S$ is never greater than $2{\times}P_s{\times}S$ [6]. The selection between $C_s{\times}S$ or $C_s{\times}S\text{-}P_s{\times}S$ is made by computing the expression $A' = (A' + C_S)mod\ P_S$, that is easier to evaluate, as P_S is a power of two number; the *Ps-bit* of A' indicates which option must be selected. The former expression generates the same sequence (true, false) than $A_i^r = (A_{i-1}^r + C_S{\times}S)\ mod(P_s{\times}S)$, as $A_i^r = A_i' \times S$ [6]. Thus the conditional clause is designed using one multiplexer, one adder and a logic circuit which selects the *Ps*-bit of A'.

```
{CS = C_s; PS = P_s; CSS = CS×S; PSS = PS×S}
{CPSS= CSS- PSS}
BDIR = A_0
A = 0
A' = 0
DIR = A_0                    ;request address
do I =1 to PS-1
    if (A' + CS > PS)        ;cond. clause
        then    A = A + CPSS
        else    A = A + CSS
    A' = (A' + CS) mod PS    ;cond. clause
    DIR = BDIR + A           ;request address
enddo
do K =1 to ⌊VL/PS⌋-1
    A = A + CPSS
    A' = 0
    DIR = BDIR + PSS         ;request address
    || BDIR = BDIR + PSS     ;in parallel
    do I =1 to PS-1
        if (A' + CS > PS)    ;cond. clause
            then    A = A + CPSS
            else    A = A + CSS
        A' = (A' + CS) mod PS;cond. clause
        DIR = BDIR + A       ;request address
    enddo
enddo
```

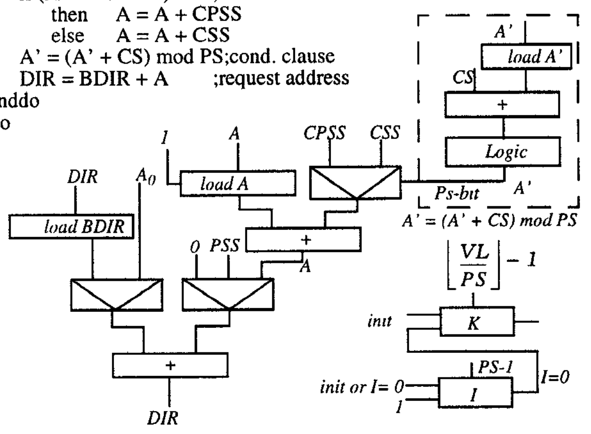


Figure 10. Algorithm and hardware necessary to generate the sequence OSR.

4.2 Memory module mapping model

The skewed memory module mapping model presented in this section is valid for the case $n_c = SC \leq NM$, where $NM=2^{nm}$ is the number of memory modules per section.

The memory subsystem is organized in a skewed module mapping model $section = (m_i + \lfloor m_i/SC \rfloor) modSC$, where m_i is a memory module. The organization for a complex memory system with $M=16$ and $SC=4$ is shown in figure 11.

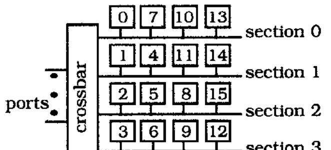


Figure 11. Complex Memory System. Skewed Memory Module Mapping.

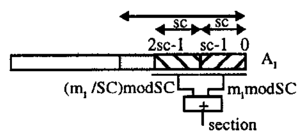


Figure 12. Skewed module generation.

In a memory system with $SC = 2^{sc}$ sections, it is necessary to have an sc-bits adder to compute the section where memory modules are mapped. The memory address bits that must be added are $[sc\text{-}1,..., 0]$ and $[2\times sc\text{-}1,..., sc]$, and the section number is coded in the sc-less significant bits of the result (figure 12).

The skewed mapping model which maps modules into sections has a period of $P^{SC} = SC^2$.

If the parameters that define the complex memory system fulfil: $n_c \leq SC \leq NM$ or $SC \leq n_c \leq NM$, the memory module mapping model differs slightly from the one we have presented in this section, and can be easily developed [8].

