INCREASING THE NUMBER OF STRIDES FOR CONFLICT-FREE VECTOR ACCESS

Mateo Valero, Tomás Lang, José M. Llabería, Montse Peiron, Eduard Ayguadé and Juan J. Navarro
Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya
c/ Gran Capità s/n, Mòdul D-4, 08034 - Barcelona, SPAIN.
Phone: + 34 - 3 - 401 69 79, E_mail: mateo@ac.upc.es

ABSTRACT

Address transformation schemes, such as skewing and linear transformations, have been proposed to achieve conflict-free vector access for some strides in vector processors with multi-module memories. In this paper, we extend these schemes to achieve this conflict-free access for a larger number of strides. The basic idea is to perform an out-of-order access to vectors of fixed length, equal to that of the vector registers of the processor. Both matched and unmatched memories are considered; we show that the number of strides is even larger for the latter case. The hardware for address calculations and access control is described and shown to be of similar complexity as that required for access in order.

KEYWORDS

Vector processors, Multi-module memories, Vectors with constant stride, Conflict-free access.

1. Introduction

To have a sufficient memory bandwidth, the memory of fast processors is organized as several modules that can be accessed simultaneously. To achieve a memory throughput of one access per processor cycle, the number of memory modules should be at least equal to the ratio between the memory cycle and the processor cycle (matched memory system). However, to obtain this throughput, the request sequence has to be such that there are no conflicts in the accesses. This is achieved, for example, with conventional memory interleaving and for vectors with odd strides, but not for other strides or more unstructured patterns. This has motivated the proposal of other addressing schemes, the use of buffers, as well as the increase of the number of memory modules (unmatched memory system).

In this paper we consider ways of obtaining minimum latency for address streams which correspond to a single vector of constant stride, fixed length and any initial address. This addressing pattern is typical of vector processors (and scalar processors accessing vectors), but does appear also in general scalar processing when the processor has decoupled memory access and execute. In both cases, to overlap memory access and execution, the processor is decomposed into two independent modules, as shown in Figure 1. The memory-access module performs loads and stores to/from a "register file" and the execute unit obtains the operands from that file.

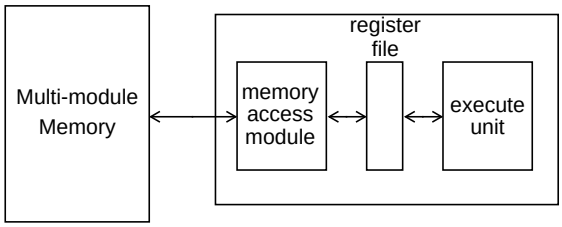


Figure 1: Decoupled memory access and execute architecture

Vectors with constant stride appear naturally for vector processors, because these patterns are directly supported by the instruction set. In the case of scalar processors, these regular patterns are convenient to achieve high effective memory bandwidth. These patterns could also be useful in a software-controlled cache in cases in which the spatial locality of the data corresponds to these blocks of "equally spaced" items.

We consider the case in which the processor accesses vectors of length equal to the number of elements of one of its vector registers, because this is the way the LOAD and STORE instructions operate. Since the size of the vectors is usually much larger than the size of vector registers, the above mentioned mode of operation requires strip-mining by the compiler so that a very high fraction of the accesses are of vectors of length equal to that of the registers. In Section 5, we discuss the access of shorter vectors.

Moreover, we propose that the elements of the vector be requested out of order and that the whole vector be stored in the vector register before its use by the processor. This prevents the chaining of LOAD/STOREs with other operations; however, this is reasonable because the complex timing of memory accesses make this chaining difficult anyhow. Even

The final publication is available at ACM via http://dx.doi.org/10.1145/146628.140400

though this decoupled operation would be the default, we discuss in Section 5 that the proposed out-of-order access can support chained access/execute in cases where ordered access would make this mode of operation impractical.

The two main address transformation schemes proposed in the literature to achieve conflict-free access to vectors with strides that produce conflicts with conventional interleaving are skewing and linear transformations. These schemes were initially proposed for array processors [1, 2, 3, 4] and later for vector processors [5, 6, 7, 8], multiprocessors [9], and VLIW processors [10]. For vectors and a matched-memory system, they can provide conflict-free access to one family of strides, where the family defined by x is the set of strides $\sigma \cdot 2^x$ with σ odd [11]. Moreover, for the case in which different vectors are accessed with different strides, dynamic schemes based on skewing [11] and on linear transformations [6] were proposed. Linear transformations have the advantage over skewing that usually the module number is simpler to compute.

A larger number of families of conflict-free strides can be achieved by increasing the number of memory modules (unmatched memory system). If $M=2^m$ is the number of memory modules and $T=2^t$ is the ratio between the memory cycle and the processor cycle, then at most (m-t+1) families are conflict free, assuming that the elements are requested in order [6].

Although out of the scope of this paper, it is worthwhile to mention that techniques have also been proposed to improve efficiency for the cases in which conflict-free access is not achieved. For the skewing and linear schemes mentioned above, peak memory throughput can be obtained for $x' < x$ for long vectors by the use of buffers [5]. Moreover, schemes based on linear transformations have been proposed to distribute randomly the modules corresponding to consecutive addresses, so that the various strides do not produce clustering to memory modules [8, 10, 12]. Recently a proposal has been made [13] for an analytical model that can be used to make comparisons among these linear transformations. For both schemes, most of the evaluations performed consider long vectors, so that the initial transient is not significant and the throughput is determined for the steady state. This throughput is evaluated as a function of several parameters, such as structure of the transformation, number of buffers, and number of memory modules. Although in [8, 10, 11] some measurements are given for short vectors, the effect of length is not discussed nor is the transformation determined with a vector length in mind.

To introduce the out-of-order accessing we use a matched memory system and a linear transformation of the addresses, although the same results can be obtained with interleaving (using an internal field of the address as module number) or with skewing. We show that this mode of accessing vectors of fixed size results in conflict-free accesses for a larger number of families of strides than ordered access. The case of unmatched memory system is also studied. The hardware required for address calculations is presented and the efficiency of the scheme is evaluated.

The results in this paper should be extended to the cases in which several vectors are accessed simultaneously by a single processor or by several processors in a multiprocessor system.

