Access to Streams in Multiprocessor Systems

Mateo Valero, Montse Peiron and Eduard Ayguadé

Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya
Gran Capità s/n, Mòdul D4, 08034 - Barcelona (Spain)

Abstract

When accessing streams in vector multiprocessor machines, degradation in the interconnection network and conflicts in the memory modules are the factors that reduce the efficiency of the system. In this paper we present a synchronous access mechanism that allows conflict-free access to streams in a SIMD vector multiprocessor system. Each processor accesses the corresponding elements out-of-order in such a way that in each cycle the requested elements do not collide in the interconnection network. Moreover, memory modules are accessed so that conflicts are avoided.

The use of the proposed mechanism in present architectures would allow conflict-free access to streams with the most common strides that appear in real applications. The additional hardware is described and shown to be of similar complexity as that required for access in order.

1. Introduction

The access to streams in vector multiprocessors is one of the factors that most reduces the efficiency of the system (by the term *stream* we denote a finite length succession of elements whose addresses are equally spaced). Degradation in the interconnection network and conflicts in the access to memory modules lead to this loss of efficiency. The problem is hard to solve and in fact is one of the factors that limit the scalability of these systems.

In order to increase the efficiency of the access, several techniques have been presented in the literature such as the proposal of storage schemes other than interleaving (skewing [1] and linear transformations [2]), the increase of the number of memory modules, and the use of buffers in the interconnection network and at the input and output of each memory module [3,4].

The performance evaluation of the memory system is difficult to do using mathematical models (some works in this direction are [5,6,7]). Other authors use simulation methods based on real or synthetic traces to evaluate the system [8]. Measurements in real systems are done in [9].

In [10] a technique to access streams in multiprocessor systems is presented. The basic idea is that processors access different slices of a stream in a synchronized way. Data are unscrambled from memory to processors avoiding conflicts in the interconnection network and memory modules. In order to use this technique, all the addresses of the vector elements have to be precalculated before starting the access.

With the aim of improving the access to streams in vector uniprocessors, an out-of-order access to the elements of a stream was proposed in [11]. It considers a bus-based system. The basic idea is to request stream elements in an order such that memory conflicts are avoided. This technique, applied to real systems, leads to conflict-free access to streams with the usual values of strides. In [12] the same idea is applied to access, in a conflict-free way, streams with power-of-two strides.

In this paper we use the techniques presented in [11] to efficiently solve the problem stated in [10]. The method described here allows the same number of conflict-free families but it does not need to precalculate the addresses before starting the access; the access is performed in a conflict-free way without any additional latency. On the contrary, we need two address generators and additional control. The cost of the hardware is shown to be of similar complexity as the required for access in order.

2. Model architecture

Our work is based on a vector multiprocessor SIMD architecture. Figure 1 shows the structure of the system; it is composed of $P = 2^p$ vector processors and $M = 2^m$ memory modules grouped in 2^s sections, so there are 2^{m-s} modules per section. Processors are connected to sections through a 2^p-input, 2^s-output multistage interconnection network; The memory latency is $T=2^t$ processor cycles. We assume $s = p$ and an Omega interconnection network [13]. The memory system is matched, i.e., $M = P \cdot T$ (so $m = s + t$).

1066-6192/92 $3.00 © 1992 IEEE

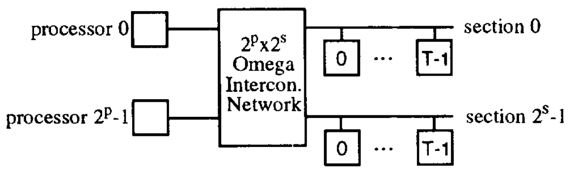


Figure 1: Structure of the SIMD multiprocessor system.