4.3 Characterization of the access to the memory system

Using *OSR* to reference the vector stream elements and the skewed memory module mapping we propose, the referenced sections are accessed in a σ-independent order; that is, all the vector streams of a subfamily reference sections with the same order. Thus, there are no section conflicts between references of vector streams belonging to the same subfamily.

In a sequence *OSR* of P_s memory references we can differentiate subsequences.

Definition 11: We name subsequence of *OSR* (*SOSR*) to every set of contiguous memory references for which $\lfloor m_i/SC \rfloor modSC$, ($m_i = A_i^{OSR} modM$), is the same.

Definition 12: We name subsequence of *OSM* (*SOSM*) to every set of memory modules associated to *SOSR*.

Figure 13 presents some examples of *SOSR* and *SOSM* (boxes) for a complex memory system with *M=16* and *SC=4*. As an example, for the vector stream *B=(9, 2, 8)* the *SOSR*s are *(9, 11) (13, 15)*, *(1,3)* and *(5,7)*.

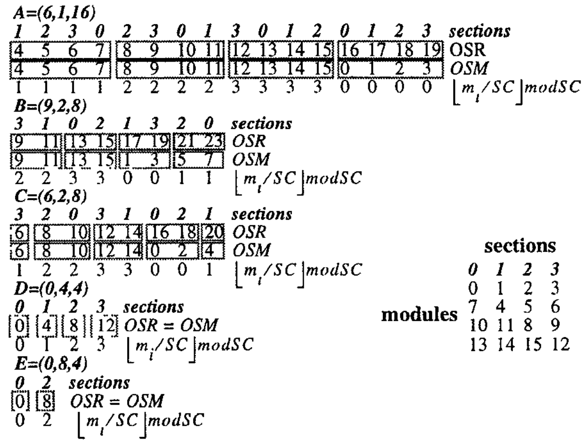


Figure 13. Examples of OSR and OSM and section order.

We can distinguish *SC* different *SOSR* if the vector stream is self-conflict-free ($s \leq m - c$, as vector streams *A, B, C* and *D* of figure 13) and P_s different *SOSR* if the vector stream is no-self-conflict-free ($s > m - c$, as vector stream *E* in figure 13). The number of elements in every *SOSR* depends on the stride subfamily of the vector stream and is $\lceil min(P_s, SC)/MS \rceil$ [8].

Lemma 3: Let *(M, SC)* be a complex memory system with an skewed memory module mapping in sections, and $A= (A_0, S, P_s)$ a vector stream. Sections in one *SOSR* are referenced using the order imposed by: $sc_{h+1} = (sc_h + MS) modSC$, $0 \leq h < \lceil min(P_s, SC)/MS \rceil$, where $sc_0 = ((\lfloor m_i/SC \rfloor) + A_0 \, mod \, 2^s) mod \, SC$ and m_i belongs to *SOSR*.

Proof: in [8]. ■

In a skewed memory module mapping, for an $OSM = (A_0 modM, MS)$ the period to reference the sections is:

$P_{MS}^{SC} = SC^2/gcd(SC^2, MS)$, and $P_{MS}^{SC} \leq P_s$.

As an example, in figure 13, for the vector stream *A=(6, 1, 16)*, sections where memory modules are mapped are (1, 2, 3, 0), (2, 3, 0, 1), (3, 0, 1, 2) and (0, 1, 2, 3). We can observe that memory module references distanced n_c cycles are placed in different sections; as an example, for vector stream *B = (9, 2, 8)*, accessing memory module 11 implies to access section 1 and the memory module referenced $n_c=4$ cycles after (memory module 3) implies to access section 3.

It is important to mention that not all the sections are always referenced in every *SOSR*; as an example, for the vector stream *B=(9, 2, 8)* the sections where *SOSR* are mapped are *(3, 1), (0, 2), (1, 3)* and *(2, 0)* respectively.

In [8] we proved the next corollary that identifies cases of concurrent accesses for which the *OSR* does not eliminates memory module conflicts.