2. Model and Condition for Conflict-Free Access

We now describe the model of the memory subsystem, present some definitions and give the condition for conflict-free access.

Figure 2 shows the general structure of the system which has been used by previous authors. The memory is composed of $M=2^m$ modules and the module latency is of $T=2^t$ processor cycles. Each memory module has q input and q' output buffers. The processor requests one element per (processor) cycle unless it has to wait because the associated input memory buffer is full. The latency of the vector access is defined as the number of processor cycles from the time the processor sends the first address until the last element is received. We assume that the interconnection network is a single bus with a delay of one cycle. Therefore, the latency of a conflict-free access to a vector of length L is (T+L+1) cycles.

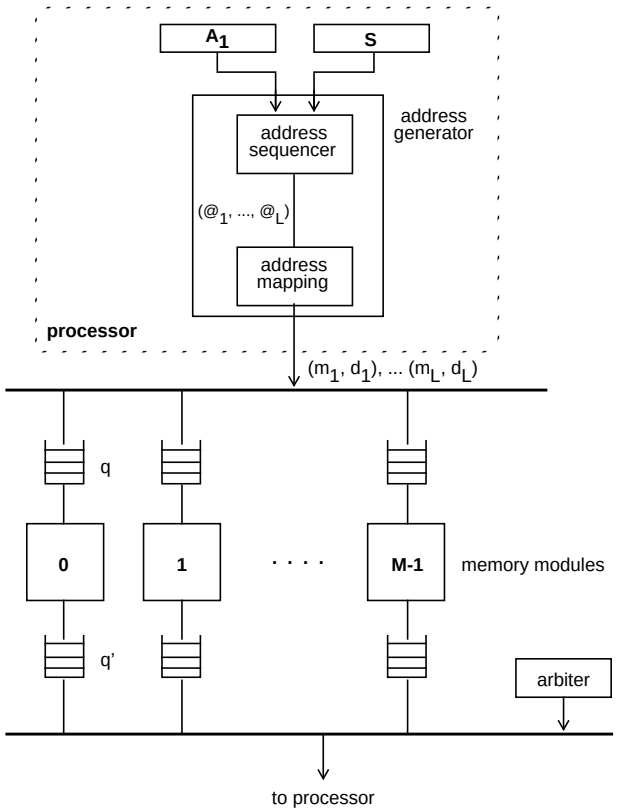


Figure 2: General structure of the system. $(@_1, ..., @_L)$ refers to any request ordering.

We consider the vector length L equal to 2^λ, with $\lambda \geq m$. The first element of the vector has address A_1 and consecutive elements are separated by a constant value S (the stride)

so that the i-th element has address $A_1+S\cdot(i-1)$. Note that the vector can have any initial address. As done in [11], we classify the strides into families defined by x so that all strides $\sigma\cdot 2^x$ with σ odd belong to the same family.

Since the memory is organized in several modules, an address mapping is required which transforms the address A (one-dimensional) with binary representation a_{n-1}, a_{n-2}, ..., a_1, a_0 (in short also denoted $a_{n-1..0}$) into the two-dimensional space (module, displacement). Since conflicts depend only on the module number part, we only consider that component of the mapping. That is, the module number b with binary representation $b_{m-1..0}$ is given by b = F(A) where F is the memory-module component of the address mapping.

We now define the spatial and temporal distributions of the elements of a vector, since these determine the latency of the access.

Definition: The SPATIAL DISTRIBUTION of a vector in the multi-module memory is the M-tuple SD, where SD(i) is the number of vector elements in module i.

The spatial distribution is LATENCY-MATCHED (T-MATCHED) if $SD(i) \le L/T$ for all i (this implies that at least T modules have SD(i) > 0). If the spatial distribution of a vector is T-matched we say that the VECTOR IS T-MATCHED.

Definition: The TEMPORAL DISTRIBUTION of a vector is the sequence of memory-module numbers (m_1, ..., m_L) where m_i is the module corresponding to the i-th processor request. Note that the elements can be requested in any order.

Other authors use similar terms related with the spatial distribution, such as short-term and long-term equidistributed sequences [12], and with the temporal distribution, such as return numbers [14] and variability [13].

Definition: A temporal distribution is CONFLICT FREE when every element can be accessed as soon as it is requested (the corresponding memory module is not busy with a previous request). This is equivalent to stating that a temporal distribution is conflict free if any subset of T consecutively requested elements are located in T different memory modules. This is the condition we will require.

Moreover, from the last definition it follows directly that a necessary condition for a conflict-free temporal distribution is that the vector be T-matched. Because of this, to determine the conditions for a conflict-free temporal distribution, we first determine conditions for a T-matched vector and then consider access orders so that any T consecutive accesses are to different memory modules.

Since the spatial distribution is independent from the temporal distribution, to determine conditions for a T-matched vector we use the canonical temporal distribution, defined below, even though it might not be conflict free.

Definition: The CANONICAL TEMPORAL DISTRIBUTION of a vector is the temporal distribution when the elements are requested in order.

For linear mappings (and for skewing) the canonical temporal distribution is periodic. Consequently, we can define the canonical temporal distribution in one period and state Lemma 1, below.

Definition: The period P_x of an address mapping for a vector with stride $\sigma\cdot 2^x$ is the period of its canonical temporal distribution.

Definition: We call CTP_x the canonical temporal distribution in one period for a vector with stride $\sigma\cdot 2^x$.

Definition: CTP_x is T-matched if the spatial distribution in the period is T-matched.

Lemma 1: If CTP_x is T-matched and $L = k\cdot P_x$ with $k > 0$ then the vector is T-matched.

Proof: This is evident since if each period is T-matched and the vector length is a multiple of the period, the vector has to be T-matched. ❑

In summary, we consider vectors with $L = k\cdot P_x$, determine the conditions for having a T-matched CTP_x and then find a temporal distribution that has the property that any T consecutive requests go to different modules.

3. Matched Memory

We now discuss the matched-memory case (M=T) and generalize to the unmatched case in the next section.

For the matched-memory case, we choose F as the linear transformation

$$b_i = a_i \oplus a_{s+i} \qquad s \ge t,\ 0 \le i \le t-1 \qquad (1)$$

For the rest of this section we assume this mapping. It has the property that when the elements of the vector are requested in order the access is conflict free for vectors with strides of the family with x = s, of any length and with any initial address [6].