In a SIMD system, each processor requests one element per cycle unless it has to wait due to collisions in the interconnection network or in the memory modules. The length of the vector registers of each processor is $L = 2^\lambda$; we assume $\lambda \geq t$. The first element of the stream has address A_0 and consecutive elements are separated by a constant value S (the stride) so that the i-th element has address $A_0+S\cdot(i-1)$. As done in [14], we classify the strides in families defined by x so that all strides $\sigma\cdot 2^x$ with σ odd belong to the same family. The length of the stream is L_1 with $L_1 = P\cdot L = 2^{\lambda+s} = 2^{\lambda_1}$ (since $\lambda \geq t$, $L_1 = k\cdot M$ for some $k > 0$).

Because the memory is organized in several sections and each section in several modules, an address mapping is required which transforms the physical address A with binary representation $a_{n-1}a_{n-2}...a_1a_0$ into a tuple (section, supermodule, displacement). The term supermodule refers to the module number within a section. In this paper the address mapping is done using the block-interleaved storage scheme shown in figure 2. A field of t bits located at position c_0 specifies the supermodule that is accessed within a section; a field of s bits located at position c_1 ($c_1 = c_0+t$) specifies the section. The rest of the bits indicate the displacement in the module. In the next section we explain how we determine the values for c_1 and c_0.

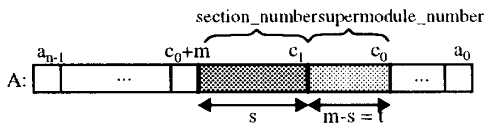
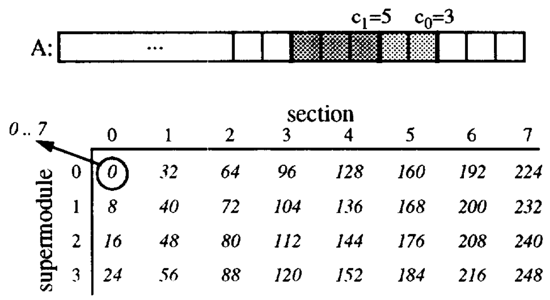


Figure 2: Block-interleaved address mapping used in this paper.

Figure 3 shows the mapping of addresses to memory modules when m=5, s=3, t=2, c_0=3 and c_1=5. It shows the mapping of the first 256 addresses. Each number i represents a block of 8 elements (elements from i to (i+7)).

We now define the spatial and temporal distributions of the elements of a stream, since they determine if the access can be done in a conflict-free way. The SPATIAL distribution of a stream in the multi-module memory is the M-tuple SD, where SD(i) is the number of stream elements in module i. The TEMPORAL distribution of a stream is the sequence of tuples $(m_1, ..., m_L)$ where m_i is formed by the P memory-module numbers accessed by the processors in the i-th request. Note that the elements of a stream can be requested in any order. The CANONICAL temporal distribution is the temporal distribution when the i-th element of the stream is accessed by processor ((i-1) mod P) and each processor request the elements in order.



supermodule	0	1	2	3	4	5	6	7
0	0	32	64	96	128	160	192	224
1	8	40	72	104	136	168	200	232
2	16	48	80	112	144	176	208	240
3	24	56	88	120	152	184	216	248

Figure 3: Example showing the mapping of the first 256 addresses when m=5, s=3, t=2, c_0=3 and c_1=5.

3. Conditions for a conflict-free access

In the synchronous model that we are considering, P requests are sent to memory in each cycle; the following conditions are required for a minimum-latency (conflict-free) access:

1.- P simultaneous requests must not collide in the interconnection network (if they do not collide, they will go to different sections and hence to different modules).

2.- Consecutive accesses to a memory module have to be separated T cycles.

To fulfil condition 1 we use the properties of the Omega interconnection network. As proven in [13], a set of input-output connections of the form

$$((a\cdot x+b) \bmod N, (c\cdot x+d) \bmod N)$$

does not collide in a NxN Omega network if $\gcd(a, N) \leq \gcd(c, N)$ for $0 \leq x < \alpha$ being $\alpha \leq N/\gcd(c,N)$ is satisfied. The values a = 1, b = 0, c odd and any d lead to a not-colliding connection pattern; this corresponds to a pattern where the output desired by input i is the output desired by input i-1 plus an odd value modulo 2^s. Such a pattern is the one we will apply.