Corollary 2: In the case of a crossbar interconnection network, the use of the *OSR* in the concurrent access of *N* vector streams belonging to different families and with a non-void intersection of *MMS*, produces memory module conflicts.

As there are memory module conflicts (we do not consider section conflicts) the concurrent access is no-conflict-free. Due to that, the use of *OSR*, the skewed memory module mapping in sections and the arbitration (we are going to introduce in the next section) only reduce the number of lost cycles due to the presence of conflicts.

4.4 Arbitration algorithm

An arbitration algorithm is needed in order to synchronize vector streams to reach a conflict-free steady-state phase or to dramatically reduce inter-conflicts, for any combination of initial memory modules.

In figure 13, we can observe that each *SOSR* references the sections following a predetermined order which is different for every *SOSR*; thus, if we overlap different *SOSR*, different sections are simultaneously referenced. The main idea is that, in every cycle concurrent vector streams reference memory modules of different *SOSR*, and these different *SOSRs* must be aligned. Thus, the *SOSR* changes between vector streams are the most important concern for the arbitration algorithm. When all *SOSR* changes have been detected for all vector streams, *SOSRs* are assigned using a fixed priority. In the steady state the *SOSRs* changes are detected, for every vector stream, every $\lceil min(P_s, SC)/MS \rceil$ memory references.

Figure 14 presents, in a memory system that has *M=16, SC = 4* and $n_c = 4$, the proposed arbitration in the concurrent access of four vector streams, *A = (0, 1), B = (0, 3), C = (0, 5)* and *D = (0, 7)*, for which $P_s = M$. In the figure, a vertical line points out the moment at which all vector streams change the *SOSR*. A character '∅' depicts the cycles in which the arbitration algorithm does not assign a *SOSR* to the vector stream. In this case the synchronization is reached at cycle 12.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A sect	0	1	2	3	1	2	3	0	2	3	0	1	3	0	1	2	0	1	2	3	1
B sect	∅	∅	∅	∅	0	1	2	3	1	2	3	0	2	3	0	1	3	0	1	2	0
C sect	∅	∅	∅	∅	∅	∅	∅	∅	0	1	2	3	1	2	3	0	2	3	0	1	3
D sect	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	0	1	2	3	1	2	3	0	2
A mod	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4
B mod	∅	∅	∅	∅	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
C mod	∅	∅	∅	∅	∅	∅	∅	∅	0	1	2	3	4	5	6	7	8	9	10	11	12
D mod	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	0	1	2	3	4	5	6	7	8
A Sn	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	0	0	0	0	1
B Sn	∅	∅	∅	∅	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	0
C Sn	∅	∅	∅	∅	∅	∅	∅	∅	0	0	0	0	1	1	1	1	2	2	2	2	3
D Sn	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	0	0	0	0	1	1	1	1	2

Figure 14. 16-way skewed memory system with $n_c = 4$ and $SC = 4$. Synchronization between A = (0, 1), B = (0, 3), C = (0, 5) and D = (0, 7) using SOSR and the arbitration algorithm. Sn =SOSR number = $\lfloor m_i/SC \rfloor$ modSC.

The *SOSR* change can be detected by computing the expression $SOSR = \lfloor m_i/SC \rfloor modSC$ ($m_i = A^{OSR}_i modM$) for two consecutive memory module requests of a vector stream (the current and the previous requests). The *SOSR* change state for a vector stream is maintained until the first memory module reference of the new *SOSR* is served.

Figure 15 shows the hardware needed for the *SOSR* change detection. The signal "active" points out if there is a vector stream.

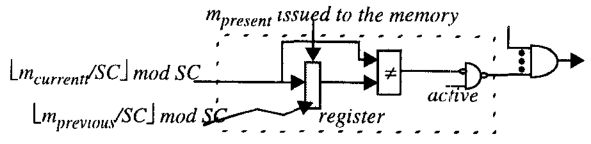

$m_{present}$ issued to the memory
$\lfloor m_{current}/SC \rfloor$ mod SC
$\lfloor m_{previous}/SC \rfloor$ mod SC
≠
active
register

Figure 15. Change of SOSR detection for one (box) and all the vector streams.