Figure 3 illustrates a portion of the mapping for m = t = 3 and s = 3.

The period for a stride $\sigma\cdot 2^x$ is $P_x = \lceil 2^{s+t-x} \rceil$ [6].

Lemma 2: Let $x \le s$ and P_x be the corresponding period. Consider the grouping of these P_x elements into 2^{s-x} subsequences consisting of 2^t elements each. The i-th subsequence ($1 \le i \le 2^{s-x}$) contains the elements ($i + k_1\cdot 2^{s-x}$) with $0 \le k_1 \le 2^t-1$. For any of these subsequences, all its elements are located in different memory modules.

Proof: Let A_i, with binary representation $a_{n-1..0}$, be the address of the i-th element. For the mapping defined in (1), this element is located in module m_i such that

$$m_i = (a_{s+t-1} \oplus a_{t-1}, ..., a_{s+1} \oplus a_1, a_s \oplus a_0) = a_{s+t-1..s} \oplus a_{t-1..0}$$

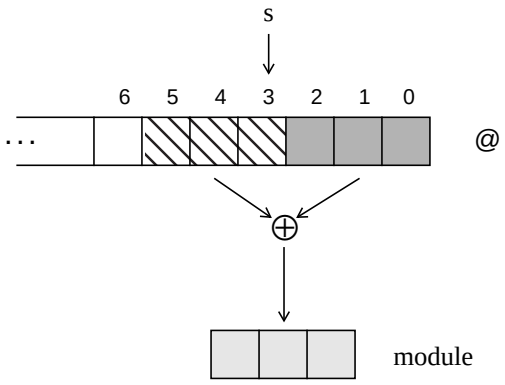
Moreover, the element $i + k_1\cdot 2^{s-x}$ has address

$$A_i + k_1\cdot 2^{s-x}\cdot\sigma\cdot 2^x = A_i + k_1\cdot\sigma\cdot 2^s$$

and is located in module

$$((a_{s+t-1..s} + (k_1\cdot\sigma) \bmod 2^t) \bmod 2^t) \oplus a_{t-1..0}$$

Since σ is odd, the values $(k_1 \cdot \sigma) \bmod 2^t$ for $0 \le k_1 \le 2^t-1$ are all different. As the bits $a_{t-1..0}$ are independent of k_1, the 2^t elements are stored in different modules. ❑



module	0	1	2	3	4	5	6	7
	0	1	2	3	4	5	6	7
	9	8	11	10	13	12	15	14
	18	19	16	17	22	23	20	21
	27	26	25	24	31	30	29	28
	36	37	38	39	32	33	34	35
	45	44	47	46	41	40	43	42
	54	55	52	53	50	51	48	49
	63	62	61	60	59	58	57	56
	64	65	66	67	68	69	70	71
				⋮				

Figure 3: XOR-based linear transformation and mapping of the address space when m=t=3 and s=3.

Lemma 3: The families of strides that produce T-matched CTP_x are those defined by x = 0, 1, ..., s-1, s.

Proof: If $x \le s$, because of Lemma 2 the elements $(i + k_1 \cdot 2^{s-x}) \bmod P_x$ for $0 \le k_1 \le 2^t-1$ are in different modules. Taking as values for i = 1, 2, ..., 2^{s-x} we obtain 2^{s-x} subsequences of 2^t elements mapped into different modules, so each module contain 2^{s-x} elements; therefore CTP_x is T-matched. On the other hand, if x > s the elements are mapped into just $\lceil 2^{s+t-x} \rceil$ modules, so not all modules are visited (s+t-x < t = m). Therefore, CTP_x is not T-matched. ❑

Theorem 1: The families of strides defined by $s-N \le x \le s$ where N = min(λ-t, s) produce T-matched vectors of length $L = 2^\lambda$.

Proof: For a vector to be T-matched it is sufficient that CTP_x be T-matched and that $L = k \cdot P_x$ (Lemma 1). Because of Lemma 3, CTP_x is T-matched for all $x \le s$. If $x \ge s-N$ then $x \ge s - (\lambda - t)$ for the definition of N; this can be rewritten as $\lambda \ge s + t - x$, or $L = k \cdot P_x$ for some k > 0. Therefore the vector is T-mached. ❑

Note that for x < s-N when λ-t < s it is possible for a vector with length $L = 2^\lambda$ to be T-matched, but this depends on its initial address. Since we consider vectors with any initial address, these cases are not of interest.

Corollary: For fixed λ and t, the value of s defines a window of families of strides that produce T-matched vectors.

Up to now we have shown the conditions for a vector to be T-matched. However, the access in order can lead to a high latency because of an unsuitable canonical temporal distribution. In the example of Figure 1, the vectors of length 64 are T-matched for $0 \le x \le 3$. Consider the access of a vector with stride 12 and whose first element is in position 16. Since x = 2 the period is $P_x = 16$ and the CTP_x is

2, 7, 5, 2, 0, 5, 3, 0, 6, 3, 1, 6, 4, 1, 7, 4

and this sequence is repeated for each of the four periods of the vector. The access is not conflict free. In fact only the family with x = 3 produces a conflict-free canonical temporal distribution.

3.1. Reordering

We now show how to reorder the access of the vector elements so as to achieve a better temporal distribution.

Theorem 2: The elements of a T-matched vector with length $L = k \cdot P_x$ for some k > 0 can be grouped in subsequences such that the temporal distribution of each subsequence is conflict free.

Proof: Since $L = k \cdot P_x$, the vector can be divided in k subvectors of length P_x. In each of these subvectors we use the 2^{s-x} subsequences defined in Lemma 2. Since the elements of each subsequence are mapped in different modules the temporal distribution of each subsequence is conflict free.❑

For the calculation of the addresses of the consecutive elements in a subsequence it is necessary to increment by $\sigma \cdot 2^s$, instead of $\sigma \cdot 2^x$ for the canonical order. The order of the subsequences is not important. One possibility is to request all subsequences in one period and then go to the following period, and so on. In such a case, the first elements of consecutive subsequences in one period are separated by $\sigma \cdot 2^x$, which is also the separation between the last element of one period and the first of the next. The control to perform the requests in this order is shown in Figure 4.

