LOGICAL DATA SKEWING SCHEMES FOR INTERLEAVED MEMORIES IN VECTOR PROCESSORS

by

G. S. Sohi

Computer Sciences Technical Report #753

February 1988

LOGICAL DATA SKEWING SCHEMES FOR INTERLEAVED MEMORIES IN VECTOR PROCESSORS

G. S. Sohi

Computer Sciences Department
University of Wisconsin-Madison
1210 W. Dayton Street
Madison, WI 53706
sohi@cs.wisc.edu
(608)-262-7985

Abstract

Sustained memory bandwidth for a range of strides is a key to high-performance vector processing. To reduce the number of memory bank conflicts and thereby increase memory bandwidth, one often resorts to data skewing schemes. In this paper, we present a family of data skewing schemes called logical skewing schemes. In logical skewing schemes, the location of a data element is determined solely by logical manipulation of the address bits. Our experiments show that logical skewing schemes allow for better vector memory system performance than other known data skewing schemes without the significant increase in memory latency that is traditionally associated with data skewing schemes.

1. Introduction

In order to maintain the balance in a computer system, the memory system must be able to sustain the bandwidth requirements of the CPU [1]. In high-performance vector processing computers, to provide high-bandwidth access to large amounts of data, *parallel* or *interleaved* memories must be used.

An interleaved memory consist of several memory *modules* or *banks*. By accessing elements in distinct banks simultaneously, the available bandwidth of the memory system can be improved considerably. However, the actual bandwidth of the memory system may be less than the available bandwidth because of *collisions* or *conflicts* in the banks. A conflict occurs when the request is to to a memory bank that is already busy servicing a previous request.

Interleaved memories have been studied in detail by several researchers [2-9]. In[2] Budnick and Kuck showed that a memory system that allows conflict-free access for vectors of *all possible* strides[1] is not possible. Therefore, a memory system must attempt to improve the average throughput or to allow conflict-free access to a particular set of strides rather than achieve the impossible goal of perfect performance for all strides of access. Most of the previous work has focussed on conflicts in memory systems for a single access stream though recent work has also investigated conflicts in multi-port memory systems that allow multiple access streams simultaneously [7, 8].

In this paper, we attempt to improve the overall single-access-stream memory system performance for vectors of arbitrary strides. In section 2 we discuss the basic concepts in the design of interleaved memory systems for vector processors and the previous work in this area. To improve memory system performance, we use logical skewing schemes to distribute components of a vector amongst the memory banks in a non-regular fashion. Section 3 introduces the logical skewing schemes that we use and discuss them in some detail. In section 4, we see how the concept of logical skewing can be extended to design a memory system that can be programmed to allow conflict-free to a particular (though not all!) stride. In section 5, we discuss the performance of a memory system that uses a

[1] The *stride* of a vector is the difference in linear memory addresses between successive elements of the vector.

simple logical skewing scheme and compare it to the performance of other memory system designs. Finally, section 6 presents some concluding remarks.

2. Interleaved Memories For Vector Processors

In the vector processing paradigm, computation is carried out on an entire chunk or a vector of data. In order to carry out the computation, elements of the vector must be accessed from the memory, preferably with a throughput of one vector element per cycle for a each stream of access[2].

If the vector machine has memory-memory vector instructions (such as the Cyber 205), the vector computation instruction itself is responsible for fetching the data from the interleaved memory. If the vector machine has register-register vector instructions (such as the CRAY-1), the vector computation instruction must be preceded by a vector load instruction. The vector load instruction transfers the appropriate data elements from the interleaved memory into a vector register. In either case, to sustain vector performance for a variety applications, it is crucial that the memory system must have adequate throughput for vector accesses of arbitrary strides.