Figure 16 shows differences between the concurrent classical access of four vector streams, *A =(0, 1), B=(12, 2), C=(8, 6)* and *D=(4, 14)*, for which the $P_s \leq M$, to an interleaved memory system (16.a) and the concurrent access of the same four vector streams using the *OSR* in the skewed complex memory system using the arbitration we propose (16.b). A character '*' points out a section conflict, and a character '-' a memory module conflict; a character '.' points out the cycles the vector stream waits for an arbitration. In 16.a, we can see that, in this case, the number of operations per cycle is 1.47, while, in the case of using the *OSR* in the skewed mapping proposed, figure 16.b, the number of operations per cycle is 2.5. In this last case, we obtain the maximum reachable performance, even if there are still some memory module and section conflicts (corollary 2), that, finally help vector streams to reach a synchronized situation.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A sect	0	1	2	3	0	0	0	0	1	2	3	0	1	2	3	0	1	2	3	0	0
B sect	*	0	*	2	*	*	*	*	0	*	2	2	2	0	0	*	0	*	2	*	*
C sect	*	*	0	*	2	2	2	2	*	0	0	*	0	*	2	2	2	*	*	2	*
D sect	*	*	*	0	*	*	*	*	2	*	*	*	*	*	*	*	*	0	*	*	2
A mod	0	1	2	3	-	-	-	4	5	6	7	8	9	10	11	12	13	14	15	-	-
B mod	*	12	*	14	*	*	*	*	0	*	-	-	2	-	-	*	4	*	6	*	*
C mod	*	*	8	*	-	-	-	14	*	-	-	*	4	*	-	-	-	*	*	10	*
D mod	*	*	*	4	*	*	*	*	2	*	*	*	*	*	*	*	*	0	*	*	-

a) 16-way interleaved memory system with $n_c = 4$ and $SC = 4$.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A sect	0	1	2	3	1	2	3	0	2	3	0	1	3	0	1	2	0	1	2	3	1
B sect	3	*	1	.	0	*	2		1	*	3		2	*	0	.	3	*	1		0
C sect	2	0		.	3	1	1		0	2	2		1	3	3	.	2	0	0	.	3
D sect	1	3	.	.	2	0	.	.	3	1	1	.	0	2	2		1	3	3		2
A mod	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4
B mod	12	*	14	.	0	*	2	.	4	*	6	.	8	*	10	.	12	*	14	.	0
C mod	8	10		.	12	-	14	.	0	-	2	.	4	-	6		8	-	10	.	12
D mod	4	6			8	10			12	-	14	.	0	-	2	.	4	-	6	.	8

b) 16-way skewed memory system with $n_c = 4$ and $SC = 4$.

Figure 16. Concurrent Access of Vector streams A=(0, 1),B=(12, 2), C=(8, 6) and D = (4, 14).

For a vector stream the first and last *SOSRs* can be incomplete sequences; that is, these *SOSRs* can have less than $\lceil min(P_s, SC)/MS \rceil$ elements (as vector stream *C* in figure 13). We name complete *SOSR* to the *SOSR* sequence that has $\lceil min(P_s, SC)/MS \rceil$ elements. In the case that some of the concurrently accessing vector streams have less than $\lceil min(P_s, SC)/MS \rceil$ memory modules in the first *SOSR*, the described arbitration is helped by the appearance of section and memory module conflicts. Figure 17 shows such an example where the first *SOSR* of three vector streams is not complete. Vector streams *A = (2, 1), B = (3, 3), C = (1, 5)* and *D = (0, 7)* access a complex memory system with *M = 16, SC = 4* and $n_C = 4$. A section conflict is depicted by a character '*', and a memory module conflict by a character '-'; a character '.' points out the cycles the vector stream waits for an arbitration and a character '∅' depicts the cycles in which the arbitration algorithm does not assign a *SOSR* to the vector stream. In the example, we can see that at cycle 5 the memory request of *B* to the memory module 3 is issued in the correct position, after 3 cycles of unsuccessful delivery. In this example the synchronization is reached in cycle 10.