The hardware required to generate the addresses is shown in Figure 5. To simplify the implementation it is convenient that the compiler issues instructions to load the values $\sigma \cdot 2^x$, $\sigma \cdot 2^s$ and 2^{s-x}. If this is done, the complexity is practically the same as that for the case in which requests are in order.

For the previous example, for the first period we obtain two subsequences that contain the vector elements (0, 2, 4, 6, 8, 10, 12, 14) and (1, 3, 5, 7, 9, 11, 13, 15), respectively. These are located in modules (2, 5, 0, 3, 6, 1, 4, 7) and (7, 2, 5, 0, 3, 6, 1, 4). Note that even if each subsequence is conflict free, the access to the whole vector is not, because the temporal

distributions of the different subsequences are not the same. Consequently, in the next section we give an additional reordering that provides this conflict-free access.

However, it is worth noticing that with two buffers at the input of each module and one buffer at the output, the above mentioned ordering produces a latency which is not greater than 2T+L cycles, that is, the increase in latency due to the non conflict-free access is at most of T-1 cycles [15].

```
SUB = A1      ;SUB is the initial address of the present
              ;subsequence
A = A1        ;A is the request address
for K=1 to 2^(λ-(s+t-x))      ;k is the period number
    for J=1 to 2^(s-x)        ;j is the subsequence number
        for I=2 to 2^t        ;the first address of each sub-
                              ;sequence is obtained outside
                              ;the loop
            A= A + σ·2^s
        end for
        if J<2^(s-x) then
            (SUB = SUB + σ·2^x  ||  A = SUB + σ·2^x)
                              ;in parallel
    end for
    SUB = A + σ·2^x  ||  A = A + σ·2^x
end for
```

Figure 4: Control to perform the memory requests.

3.2. Conflict-free Ordering

Even though the additional latency associated with the ordering described in the previous section is low in practice (since L>>T), we now propose a scheme that eliminates this additional latency and permits the access of the whole vector in a conflict-free manner.

To achieve this, it is necessary to incorporate a second reordering so that the temporal distribution of all subsequences is the same. However, this poses a problem with the calculation of the addresses inside the subsequences, since to have a simple incremental calculation (adding $\sigma \cdot 2^s$) it is necessary to do this in the order described in the previous section. The solution is to decouple the calculation of the addresses from the actual requests. This is achieved by calculating the addresses of subsequence i+1 while accessing subsequence i. Consequently, during the first 2^t cycles, it is necessary to calculate the addresses of the first subsequence (which are used immediately for memory access) and of the second subsequence (which are stored in a set of latches for access as the next subsequence). After that, for each subsequence, the addresses for access are obtained from the latches and a new address is calculated to store. Consequently, as shown in Figure 6, two address generators are needed, although one of them is only used in the first 2^t cycles. Moreover, it is necessary to store the temporal distribution of the first subsequence, which is used to control the order of the requests of the following subsequences. In addition to the latches in the processor, no buffers are needed in the memory modules.

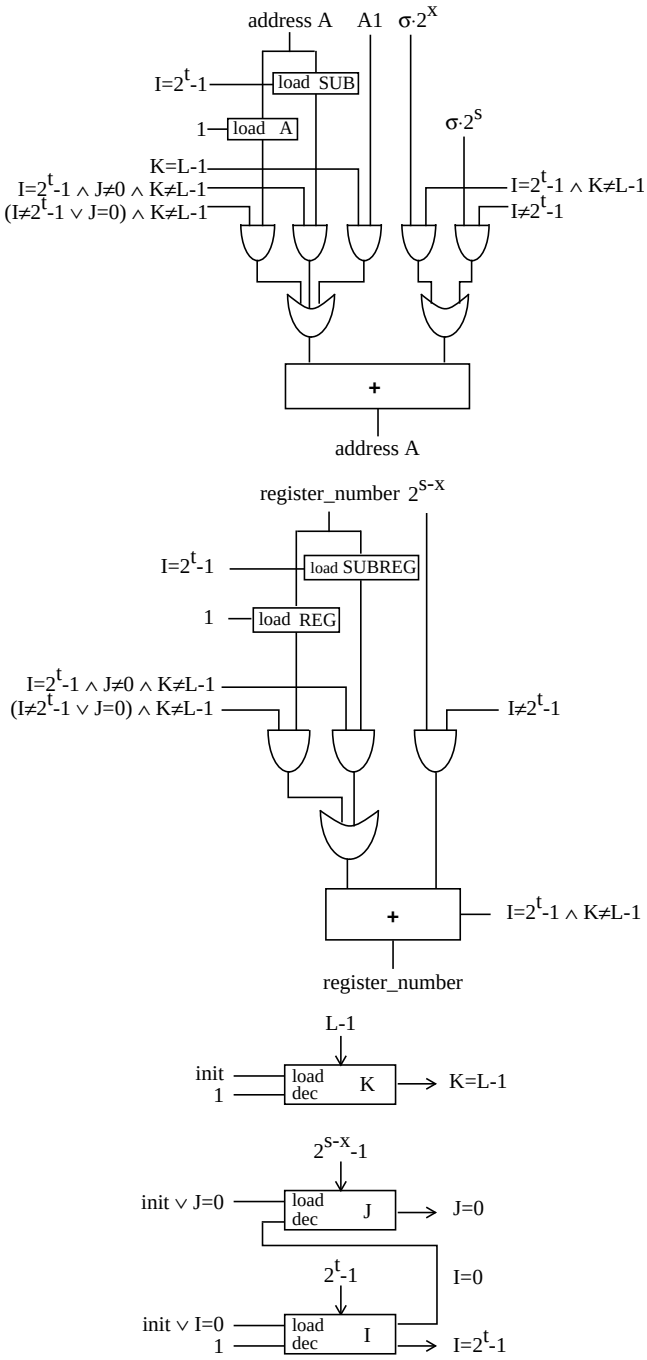


Figure 5: Hardware for address calculation and register addressing.

3.3. Choice of s

As shown previously, the proposed scheme achieves conflict-free access to the families of strides $\sigma \cdot 2^x$ such that $s-N \le x \le s$, and the choice of s determines the window of conflict-free strides. Since the family for x = 0 includes all the odd strides (and in particular stride one), it is certainly convenient to include this family by making $s \le \lambda - t$; the largest window occurs when $s = \lambda - t$. In such a case, the conflict-free strides belong to the families with $0 \le x \le \lambda - t$.

