CONFLICT-FREE STRIDES FOR VECTORS IN MATCHED MEMORIES

MATEO VALERO, TOMÁS LANG, JOSÉ M. LLABERÍA,
MONTSE PEIRON, JUAN J. NAVARRO and EDUARD AYGUADÉ
*Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya,
Gran Capità s/n Mòdul D4, 08034 Barcelona, Spain
E-mail: mateo@ac.upc.es*

ABSTRACT
Address transformation schemes, such as skewing and linear transformations, have been proposed to achieve conflict-free access to one family of strides in vector processors with matched memories.
In this paper, we extend these schemes to achieve this conflict-free access for several families. 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. The hardware required is similar to that for the access in order.

Keywords: Conflict-free Access, Out-of-order Access, Parallel Memory Architecturess, Storage Schemes, Temporal Distribution, Vector Access.

1. Introduction

To have a sufficient memory bandwidth, the memory of vector processors is organized as several modules that can be accessed simultaneously. The memory is matched if the number of memory modules is equal to the ratio between the memory cycle and the processor cycle, since in this case the peak memory throughput is one word per processor cycle. However, to obtain this peak throughput, the request sequence has to be such that there are no conflicts in the accesses. This is achieved, for example, with standard memory interleaving and for vectors of odd strides. However, this is not the case for other strides, which has motivated the proposal of other addressing schemes.

The two main address transformation schemes used to achieve conflict-free access to other strides are skewing and linear transformations. These schemes were initially proposed for array processors [1, 2] and later for multiprocessors [3], vector processors [4, 5], and VLIW processors [6]. For vectors, 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 [7]. Moreover, for the case in which different vectors are accessed with different strides, dynamic schemes based on skewing [7] and on linear transformations [5] were proposed. Linear transformations have the advantage over skewing that the module number is simpler to compute.

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 [4]. 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 [6, 8, 9]. Recently a proposal has been made [10] for an analytic model that can be used to make comparisons among these methods. 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

Electronic version of an article published as :
Parallel processing letters, vol. 1, núm.2, 1991, p.95-102. DOI 10.1142/S0129626491000045

a function of several parameters, such as structure of the transformation, number of buffers, and number of memory modules. Although in [6, 8, 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.

In this paper, we consider the case in which the processor accesses vectors of fixed 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. Moreover, we propose that the elements of the vector be requested out of order and that the whole vector be stored in the 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.

To introduce the out-of-order accessing we use a linear transformation of the addresses, although the same results can be obtained with interleaving or with skewing. We show that this mode of accessing results in conflict-free accesses for several families of strides, whereas, as discussed in previous works, ordered access produces conflict-free access only to one family.

2. Model

The memory is composed of $M=2^m$ modules and the module latency is of M processor cycles (matched system). Each memory module has input and output buffers.

The processor requests one element per (processor) cycle unless it has to wait because the memory module is busy and the associated input buffers are 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 zero delay in the interconnection network.

The vector length is equal to $L=2^\lambda$, which is a multiple of 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 $A_1+S\cdot(i-1)$. As done in [7], 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 $a_{n-1}...a_0$ 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 $b_{m-1}...b_0$ is given by $b = F(A)$ where F is the module component of the address mapping.

We choose as F the linear transformation

$$b_i = a_i \oplus a_{s+i} \quad s \geq m,\ 0 \leq i \leq m - 1 \tag{1}$$

This mapping has the property that when the elements of the vector are requested in order the access is conflict free for strides of the family with x = s and for vectors of any length and any initial address [5].

Figure 1 illustrates a portion of the mapping for M=8 and s=3.

M_i	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
				⋮				

Fig. 1: Mapping of address space for (1) with M=8 and s=3

3. Balanced Vectors

We now discuss 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. A vector is BALANCED if SD(i)= L/M for all i.

Definition: The TEMPORAL DISTRIBUTION TD 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 out of order.

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

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).

In a matched memory system, a temporal distribution is conflict free if any subset of M consecutively requested elements are located in the M memory modules.