cycles	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A sect	2	3	1	2	3	0	2	3	0	1	3	0	1	2	0	1	2	3	1	2	3
B sect	∅	∅	*	3	*	3	1	2	3	0	2	3	0	1	3	0	1	2	2	3	0
C sect	∅	∅	∅	∅	∅	∅	*	1	2	3	1	2	3	0	2	3	0	1	3	0	1
D sect	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	0	1	2	3	1	2	3	0	2	3	0
A mod	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6
B mod	∅	∅	*	-	*	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2
C mod	∅	∅	∅	∅	∅	∅	*	1	2	3	4	5	6	7	8	9	10	11	12	13	14
D mod	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	0	1	2	3	4	5	6	7	8	9	10
A Sn	0	0	1	1	1	1	2	2	2	2	3	3	3	3	0	0	0	0	1	1	1
B Sn	∅	∅	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	0	0	0
C Sn	∅	∅	∅	∅	∅	∅	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3
D Sn	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	0	0	0	0	1	1	1	1	2	2	2

Figure 17. 16-way skewed memory system with $n_c = 4$ and $SC = 4$. Synchronization between $A = (2, 1)$, $B = (3, 3)$, $C = (1, 5)$ and $D = (0, 7)$ using SOSR and the arbitration algorithm. Sn =SOSR number = $\lfloor m_i/SC \rfloor \mathrm{mod} SC$.

5 New method performance

In this section we are going to present the advantages of the method proposed in this paper (*OSR*+skew+arbitration, we call it *OSRS*).

First, we can say that the benefits of the use of the *OSR* in front of the Classical Order in a complex memory system, either with interleaved or skewed mapping of memory modules into sections, are that the memory module conflicts are dramatically reduced because vector streams access the memory modules following the same order.

Table 2 shows the comparison between *OSRS* and the *OSR* order applied in a memory system with an interleaved memory module mapping (we call it *OSRI*), and figures 18 show the comparison of the *OSRS* with the Classical Order in a complex memory system with an interleaved mapping of memory modules into sections (we call it *COI*). All the comparisons have been made by computing the increment in performance (*IP*) which is implied by the *OSRS* in a memory system with $M=16$, $SC=4$ and $n_c=4$. The *IP* is computed using the equation: $IP = ((c1 - c2)/c2) \times 100$, where $c1$ and $c2$ are the number of cycles the *OSRS* and the other method (*OSRI* or *COI*) respectively need to complete the memory accesses of the vector streams.

The vector streams for which the simulations have been made are $A = (0, S_A, 128)$, $B_e = (e, S_B, 128)$, $C_f = (f, S_C, 128)$ and $D_g = (g, S_D, 128)$; the relative distances between the first memory module referenced by vector streams B_e, C_f and D_g and the first memory module referenced by A, fulfil $0 \le e, f, g \le M-1$, and S_A, S_B, S_C and S_D belong to the interval $[1, M-1]$.

The number of cycles needed for every quartet (A, B_e, C_f, D_g) to complete their accesses is calculated. The relative locations are assumed to take any value with the same probability (avoiding the combinations of relative locations that cause vector streams to have disjoint memory module sets -*MMS*-). Thus, we computed the *IP* average for every set $\{(A, B_d, C_f, D_g);\ S_A, S_B, S_C$ and S_D are constant, and $0 \le e, f, g \le M-1\}$, we name it $IP(S_A, S_B, S_C, S_D)$.

Table 2 shows the comparison between *OSRI* and *OSRS* for different combinations of four strides. Both methods use the order *OSR* to access vector stream elements; this order reduces the memory module conflicts, but the results show the benefits of the new skewed mapping of memory modules into sections.