For example, for L = 128 and m = t = 3 we choose s = 4 and obtain conflict-free access for the families defined by x = 0, 1, 2, 3, 4.

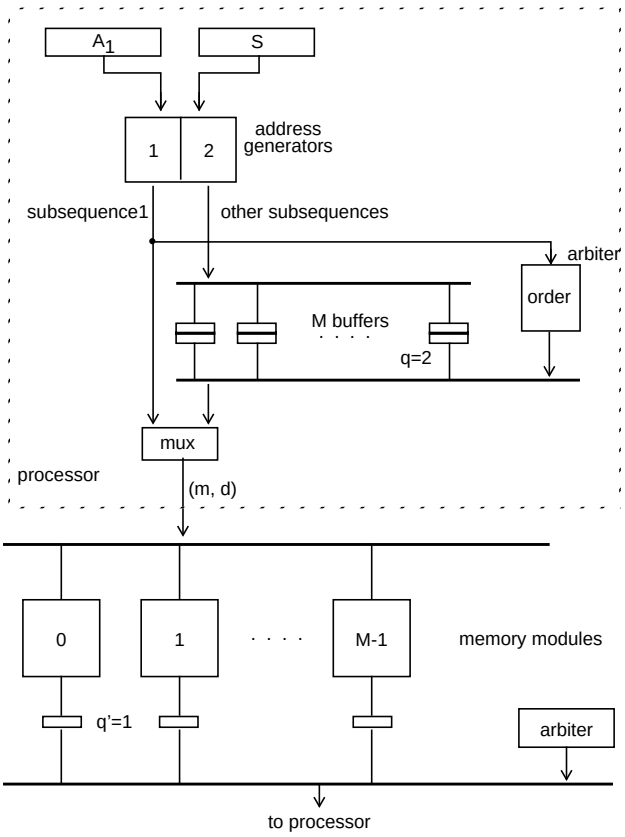


Figure 6: Architecture model for out-of-order memory accesses

4. Unmatched Memory

One way to increase the number of families of strides that produce conflict-free access, is to increase the number of modules (m > t).

A possible mapping for this case is to use the same one as defined in (1) replacing t by m. In such a case, the period P_x is $\lceil 2^{s+m-x} \rceil$, and for ordered access, any vector length and any initial address, conflict-free access is obtained for the families defined by x = s, s+1,..., s+m-t [6].

This can be combined with the technique presented in Section 3 for vectors of length L, so that conflict-free access is obtained for the families defined by

$$x = s-N, s-N+1, ..., s, s+1, ..., s+m-t$$

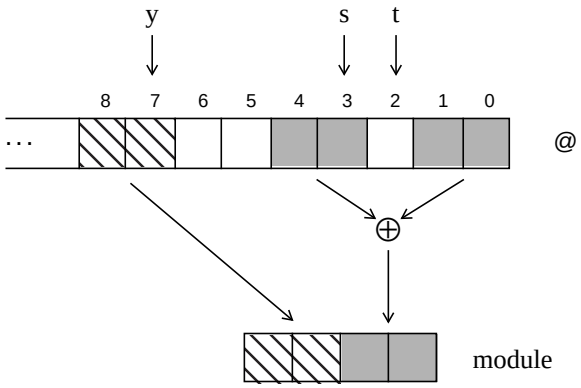
In this case, the vectors with strides belonging to the families x = s, s+1, ..., s+m-t are accessed in order and the rest are accessed out-of-order using the results of Section 3. In particular, if s = λ-t, the conflict-free families have $0 \le x \le \lambda+m-2t$.

4.1. A Better Mapping

We now consider a way of increasing the number of conflict-free strides for vectors of length L and out-of-order access. To simplify our discussion we consider the special case where m=2t. For this case we use F as the following mapping:

$$b_i = \begin{cases} a_i \oplus a_{s+i} & 0 \le i \le t-1,\ s \ge t \\ a_{y+i-t} & t \le i \le 2t-1,\ y \ge s+t \end{cases} \quad (2)$$

For the rest of this section we assume this mapping. An example for t = 2, m = 4, s = 3 and y = 7 is given in Figure 7.



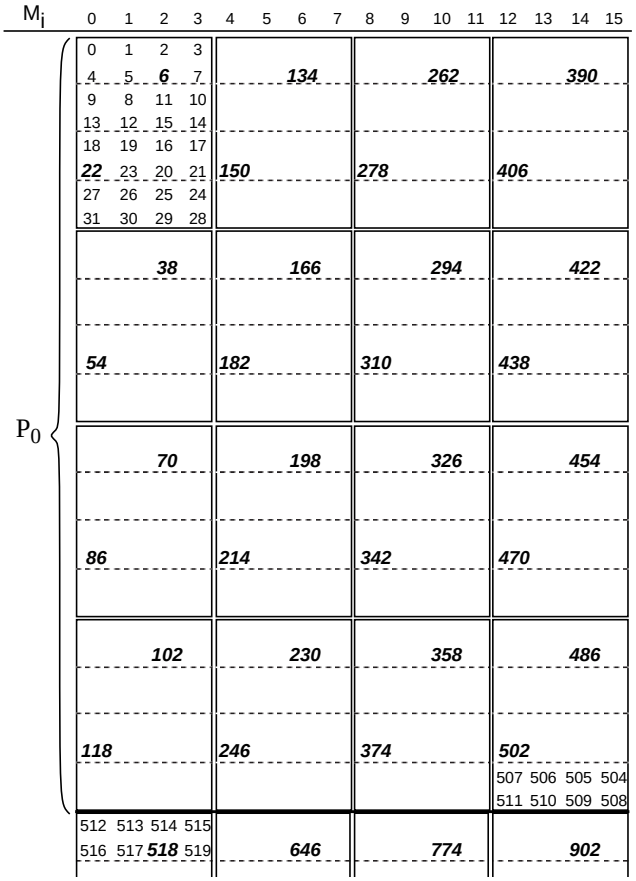


Figure 7: XOR-based transformation and mapping of the address space when m=4, t=2, s=3 and y=7. In italic the elements of a vector with λ=5 with initial address A_1=6 and stride S=16 are shown.

This mapping corresponds to a division of the modules into T sections of T modules and of the address space into blocks of 2^y locations; each block is mapped into one section, using the mapping defined by the lower t bits of b.