Condition 2 is equivalent to state that P·T consecutive requests must visit P·T = M different memory modules. Since $L_1 = k\cdot M$, a necessary condition for this is that all memory modules have to contain the same number of stream elements; we say that a spatial distribution of a stream is BALANCED if it satisfies this condition. Because of this, to determine conditions for a conflict-free temporal distribution, we first determine conditions for a balanced stream and then consider access orderings so that condition 2 is fulfilled.

Note that the canonical temporal distribution for a stream with $x=c_1$ fulfils condition 1. In this case, 2^s consecutive elements of the stream do not collide in the interconnection network because they have different values in bits $a_{c_1+s}..a_{c_1}$ and the difference in value of these bits for two consecutive elements is (σ mod 2^s). On the other side, only streams with a stride of the family $x=c_0$ fulfil condition 2 when accessed with their canonical temporal distribution. So in conclusion, it is not possible to obtain a minimum latency access when a stream is accessed with its canonical temporal distribution with the proposed address mapping.

Lemma: For the address mapping considered, a stream with any initial address A_0, length $L_l=2^{\lambda l}$ and stride $S=\sigma \cdot 2^x$ is balanced if and only if $x \leq c_0$ and $\lambda_l \geq c_0 + m - x$.

Proof:

Necessary condition: to have a balanced stream it is necessary that all memory modules are visited by the stream. This requires that the addresses of the elements of the stream have different combinations in bits $c_0+m-1..c_0$. If $x > c_0$ this is not true since bits $x..c_0$ remain unchanged. If $\lambda_l < c_0 + m - x$, then there exists at least one instance of a stream that is non-balanced (e. g. a stream with initial address $A_0 = 0$ and stride $1 \cdot 2^0$ has its elements not mapped in all the memory modules).

Sufficient condition: if $x \leq c_0$ and $\lambda_l = c_0 + m - x + a$ (for some $a \geq 0$), the elements of the stream have addresses

$$A_0 + k \cdot \sigma \cdot 2^x,\ k = 0, 1, ..., 2^{\lambda l}-1$$

Since σ is odd, the values $k \cdot \sigma \cdot 2^x$ make $2^{\lambda l}$ different combinations to appear in the bits $\lambda_l+x..x$; therefore, every combination of the bits $c_0+m-1..c_0$ appears 2^{c0-x+a} times; as a consequence, the stream is balanced. ❑

From this property, making $c_0 \leq \lambda_l - m + x$ we obtain balanced streams of length $2^{\lambda l}$ for $x \leq c_0$. It is desirable to obtain conflict-free access to streams with strides of the family $x = 0$ (odd strides), since they are the most frequent in real programs; so, we will force that $c_0 \leq \lambda_l - m$. Making $c_0 = \lambda_l - m$ (and therefore $c_l = \lambda$) we obtain the largest set of families that produce balanced streams.

Up to now we have shown the conditions for a stream to be balanced. However, the access in order can lead to a high latency because of an unsuitable canonical temporal distribution. For instance, consider again the example shown in figure 3 and an access with stride S=6 (family x=1 and σ=3) and A_0=74. If the elements are accessed in order, the following succession of supermodules and sections is produced:

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
sm.:	1	2	2	3	0	1	1	2
sec.:	2	2	2	2	3	3	3	3
sm.:	3	0	0	1	2	3	3	0
sec.:	3	4	4	4	4	4	4	5
				...	...			

Observe that conflicts are produced both at the interconnection network and at the memory modules. In the next section we show how to reorder the access of the stream elements so as to achieve a better temporal distribution.