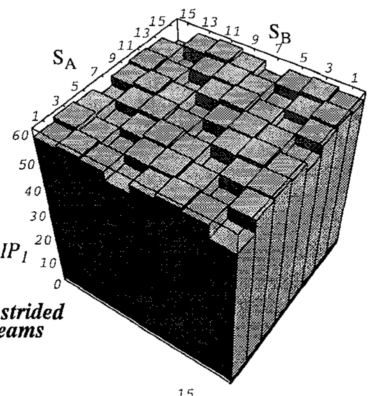
The maximum *IP* is obtained for 4 odd strided vector streams; the 0% *IP* value is obtained in the case that the 4 vector streams access the memory with stride 8, vector streams only reference 2 memory modules.

In the case that $s > sc$ ($SC=n_c=2^{sc}$, as an example: stride=$2^s=8$ and $n_c=4$), memory module conflicts hide the possible section conflicts, and, either *OSRS* or *OSRI* have a bad behavior.

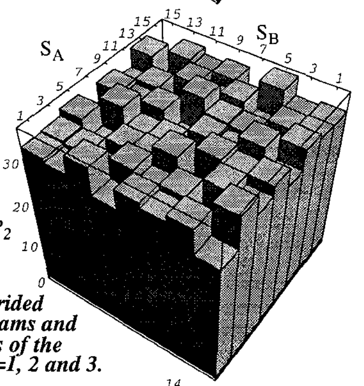
Combinations of Strides		*OSRS vs OSRI*	
Odd Family	*Families s>0*	*Max. I.P.*	*Min. I.P.*
4	*0*	*43%*	*43%*
3	*1*	*29%*	*13%*
2	*2*	*37%*	*7%*
1	*3*	*36%*	*4%*
0	*4*	*37%*	*0%*

Table 2. System $M = 16$, $SC = 4$ and $n_c = 4$. Increment in Performance for 4 vector streams.

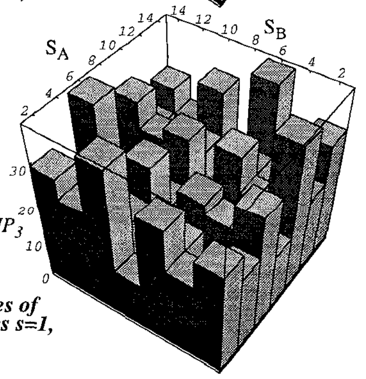
We will now present the comparison of *OSRS* with *COI* in figures 18. Figure 18.a shows the results for the $IP_1(S_AS_C, S_BS_D)$, that is the average of the $IP(S_A, S_B, S_C, S_D)$ for which S_A and S_B are constant and odd and, S_C and S_D are variable and odd. Next, figure 18.b shows the $IP_2(S_AS_C, S_BS_D)$, that is the average of the $IP(S_A, S_B, S_C, S_D)$ for which S_A and S_B are constant and odd and, S_C and S_D are variable and belong to the families $s=1$, 2, and 3. Figure 18.c shows the $IP_3(S_AS_C, S_BS_D)$, that is the average of the $IP(S_A, S_B, S_C, S_D)$ for which S_A and S_B are constant, S_C and S_D are variable and all of them belong to the families $s=1$, 2, and 3.



18.a. Four odd strided vector streams



18.b. Two odd strided vector streams and two strides of the families s=1, 2 and 3.



18.c. Four strides of the families s=1, 2 and 3.

Figure 18. System $M = 16$, $SC = 4$ and $n_c = 4$. IP Average for the concurrent access of combinations of four vector streams from the families s=0, 1, 2 and 3.

The maximum *IP* is 65.6% which is obtained for some combinations of odd strides. For odd strides, the *IP* averages range from 52.6% to 58.7% (figure 18.a); these are the best results as all kind of conflicts (memory module and section) are avoided. For combinations of strides of families $s=0$, *1*, *2*, and *3*, values are into the interval [27.6%, 33.9%] (figure 18.b). In the case that all the strides belong to families with $s > 0$, values are between 6.3% and 38.2% (figure 18.c). Figures 18.b and 18.c present the lowest benefits as they include cases of vector streams combinations for which the request rate to every memory module is greater than the memory module service rate, and their concurrent access generate memory module conflicts as well as section conflicts.