The period for a stride of the family $\sigma \cdot 2^x$ is $P_x = \lceil 2^{y+t-x} \rceil$.

Lemma 4: Let $x \le y$ and P_x be the corresponding period. Consider the grouping of these P_x elements into 2^{y-x} subsequences consisting of 2^t elements each. The i-th subsequence ($1 \le i \le 2^{y-x}$) contains the elements ($i + k_1 \cdot 2^{y-x}$) with $0 \le k_1 \le 2^t-1$. For any of these subsequences, all its elements are located in different memory sections.

Proof: The addresses of the elements in the i-th subsequence are given by

$$A_i + k_1 \cdot 2^{y-x} \cdot \sigma \cdot 2^x = A_i + k_1 \cdot \sigma \cdot 2^y$$

with $0 \le k_1 \le 2^t-1$. Since σ is odd, all these addresses have different combinations in the bits $a_{y+t-1..y}$, so all 2^t sections are visited. ❑

Lemma 5: The families of strides that produce T-matched CTP_x are those defined by x = 0, ..., y.

Proof: Because of Lemma 4, there are 2^{y-x} elements of CTP_x mapped in each section. If $x \ge s+t$ all 2^{y-x} elements are located in the same module inside the section, so each one of the T modules that appear in CTP_x contain P_x/T elements. Therefore CTP_x is T-matched. If $x < s+t$, more modules are visited in each section, so each one contains less than P_x/T elements. CTP_x is, therefore, also T-matched.

On the other hand, if $x > y$ just $\lceil 2^{y+t-x} \rceil$ modules are visited, so CTP_x is not T-matched. ❑

Theorem 3: The families of strides defined by x = s-N, .., s and x = y-R, .., y where N = min(λ-t, s) and R = min(λ-t, y) produce T-matched vectors of length $L = 2^\lambda$.

Proof: Let x = s-N, ..., s, so $\lambda \ge s+t-x$. The elements ($i + k_1 \cdot 2^{s-x}$) with $0 \le k_1 \le 2^t-1$ have different combinations in the bits $a_{s+t-1..s}$ (see Lemma 2), and therefore are located in at least T modules, exactly T if the bits $a_{y+t-1..y}$ are not modified. Let's assume for the moment that those bits are not modified. Taking as values for $i = k_2 \cdot 2^{s+t-x} + k_3$ with $0 \le k_2 \le (L/2^{s+t-x})-1$ and $1 \le k_3 \le 2^{s-x}$ we obtain that each of the T modules contain L/T elements. If the bits $a_{y+t-1..y}$ are modified then the elements are distributed in more modules, containing each one less than L/T elements. Therefore, the vector is T-matched.

Let x = y-R, ..., y. Then, $\lambda \ge y+t-x$, so $L = k \cdot P_x$ for some $k > 0$. Moreover, because of Lemma 5, CTP_x is T-matched. Therefore the vector is T-matched. ❑

To correctly partition the set of families of strides, we will assume from now on that $y-R \ge s+1$; this implies $R = \lambda-t$ and $y \ge \lambda-t$. Therefore there are two groups of families of strides, those with x in [s-N, s] and those in [y-R, y]. As a consequence of the previous lemmas and theorem, depending on the value of x, one of the following subsequences is used:

i) for $s-N \le x \le s$, we define subsequences as stated in Lemma 2. We know that its elements are mapped into different modules because they have different combinations in the bits s+t-1..s; therefore the temporal distribution is conflict free. For some values of σ and A_1, also the bits y+t-1..y of the elements in one subsequence may vary; in this case many sections are visited in one subsequence, which also leads to a conflict-free temporal distribution of the subsequence.

ii) for $y-R \le x \le y$, we define subsequences as stated in Lemma 4. Its elements are mapped into different sections, so its temporal distribution is conflict free.

For example, consider the mapping shown in Figure 7 and let x=4, σ=1, A_1=6 and L=32. The elements of the period P_x are marked in italic in Figure 7. There are eight subsequences in one period, corresponding to the elements of the vector (0, 8, 16, 24), (1, 9, 17, 25), (2, 10, 18, 26), ..., (7, 15, 23, 31). These elements are located in the modules (2, 6, 10, 14), (0, 4, 8, 12), (2, 6, 10, 14), ..., (0, 4, 8, 12).

For the calculation of the addresses we could use the same algorithm as in section 3.1, where the increment of address in the inner loop is either $\sigma \cdot 2^s$ or $\sigma \cdot 2^y$.

As in section 3, we have obtained subsequences whose temporal distribution is conflict free; however this might not be the case for the whole vector. For example, consider x=6, σ=3 and A_1=0. In this case, P_x=8, so there are two subsequences in one CTP_x, corresponding to the vector elements (0, 2, 4, 6) and (1, 3, 5, 7). These elements are located in the modules (0, 12, 8, 4) and (4, 0, 12, 8) respectively. Again, next we give an additional reordering that provides conflict-free access to the whole vector.

4.2. Conflict-free reordering for unmatched memory

In the case of matched memory, all subsequences contain all modules, so it is sufficient to remember the order in which the first one is requested and use it to access the remaining subsequences. This is not the case for unmatched memory, since different subsequences may contain different modules.

To achieve the conflict-free access to the whole vector we will apply the same strategy as in section 3.2 with some modifications depending on the value of x, as explained below. The following definition is useful:

Definition: A section is composed of 2^t modules labeled 0 to 2^t-1. The SUPERMODULE i consists of the i-th module of each section. That is, the supermodule number of an address is determined by the bits $a_{s+t-1..s}$.

Two cases have to be considered, as follows:

i) $s-N \le x \le s$: for this case, as stated before, the subsequences used are those defined in Lemma 2. Since the 2^t elements in these subsequences have different combinations in the bits $a_{s+t-1..s}$, all supermodules appear in each subsequence. Therefore we can apply the same strategy as in section 3.2 but at the supermodule level, i.e.: the supermodule numbers of the first subsequence must be remembered and the elements of the remaining subsequences should be requested in

the same "supermodule order" as the first one. Note that this implies that two latches are needed per supermodule, not per module. This results in $2 \cdot 2^t$ latches (not $2 \cdot 2^m$).

ii) $y-R \le x \le y$: now the subsequences are defined by Lemma 4. Since its elements are in different sections, it is sufficient to request each subsequence in the section order of the first subsequence.