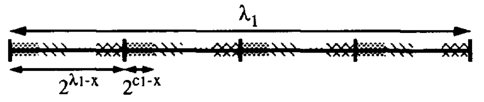
4. Access method

To avoid collisions in the interconnection network we force that, in every cycle, if processor P_i requests an element with address A, processor P_{i+1} will request an element with address $A + \sigma \cdot 2^{c1}$ for all $i=0...2^s-2$; in this way, while processor P_i is accessing section $j = \lfloor A/2^{cl} \rfloor$ mod 2^s processor P_{i+1} is accessing section $(j+\sigma)$ mod 2^s and condition 1 is fulfiled. To guarantee condition 2, $T = 2^{m-s}$ consecutive requests of each processor must be mapped into different modules, i.e., the bits $c_0+t-1..c_0$ must be different.

The mechanism to achieve these properties is as follows:

1) split the stream in 2^x subvectors of $2^{\lambda l-x}$ consecutive elements each. The first element of subvector i is $v_i = i \cdot 2^{\lambda l-x}$.

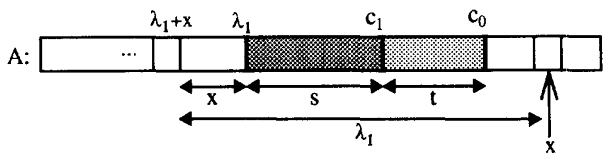
2) partition each subvector in 2^s slices of 2^{cl-x} consecutive elements each. Slices are assigned to processors as shown in the following figure.



▩ : elements assigned to processor 0
▧ : elements assigned to processor 1
⋮
▦ : elements assigned to processor 2^s-1

Each processor will access its slices sequentially. The addresses of the qth elements of any consecutive slices differ by $\sigma \cdot 2^{cl}$ for all q; hence, if all processors follow the same ordering to access the elements of its slices, there will never be collisions in the interconnection network.

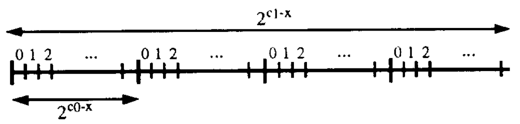
3) the elements in a slice are accessed in the following way. Elements separated by 2^{c0-x} form a sequence of T elements that are mapped into different modules



	P_0	P_1	...	P_{2^s-1}	
subvector 0:					
sequence 0:					
	0	2^{c_1-x}	...	$(2^s-1)\cdot 2^{c_1-x}$	
	2^{c_0-x}	$2^{c_0-x}+2^{c_1-x}$	...	$2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$	
	:	:		:	
	$(2^t-1)\cdot 2^{c_0-x}$	$(2^t-1)\cdot 2^{c_0-x}+2^{c_1-x}$	...	$(2^t-1)\cdot 2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$	
sequence 1:					
	1	$1+2^{c_1-x}$	...	$1+2^{c_1-x}$	
	$1+2^{c_0-x}$	$1+2^{c_0-x}+2^{c_1-x}$	...	$1+2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$	
	:	:		:	
	$1+(2^t-1)\cdot 2^{c_0-x}$	$1+(2^t-1)\cdot 2^{c_0-x}+2^{c_1-x}$	...	$1+(2^t-1)\cdot 2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$	
		:			
sequence $2^{c_0-x}-1$:					
	$2^{c_0-x}-1$	$2^{c_0-x}-1+2^{c_1-x}$	...	$2^{c_0-x}-1+(2^s-1)\cdot 2^{c_1-x}$	
	$2^{c_0-x}-1+2^{c_0-x}$	$2^{c_0-x}-1+2^{c_0-x}+2^{c_1-x}$	...	$2^{c_0-x}-1+2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$	
	:	:		:	
	$2^{c_0-x}-1+(2^t-1)\cdot 2^{c_0-x}$	$2^{c_0-x}-1+(2^t-1)\cdot 2^{c_0-x}+2^{c_1-x}$	...	$2^{c_0-x}-1+(2^t-1)\cdot 2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$ $=2^{\lambda_1-x}-1$	time
subvector 1:					
sequence 0:					
	$2^{\lambda_1-x}=v_1$	$2^{c_1-x}+v_1$	...	$(2^s-1)\cdot 2^{c_1-x}+v_1$	
	:	:		:	
	$(2^t-1)\cdot 2^{c_0-x}+v_1$	$(2^t-1)\cdot 2^{c_0-x}+2^{c_1-x}+v_1$	...	$(2^t-1)\cdot 2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}+v_1$	
		:			
subvector 2^x-1:					
sequence 0:					
	$(2^x-1)\cdot 2^{\lambda_1-x}=v_{2^x-1}$	$2^{c_1-x}+v_{2^x-1}$	...	$(2^s-1)\cdot 2^{c_1-x}+v_{2^x-1}$	
		:			
sequence $2^{c_0-x}-1$:					
	$2^{c_0-x}-1+v_{2^x-1}$	$2^{c_0-x}-1+2^{c_1-x}+v_{2^x-1}$	...	$2^{c_0-x}-1+(2^s-1)\cdot 2^{c_1-x}+v_{2^x-1}$	
	:				
	$2^{c_0-x}-1+(2^t-1)\cdot 2^{c_0-x}$ $+v_{2^x-1}$	$2^{c_0-x}-1+(2^t-1)\cdot 2^{c_0-x}$ $+2^{c_1-x}+v_{2^x-1}$	...	$2^{c_0-x}-1+(2^t-1)\cdot 2^{c_0-x}+(2^s-1)\cdot 2^{c_1-x}$ $+v_{2^x-1}=2^{\lambda_1}-1$	