Definition: The period P of an address mapping is the period of the canonical temporal distribution for a vector with stride 1 [5].

For the mapping defined in (1), the period is $P = 2^{s+m}$ [5].

Definition: The period P_x of an address mapping is the period of the canonical temporal distribution for a vector with stride $\sigma \cdot 2^x$. For our mapping $P_x = \lceil 2^{s+m-x} \rceil$ [5].

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 BALANCED if it contains each module 2^{s-x} times.

Lemma 1: *Let $x \leq s$ and P_X be the corresponding period. Consider the grouping of these*

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

Proof: Let A_i, with binary representation $a_{n-1}a_{n-2}...a_1a_0$, be the address of the i-th element. For the mapping defined in (1), this element is located in module m_i such that

$$m_i = (a_{s+m-1} \oplus a_{m-1}, ..., a_{s+1} \oplus a_1, a_s \oplus a_0) = a_{s+m-1..s} \oplus a_{m-1..0} \qquad (2)$$

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

$$A_i + k_1 \cdot 2^{s-x} \cdot \sigma \cdot 2^x = A_i + k_1 \cdot \sigma \cdot 2^s \qquad (3)$$

and is located in module

$$((a_{s+m-1..s} + k_1 \cdot \sigma \bmod 2^m) \bmod 2^m) \oplus a_{m-1..0} \qquad (4)$$

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

Lemma 2: *For the mapping defined in (1), the families of strides that produce balanced CTP_x are those defined by $x = 0, 1,.., s$.*

Proof: If $x \le s$, because of Lemma 1 the elements $(i + k_1 \cdot 2^{s-x}) \bmod P_x$ for $0 \le k_1 \le 2^m-1$ are in different modules. Taking as values for i 1, 2, ..., 2^{s-x} we obtain 2^{s-x} subsequences of 2^m elements mapped into different modules, so each module contain 2^{s-x} elements; therefore CTP_x is balanced.

On the other hand, if $x > s$ the elements are mapped into just $\lceil 2^{s+m-x} \rceil$ modules, so not all modules are visited ($s+m-x < m$). Therefore, CTP_x is not balanced ❑.

Lemma 3: *If CTP_x is balanced and $L = k \cdot P_x$ with $k > 0$ then the vector is balanced.*

Proof: This is evident since the length of the vector is a multiple of the period P_x, which is the length of CTP_x ❑.

Theorem 1: *The families of strides that produce balanced vectors correspond to the values of $x = s-N,..., s-1, s$ where $N = min(\lambda-m, s)$.*

Proof: Because of the address mapping used, $s \ge m$. Because of Lemma 3, for a vector with stride $S_x = \sigma \cdot 2^x$ to be balanced it is necessary that CTP_x be balanced and that $L = k \cdot P_x$. Since $P_x = 2^{s+m-x}$, this last property can be expressed as $\lambda \ge s+m-x$. Moreover, because of Lemma 2, CTP_x is balanced if $x \le s$. Consequently, we get the following two cases:

(i) if $\lambda \ge s+m$ then $0 \le x \le s$

(ii) if $\lambda < s+m$ then $s-(\lambda-m) \le x \le s$ ❑.

Corollary: *For fixed λ and m, the value of s defines a window of families of strides that produce balanced vectors.*

Up to now we have shown the conditions for a vector to be balanced. 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 balanced 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 (in general with x = s) produces a conflict-free access.

3. 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 balanced vector can be grouped in subsequences such that the temporal distribution of each subsequence is conflict free.*

Proof: We know that the length of the vector is $L = k \cdot P_x$ for some $k > 0$ because the vector is balanced. Then the vector can be divided in k groups of length P_x. In each of these groups we define 2^{s-x} subsequences as defined in Lemma 1. 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 then:

```
SUB = $A_1$          ;SUB is the initial address of the present subsequence
for k=1 to $2^{\lambda-m-s+x}$      ; k is the period number
   for j=1 to $2^{s-x}$         ; j is the subsequence number
      A = SUB              ; A is the request address
      for i=2 to $2^m$
         A= A + $\sigma \cdot 2^s$
      end for
      SUB = SUB + $\sigma \cdot 2^x$
   end for
   SUB = A + $\sigma \cdot 2^x$
end for
```

The hardware required to generate the addresses is practically the same as that for the requests in order, which consists of the addition of the stride. In our case, it is necessary also to save and update SUB.