In summary,

i) for $x \le s$ the supermodule order of the first subsequence is stored (bits $b_{t-1..0}$) and the latches are labeled by supermodule.

ii) for $x > s$, the section order of the first subsequence is stored (bits $b_{2t-1..t}$) and the latches are labeled by section.

4.3. Choice of s and y

For the reordering proposed, we know how to obtain a conflict-free access to the families with x = s-N, ..., s and x = y-R, ..., y; making y-R = s+1 a single window of N+R+2 families is obtained. As discussed in Section 3, a convenient choice is $s=\lambda-t$; for this case, to achieve the largest possible single window, we choose

$$y = \lambda-t+1+\lambda-t= 2\ (\lambda-t)+1$$

Consequently, the conflict-free families correspond to

$$0 \le x \le 2\ (\lambda-t)+1$$

Compared to the mapping used at the beginning of Section 4, this provides $\lambda-t+1$ additional families.

For example, for L = 128, T = 8 and M = 64 we choose s = 4 and y = 9 and obtain conflict-free access for the families defined by x = 0, 1, ..., 9.

5. Evaluation

In this Section we present a discussion of the effectiveness of the proposed scheme, in terms of its efficiency, the cost of implementation, and its completeness to handle any stride and any vector length. Moreover, we comment on the possibility of using chaining of LOAD/STORE and EXECUTE in important particular cases.

A) Fraction of strides that are conflict free.

We now determine the fraction f of conflict-free strides for the choices of s and y presented in sections 3 and 4.

Since the fraction of strides belonging to family $\sigma \cdot 2^x$ is $1/2^{x+1}$, the fraction of conflict-free strides produced by a window from x=0 to x=w is

$$\sum_{i=0}^{w} \frac{1}{2^{i+1}} = 1 - \frac{1}{2^{w+1}}$$

Consequently, for the matched memory system case ($w=\lambda-t$) we get

$$f = 1 - \frac{1}{2^{\lambda-t+1}}$$

and for the unmatched case ($w =2(\lambda-t)+1$)

$$f = 1 - \frac{1}{2^{2(\lambda-t+1)}}$$

For example, for L = 128 and M = T = 8, we choose s = 4 and get 31/32th conflict-free strides. Moreover, this set of strides probably includes the most-frequently used ones.

If we increase the number of memory modules to M = 64 and keep T = 8, for s = 4 and y = 9 we get 1023/1024th conflict-free strides.

B) Access of non conflict-free strides and efficiency assuming uniform distribution of strides.

The families of strides included in A) are conflict free so one element can be obtained from memory per cycle after the startup of T+1. The rest of families are not conflict free because the elements of vectors of the family with $x=w+i$ ($i > 0$) are located in $\lceil 2^{t-i} \rceil$ modules and one element is accessed every $2^t / \lceil 2^{t-i} \rceil$ cycles in average. Consequently, the efficiency η can be approximated by

$$\eta = \frac{1}{1 + \frac{t}{2^{w+1}}}$$

where w is the boundary of the conflict-free window.

For the matched example given before the efficiency is $\eta=0.914$ whereas for the unmatched memory case it is $\eta=0.997$. In comparison, for ordered access the highest efficiency is obtained for s=0 (to have conflict-free access for odd strides). This results in $\eta=0.4$ for the matched memory case and $\eta=0.84$ for the unmatched one.

C) Vectors with length different than L.

As indicated in the introduction, the scheme is designed for vectors of length L equal to the number of elements of the vector registers. Also, we have indicated that a large fraction of the vectors will have this length since strip-mining is used for longer vectors. This leaves a small fraction of shorter vectors.

Moreover, in some vector processors there are vector registers of several lengths; normally these lengths are a multiple of the length of the shortest vector register. In such a case, L would correspond to the length of the smallest vector register and the others would have lengths which are a multiple of L. This situation can also appear in memory-access units for scalar processors, as discussed in the introduction.

Consequently, the following cases can occur:

i) the length of the vector is smaller than L. In this case two alternatives exist:

- access the vector in order. This would produce the inefficiency of the resulting temporal distribution.
- use the scheme presented in Sections 3 and 4 if the length V is related to the family of strides x so that

$$V = k \cdot 2^{w+t-x}$$

where w is either s or y. This is because this

length satisfies the requirements for the proposed reordering.

In general, divide the vector into two parts, one of length equal to V above and the other of the rest. Access the first part using the scheme of Sections 3 and 4 and the second part in order. This separation can be done by the compiler.

ii) the length of the vector is a multiple of L. This occurs for the case of multiple-size registers. In such a case the same scheme described in Sections 3 and 4 is used for each portion of length L. The overall efficiency is the same as for vectors of length L.

D) Complexity of the hardware: address calculation, buffers, arbitration, and register addressing.

As shown in Figure 6, the address calculation for conflict-free access in out-of-order mode requires two address generators instead of one for the standard ordered scheme. Moreover, it requires a buffer of size 2T, a queue to store the temporal distribution of the first subsequence and an arbiter to issue the remaining subsequences in the same order. Finally, a controller is required for the address calculations in the required order. We estimate that the hardware cost of these components is a minor part of the cost of the memory subsystem.

To achieve the required throughput, it might be necessary to pipeline the adder (this would also be needed in the standard ordered access and the latency of the adder would also have to be added to the latency of the vector access, unless several vector accesses are chained together). Moreover, the usual techniques have to be used to eliminate the dependencies in the calculation of successive addresses.

In addition to the buffers mentioned before, no additional ones are needed. This is in contrast with other proposals that include a significant number of buffers to eliminate the effect of an unsuitable temporal distribution.

To support the out-of-order access, elements of the vector register have to be addressed out of order. Consequently, this register has to be of the random access type, whereas for ordered access and return a FIFO organization is adequate.

E) Efficiency of more memory modules.

As has been seen in Section 4, the addition of modules to make the memory unmatched increases the number of families of strides that are conflict free. However, this is obtained at a large expense because to double the number of conflict-free families it is necessary to square the number of modules. This is aggravated by the fact that the added families contain fewer strides and that these strides are probably less frequently used.

Of course, the addition of memory modules can be justified by other reasons, such as simultaneous access to several vectors and non vector access.