Figure 4: Accessing method.

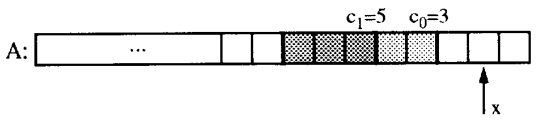
because their addresses differ by $\sigma \cdot 2^{c0}$. They are requested consecutively, so conflicts are avoided at the memory modules. The following graphic shows the sequences and elements that belong to each sequence in one slice:



There are 2^{c0-x} sequences and they are accessed sequentially.

Figure 4 shows the elements of some sequences.

Next we consider again the example of figure 3. If we suppose $\lambda = 5$, then $c_1 = 5$ and $c_0 = 3$ is the best choice, as explained in the previous section. A conflict-free access is possible for streams with any initial address, length = 256 ($\lambda_1 = 8$) and stride $S = \sigma \cdot 2^x$ with x = 0, 1, 2 or 3.



Let x = 1. The stream is splitted in two subvectors (elements 0 to 127 and 128 to 255). Each subvector is splitted again into 8 slices of 16 consecutive elements each. Within each slice there are 4 sequences of 4 elements each. The sequences and the access ordering are the following:

P_0	P_1	P_2	P_3
0,4,8,12	16,20,24,28	32,36,40,44	48,52,56,60
1,5,9,13	17,21,25,29	33,37,41,45	49,53,57,61
2,6,10,14	18,22,26,30	34,38,42,46	50,54,58,62
3,7,11,15	19,23,27,31	35,39,43,47	51,55,59,63
128,132,136,140	...	...	...

P_4	P_5	P_6	P_7
64,68,72,76	80,84,88,92	96,100,104,108	112,116,120,124
65,69,73,77	81,85,89,93	97,101,105,109	113,117,121,125
66,70,74,78	82,86,90,94	98,102,106,110	114,118,122,126
67,71,75,79	83,87,91,95	99,103,107,111	115,119,123,127
192,196,200,204	...	...	...

A stream with initial address $A_0 = 74$ and $\sigma = 3$, for instance, causes the following succession of supermodules and sections:

	P_0	P_1	P_2	P_3
sm.:	1,0,3,2	1,0,3,2	1,0,3,2	1,0,3,2
sec.:	2,3,3,4	5,6,6,7	0,1,1,2	3,4,4,5
sm.:	2,1,0,3	2,1,0,3	2,1,0,3	2,1,0,3
sec.:	2,3,4,4	5,6,7,7	0,1,2,2	3,4,5,5
	...	...	...	...