From the results shown in figures 18 and table 2, we can conclude that the use of *OSRS* implies a large increment in performance (*IP*).

J. Fu and J.H. Patel in [9] show that between 7% and 54% of the vector streams in four programs of the Perfect Club benchmark set [10] (*ADM, ARC2D, BDNA* and *DYESM*) access the memory with stride 1; for these cases, the presented results show an *IP* average of almost 60%, for the *OSRS* respect the *COI*.

Vectors and matrix are the most common data structures in vector processors. In Fortran, the most frequent accesses to matrix are made to columns, rows and diagonals, that correspond to the strides 1, n and $n+1$ respectively, where n is the column length, which is dependent on the problem size that varies widely. Present compilation technology allows us to detect if n is even and to increment the matrix size in one row, this determines that the number of referenced memory modules is *M*. Thus, in row-major and column-major accesses the use of *OSR* performs equally well, and there are no conflicts. When n is even and there is no possibility of increasing the number of rows, *OSRS* reduces the number of conflicts.

6 Conclusions

This article has focused on the inter-conflicts generated by multiple vector streams concurrently accessing a complex memory system. The access patterns correspond to constant strides.

As a mean of reducing inter-conflicts, we define an order to access vector stream elements to reference the memory modules that is σ-independent. The stride and the first memory module referenced determine the set of memory modules and the corresponding sections to be accessed. Also, we have designed a mapping model to place memory modules into sections. Furthermore, we propose an arbitration algorithm (which is independent of the first memory module referenced by vector streams) to help vector streams to reach a conflict-free access (synchronization) or to dramatically reduce inter-conflicts.

The comparison between *OSRS* and the classical method shows a great increment in performance for our method. This increment is bigger for memory systems which allow a great number of concurrent vector streams, as the probability of conflict in these systems increases. The proposed access order also takes advantage of a large n_C. The larger the n_c ($n_c \approx 4$) the greater the probability of conflict.

Acknowledgments

This work has been supported by the Ministry of Education of Spain under contract TIC-92-0880 and the CEPBA (European Center for Parallelism of Barcelona).

References

[1] T. Cheung and J.E. Smith
A Simulation Study of the CRAY X-MP Memory System
IEEE Transactions on Computers, Vol. C-35, no 7, october 1980, pp. 613-622.

[2] U. Detert and G. Hofemann
CRAY X-MP and Y-MP memory performance
Parallel Computing, North-Holland, n0 17, 1991, pp. 579-590.

[3] D.T. Harper III and D.A. Linebarger
Conflict-free Vector Access Using a Dynamic Storage Scheme
IEEE Transactions on Computers, Vol. C-40, no 3, march 1991, pp. 276-283.

[4] S. Edirisooriya and G. Edirisooriya
Enhancing Vector Access Performance in CRAY X-MP Memory System
Compcon-93, pp. 569-576.

[5] W. Oed and O. Lange
On the Effective Bandwidth of Interleaved Memories in Vector Processor Systems
IEEE Transactions on Computers, Vol. C-34, no 10, october 1985, pp. 949-957.

[6] A. M. del Corral and J. M. Llabería
Out-of-Order Access to Vector Elements in order to Reduce Conflicts n Vector Processors
Sixth IEEE Symposium on Parallel and Distributed Processing. October 1994, pp. 126-134.

[7] A. M. del Corral and J. M. Llabería
Hardware Support to Reduce Conflicts between Vector Streams
2nd Int. Workshop on Massive Parallelism: Hw, Sw and Applications. October 1994, pp. 90-104.

[8] A. M. del Corral and J. M. Llabería
Reducing Inter-Vector-Conflicts in Complex Memory Systems
CEPBA Report RR-95/29. and DAC-UPC Report. RR-95/43. November, 1995.

[9] J.W. FU and J.H.Patel
Memory Reference Behavior of Compiler Optimized Programs on High Speed Architectures
Int. Conference on Parallel Processing, 1993, Vol II, pp. 87-94.

[10] M. Berry et al.
Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers
Int. Journal for Supercomputer Applications, Fall 1989.