F) Possibility of chaining of LOAD and EXECUTE.

As mentioned in the introduction, the complicated timing produced when access in order is coupled with buffers, makes it impractical to chain two instructions if one of the operands of the second is being obtained with a LOAD. In contrast, the scheme proposed produces one vector element each cycle in a deterministic order (for conflict-free strides). Consequently, it is possible to perform the chaining if the first instruction is executed using the same order of elements as the LOAD. Note that the sequence of addresses to the register elements is produced anyhow as part of the LOAD.

G) Maximum number of conflict-free families for the unmatched case.

The number of conflict-free families obtained in section 4 is not the maximum achievable with out-of-order access. In fact, it is possible to have t-1 more families [15]. However, the structure of the subsequences for these t-1 additional families is different that presented. Because of this, the inclusion of these families would complicate the hardware for address generation and access control.

H) Conflict-free families and vector length.

For unmatched memory, the access in order produces at most t+1 conflict-free families for any vector length (for m = 2t). In contrast, the scheme we propose produces only two conflict-free families for any vector length, but increases to $2(\lambda-t+1)$ the number of conflict-free families for vectors of length $L = 2^{\lambda}$.

6. Conclusions

In this paper we have considered the access of vectors of fixed length, equal to the length of a vector register. The access patterns correspond to constant strides and the vector can begin in any address. The basic idea we propose is an out-of-order access of the elements of the vector to achieve conflict-free access for all strides that produce T-matched vectors. We first consider the matched memory case, where M=T. In this case, we obtain a window of $\lambda-t+1$ families of strides that are conflict free, whereas previous schemes that perform the access in order result in a single conflict-free family (for vectors of any length). To achieve this, we divide the vector in subvectors which are accessed in a conflict-free manner. This by itself does not produce conflict-free access to the whole vector, although the added latency is low. To achieve the conflict-free access to the whole vector we propose that an additional set of T addresses be calculated and latched, so that the temporal distribution of all subsequences is the same. The analysis of the required hardware for address calculations shows that, with compiler support, the complexity is similar to that of the address generator for access in order.

We present then an extension of this scheme for the unmatched memory case. For a number of modules $M=T^2$, the size of the conflict-free window is doubled; this compares favorably to the t+1 conflict-free families obtained with ordered access. Note however, as we discuss in Section 5, that the resulting increase in efficiency from the matched to the unmatched case is obtained at the cost of squaring the number of memory modules.

We discuss also the access of vectors that are shorter than the size of the vector registers. In this case, depending on the stride family, we propose a combination of out-of-order and ordered access. If the length of the vector is known at compile time, the division of the vector in this two subvectors can be done by the compiler.

The ideas have been presented using an address mapping based on linear transformations. However, the same results can be achieved with interleaving or with skewing. For this, it is necessary to select in a suitable manner the bits that determine the module number in the interleaved case, and the number of rows to rotate for skewing. The difference between these schemes is the behavior for vectors of length smaller than L.

We plan to extend this work to the case in which several vectors are accessed simultaneously, either in a single processor with several memory ports or in a multiprocessor with vector processors. We also will explore further the use of these techniques for scalar processors with decoupled memory access and execute units.

7. Acknowledgments

This work has been supported by the Ministry of Education of Spain under contract TIC-299/89 and by the CEPBA (European Center for Parallelism of Barcelona).

8. References

1. P. Budnik and D. J. Kuck, "The Organization and Use of Parallel Memories", IEEE Trans. on Computers, vol. C-20, no. 12, pp. 1566-1569, 1971.
2. D.H. Lawrie, "Access and Alignment of Data in an Array Processor", IEEE Trans. on Computers, vol. C-24, no. 12, pp. 1145-1155, Dec. 1975.
3. J. Frailong, W. Jalby and J. Lenfant, "XOR-schemes: A Flexible Data Organization in Parallel Memories", Int'l Conference on Parallel Processing, pp. 276-283, 1985.
4. H.A.G. Wijshoff and J. van Leeuwen, "The Structure of Periodic Storage Schemes for Parallel Memories", IEEE Trans. on Computers, vol. C-34, pp. 501-505, June 1985.
5. D.T. Harper III and J.R. Jump, "Performance Evaluation of Vector Accesses in Parallel Memories Using a Skewed Storage Scheme", Int'l Symposium on Computer Architecture, pp. 324-328, 1986.
6. D.T. Harper III, "Block, Multistride Vector and FFT Accesses in Parallel Memory Systems", IEEE Trans. on Parallel and Distributed Systems, vol. 2, no. 1, pp. 43-51, 1991.
7. C-L. Chen and C-K Liao, "Analysis of Vector Access Performance on Skewed Interleaved Memory", Int'l Symposium on Computer Architecture, pp. 387-394, 1989.
8. S. Weiss, "An Aperiodic Storage Scheme to Reduce Memory Conflicts in Vector Processors", Int'l Symposium on Computer Architecture, pp. 380-386, 1989.
9. A. Norton and E. Melton, "A Class of Boolean Linear Transformations for Conflict-Free Power-of-Two Stride Access", Int'l Conference on Parallel Processing, pp. 247-254, 1987.
10. B. R. Rau, M. S. Schlansker and D. W. L. Yen, "The Cydra™ 5 Stride-Insensitive Memory System", Int'l Conference on Parallel Processing, pp. 242-246, 1989.
11. D.T. Harper III and D. A. Linebarger, "Conflict-Free Vector Access Using a Dynamic Storage Scheme", IEEE Trans. on Computers, vol. 40, no. 3, pp. 276-283, 1991.
12. B.R. Rau, "Pseudo-Randomly Interleaved Memory", Int'l Symposium on Computer Architecture, pp. 74-83, 1991.
13. D.T. Harper III and Y. Costa, "Analytical Estimation of Vector Access Performance in Parallel Memory Architectures", Internal Report, Dept. of Electrical Engineering. The University of Texas at Dallas, 1991.
14. W. Oed and O. Lange, "On the Effective Bandwidth of Interleaved Memories in Vector Processing Systems, IEEE Trans. on Computers, vol. C-34, no. 10, pp. 949-957, October 1985.
15. M. Valero et al., "Conflict-Free Access to Vectors", Research Report, Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, UPC/DAC RR91-22, October 1991.