	P_4	P_5	P_6	P_7
sm.:	1,0,3,2	1,0,3,2	1,0,3,2	1,0,3,2
sec.:	6,7,7,0	1,2,2,3	4,5,5,6	7,0,0,1
sm.:	2,1,0,3	2,1,0,3	2,1,0,3	2,1,0,3
sec.:	6,7,0,0	1,2,3,3	4,5,6,6	7,0,1,1
	...	...	...	...

The control to perform the requests in the order proposed in each processor P_i is as follows:

```
    A_seq=A=A_1 + i·σ·2^c1          ;initial address
    for K = 1 to 2^x                ;subvectors
    for J = 1 to 2^(c0-x)           ;sequences
        for I = 2 to 2^t            ;access a se-
quence
            A = A + σ·2^c0
        end for
        if J<2^(c0-x) then          ;next sequence
            A_seq=A=A_seq + σ·2^x
        end if
    end for
    A_seq=A=A_seq + σ·(2^λ1-2^c0) ;next subvector
    end for
```

Figure 5 shows the hardware required to generate the addresses as explained in this section.

To simplify the implementation we have considered the compiler generates instructions to load the values $A_0+i \cdot \sigma \cdot 2^{c1}$ (initial address of the first slice accessed by processor i), and $\sigma \cdot 2^x$, $\sigma \cdot 2^{c0}$, $\sigma \cdot (2^{\lambda 1}-2^{c0})$ and 2^{c0-x} as well.

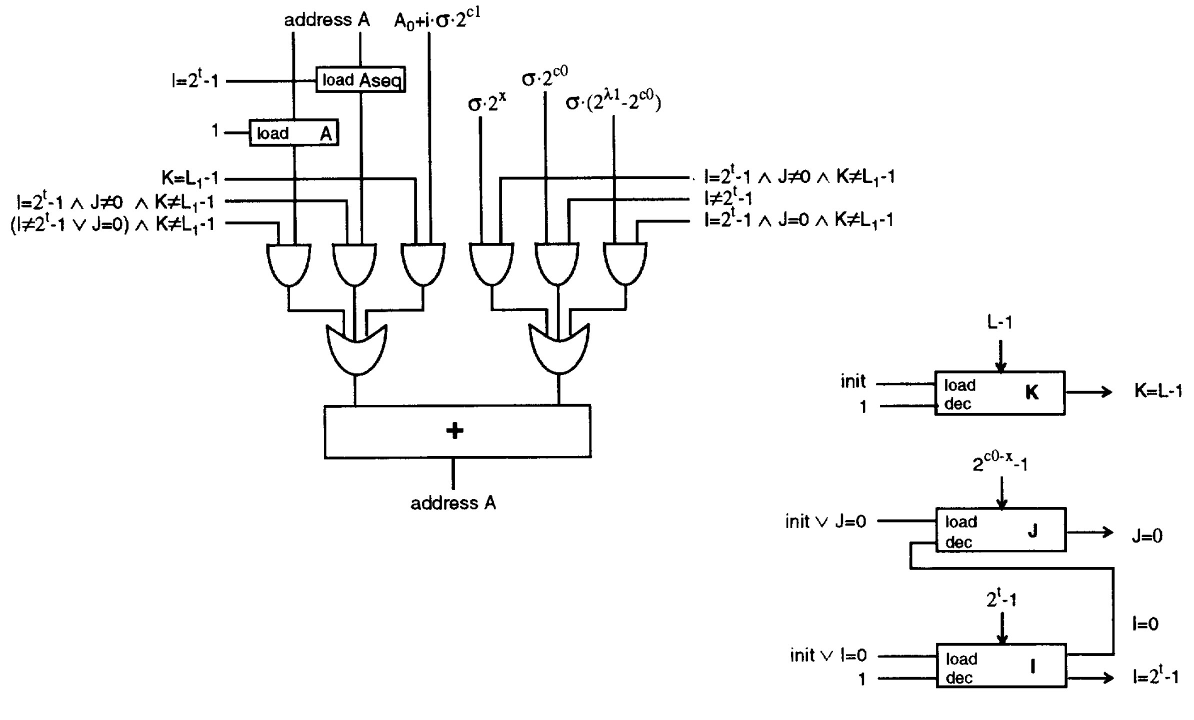
Notice that in the special case of $L_1 = M$, each processor accesses only one slice of T consecutive elements of the stream, instead of accessing several discontinuous slices. The address mapping proposed leads to $c_0 = 0$, and therefore conflict-free access is obtained just for vectors with stride belonging to the family x = 0. There is only one sequence in each slice, containing the following elements:

P_0	0	1	...	$2^{c1}-1$
P_1	$2^{c1-x} = 2^{c1}$	$1 + 2^{c1}$	...	$2^{c1}-1 + 2^{c1}$
:	:	:		:
P_{2^s-1}	$(2^s-1) \cdot 2^{c1}$	$1 + (2^s-1) \cdot 2^{c1}$	...	$2^{c1}-1 + (2^s-1) \cdot 2^{c1}$

However, the same result can be obtained for any x > 0 forcing $c_0 = x$. The idea of using a dynamic storage scheme presented in [14] for the uniprocessor system case could be applied to obtain a conflict-free access to streams with a stride belonging to any single family.

5. Additional reordering

The previous example shows that although one sequence can be accesed in a conflict-free manner, there may be conflicts between different sequences. For instance, in the 5th cycle P_1 requests an element located in the same



address A
$A_0+i\cdot\sigma\cdot 2^{c1}$
$I=2^t-1$
load Aseq
1
load A
$K=L_1-1$
$I=2^t-1 \wedge J\neq 0 \wedge K\neq L_1-1$
$(I\neq 2^t-1 \vee J=0) \wedge K\neq L_1-1$
$\sigma\cdot 2^x$
$\sigma\cdot 2^{c0}$
$\sigma\cdot(2^{\lambda 1}-2^{c0})$
$I=2^t-1 \wedge J\neq 0 \wedge K\neq L_1-1$
$I\neq 2^t-1$
$I=2^t-1 \wedge J=0 \wedge K\neq L_1-1$
+
address A
L-1
init
1
load
dec
K
K=L-1
$2^{c0-x}-1$
init ∨ J=0
load
dec
J
J=0
2^t-1
I=0
init ∨ I=0
1
load
dec
I
$I=2^t-1$

Figure 5: Hardware for address calculation

module as the one requested by P_3 in the 4th cycle, which is still occupied.

For a global conflict-free access, the supermodule visited by sequence k in its qth cycle must be the same as the one visited by sequence k+1 in its qth cycle; by now, our method does not guarantee the fulfillment of this condition. In the example considered above, for instance, conflicts would be avoided if the elements of each sequence were sent in the supermodule order (1,0,3,2), defined by the first sequence.

To achieve this, we propose to decouple the calculation of the addresses from the actual requests. This is achieved by precalculating the addresses of sequence i+1 while accessing sequence i. Consequently, during the first 2^t cycles, it is necessary to calculate the addresses of the first sequence (which are used immediately for memory access) and of the second sequence (which are stored in a set of latches for access as the next sequence). After that, for each sequence, the addresses for access are obtained from the latches and the addresses of the next sequence are precalculated 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 order of supermodules accessed by the first sequence, which is used to control the order of the requests of the following sequences.

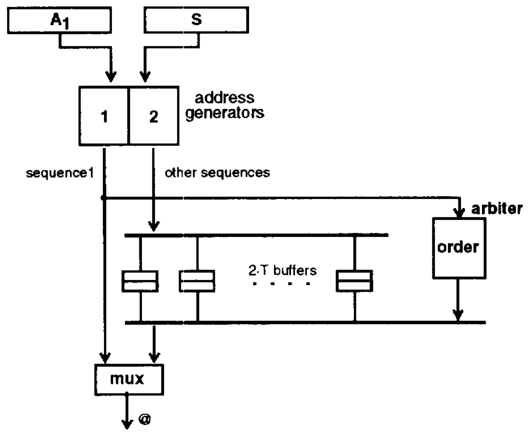


Figure 6: Architecture model for out-of-order memory accesses.