For the previous example, for the first period we obtain two subsequences that contain the 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 $2 \cdot M+L$ cycles, that is, that the increase in latency due to the non CF access is at most of M-1 cycles [12].

4. Conflict-free Ordering

Even though the additional latency associated with the ordering described in the previous section is low in practice (since L>>M), 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 subsequence, 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, two adders are needed, although one of them is only used in the first 2^m 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.

5. Choice of s and Fraction of Conflict-Free Strides

As shown previously, the proposed scheme achieves conflict-free access to the families of strides S_x such that $s-N \le x \le s$, where $N = \min(\lambda-m, s)$ 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 $s \le \lambda-m$. In such a case, the conflict-free strides belong to the families for $0 \le x \le s$.

We now determine the fraction of conflict-free strides for the above mentioned choice of s. Since the fraction of strides belonging to family S_x is $1/2^x$, the fraction of conflict-free strides is

$$\sum_{i=0}^{s} \frac{1}{2^i} = 1 - \frac{1}{2^s} \qquad (5)$$

This fraction is quite high for practical values of λ, m and s. For example, for L=128 and M=8 if we choose s=4, we get 15/16th conflict-free strides. Moreover, this set of strides includes probably the most-frequently used ones.

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 balanced vectors. This corresponds to a window of stride families, in contrast with other schemes that provide conflict-free access only to one family.

Specifically, 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 M addresses be calculated and latched, so that the temporal distribution of all subsequences is the same.

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 will be the behavior for vectors of length smaller than L.

Acknowledgements

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

References

1. P. Budnik and D. J. Kuck, The Organization and Use of Parallel Memories, IEEE Trans. Comput., vol. C-20, no. 12 (1971) 1566-1569.
2. J. Frailong, W. Jalby and J. Lenfant, XOR-schemes: A Flexible Data Organization in Parallel Memories, in Proc. Int. Conf. Parallel Processing (1985) 276-283.
3. A. Norton, E. Melton, A Class of Boolean Linear Transformations for Conflict-Free Power-of-Two Stride Access, in Proc. Int. Conf. Parallel Processing (1987) 247-254.
4. D.T. Harper III and J.R. Jump, Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme, IEEE Trans. Comput., vol. C-36 no. 12 (1987) 1440-1449.
5. D.T. Harper III, Block, Multistride Vector and FFT Accesses in Parallel Memory Systems, IEEE Trans. Parallel and Distributed Systems, vol. 2 no. 1, (1991) 43-51.
6. B. R. Rau, M. S. Schlansker and D. W. L. Yen, The Cydra^{TM} 5 Stride-Insensitive Memory System, in Proc. Int. Conf. Parallel Processing (1989) 242-246.
7. D.T. Harper III and D. A. Linebarger, A Dynamic Storage Scheme for Conflict-Free Vector Access, in Proc. Int. Symp. Computer Architecture (1989) 72-77.
8. S. Weiss, An Aperiodic Storage Scheme to Reduce Memory Conflicts in Vector Processors, in Proc. Int. Symp. Computer Architecture (1989) 380-386.
9. B.R. Rau, Pseudo-Randomly Interleaved Memory, in Proc. Int. Symp. Computer Architecture (1991) 74-83.
10. 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).
11. D.T. Harper III and D. A. Linebarger, Conflict-Free Vector Access Using a Dynamic Storage Scheme, IEEE Trans. Comput. vol. 40 no. 3 (1991) 276-283.
12. M. Valero, T. Lang et al., Acceso Libre de Conflicto a Vectores, Research Report RR-91/22, Dept. d'Arquitectura de Computadors. Univ. Politècnica de Catalunya. (1991).