6. Conclusions

When accessing streams in vector multiprocessor machines, degradation in the interconnection network and conflicts in the memory modules are the factors that reduce the efficiency of the system. In this paper we have presented a synchronous access mechanism that allows conflict-free access to streams in a SIMD system.The work is a continuation of the method presented in [11], where an out-of-order access to the elements of a stream in a bus-based vector uniprocessor was proposed.The basic idea was to request stream elements in such an order that memory conflicts are avoided.

Each processor accesses the corresponding elements out-of-order in such a way that in each cycle the requested elements do not collide in the interconnection network. Moreover, memory modules are accessed in a way that avoids conflicts. The same problem was tackled in [10] although the solution in this case needed to precalculate the addresses before starting the access; our solution to the problem avoids this precalculation so the access is performed in a conflict-free way without any additional latency. On the contrary, we need two address generators and additional control.

The use of the proposed mechanism in present architectures would allow conflict-free access to streams with the most common strides that appear in real applications. The additional hardware has been described and shown to be of similar complexity as that required for access in order.

We plan to extend this work so that each processor access a slice of L consecutive elements of the stream. With the aim of increasing the number of conflict-free families we also plan to extend this work to the unmatched-memory case.

7. Acknowledgements

This work has been supported by the ESPRIT Basic Research Action 6634 APPARC, the Ministry of Education of Spain under contract TIC-880/92, 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, Dec. 1971.
2. J. Frailong, W. Jalby and J. Lenfant: "XOR-schemes: A Flexible Data Organization in Parallel Memories", Int. Conf. on Parallel Processing, pp. 276-283, 1985.
3. M. Dubois et al.: "Memory Access Buffering in Multiprocessors", Int. Symp. on Computer Architecture, pp. 422-434, 1988.
4. K. A. Robbins and S. Robbins: "Bus Conflicts for Logical Memory Banks on a Cray Y-MP type Processor System", Int. Conf. on Parallel Processing, pp. I.21-I.24, 1991.
5. 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.
6. D. H. Bailey: "Vector Computer Memory Bank Contention", IEEE Trans. on Computers, vol. C-36, no. 3, pp. 293-298, March 1987.
7. I. Y. Bucher and D. A. Calahan: "Models of Access Delays in Multiprocessor Memories", IEEE Trans. on Parallel and Distributed Systems, vol. 3, no. 3, May 1992.
8. T. Cheung and J. E. Smith: "A Simulation Study of the Cray X-MP Memory System", IEEE Trans. on Computers, vol. C-35, no. 7, pp. 613-622, July 1986.
9. J.E. Smith and W.R. Taylor, "Characterizing Memory Performance in Vector Multiprocessors", Int. Conf. on Supercomputing, pp. 35-44, 1992.
10. A. Seznec and J. Lenfant: "Interleaved Parallel Schemes: Improving Memory Throughput on Supercomputers", Int. Symp. on Computer Architecture, pp. 246-255, 1992.
11. M. Valero, T. Lang, J.M. Llaberia, M. Peiron, E. Ayguadé and J.J. Navarro: "Increasing the Number of Strides for Conflict-Free Vector Access", Int. Symp. on Computer Architecture, pp. 372-381, 1992.
12. M. Valero, T. Lang and E. Ayguadé: "Conflict-Free Access of Vectors with Power-of-Two Strides", Int. Conf. on Supercomputing, 1992. To be published.
13. 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.
14. D.T. Harper III and D. A. Linebarger: "Conflict-Free Vector Access Using a Dynamic Storage Scheme", IEEE Trans. on Computers, vol. C-40, no. 3, pp. 276-283, March 1991.