As mentioned earlier, an interleaved memory system which consists of several independent banks of memory is a cost-effective way of providing adequate memory throughput. Let us suppose that the number of banks in the memory is M. In most interleaved memories, M is a power of 2. However, M need not be restricted to a power of 2. Indeed, memory systems that have a *prime* number of banks have been studied[5] and built [10]. Unfortunately, in a memory system with an arbitrary value of M, the process of determining the location of a desired data element is quite complicated [5]. By restricting M to be a power of 2, this process is simplified greatly. Therefore, we shall restrict our discussion to memory systems with a number of banks that is a power of 2.

The simplest and most widely used interleaving scheme, i.e., a *low-order* interleaving scheme, uses the low-order $n=\log_2 M$ bits of an N-bit address to select the bank and the remaining $N-n$ bits to select the word within the bank. We shall also refer to this as a *standard interleaving* scheme. Using

[2] In this paper, we shall restrict ourselves to a single stream of accesses to the memory system. The ideas presented in this paper could easily be applied to the design of a multi-port memory system that handles multiple streams of access.

this scheme, the bank number in which an arbitrary address A is located is specified by A *mod* M and the word with the bank for the address A is A *div* M. If a standard interleaving scheme is used, consecutive elements in the linear memory address space are placed in distinct banks. A distribution of the elements of a 64-element vector stored contiguously in the linear address space in 8 banks using a standard low-order interleaving scheme is shown in Figure 1.

Other interleaving schemes that use an arbitrary, but known, n bits of the address to select the bank and the remaining $N-n$ bits of address to select the word within the bank are also possible. For example, a *high-order* interleaving scheme would use the high-order n bits of the address to select the bank and the low-order $N-n$ bits of the address to select the word with the selected bank. However, they are not as popular as a standard interleaving scheme because they do not allow for conflict-free access to a vector of stride 1.

In a memory system with a standard interleaving scheme the components of a vector with stride 1 fall in different banks. However, the components of a vector with stride M all fall in the same bank and result in conflicts. Conflict situations can also arise for other strides depending upon the number of banks and the relative bank cycle time.

Word	Bank							
	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47
6	48	49	50	51	52	53	54	55
7	56	57	58	59	60	61	62	63

Figure 1: Distribution of Elements Using Low-order Interleaving

One could reduce the number of conflicts by using *array reshaping* techniques. Array reshaping involves the embedding of an array in a larger array in an attempt to make the "stride" of access relatively prime to the number of banks. Memory conflicts for common reference patterns are reduced at the expense of wasted memory. In order for a compiler to carry out array reshaping, it must have *a priori* knowledge of the access patterns to the array structure. Array reshaping techniques, however, have their drawbacks. In this paper, we shall not discuss array reshaping techniques any further; the interested reader is referred to[9] for some discussion on such techniques.

Since the problem of accessing elements of an arbitrary-stride vector from a set of interleaved memory banks is somewhat similar to the problem of accessing a superword from a set of parallel memory banks one could use *data skewing* techniques to reduce the possibility of conflicts. Indeed, several data skewing schemes to reduce conflicts have also been proposed and studied in the literature [3, 5, 9].

Traditional data skewing schemes discussed in the literature have two distinct characteristics: (i) they distribute the data across the banks in an "orderly" fashion, and (ii) they require arithmetic operations to determine the location of each addressed word (the location of an addressed word is specified by the bank number and the position of the word within the bank). For example, Harper and Jump use a *1-skew* scheme in which the bank number is "shifted over by 1" for each word in the bank [9]. In the 1-skew scheme, the bank number (M_i) for an arbitrary address i is calculated as:

$$M_i = \left\lceil i + \left\lfloor \frac{i}{M} \right\rfloor \right\rceil \bmod M$$

The distribution of the components of a 64-element vector in 8 banks using a 1-skew storage scheme is given in Figure 2. Note that elements of the subvector that comprises word k in the memory banks are distributed in a regular fashion. The only difference between word 0 and word k is that the first element of word k is placed in bank k. Also note that arithmetic operations (an addition) have to be carried out to determine the bank number. Such arithmetic operations can degrade the memory latency considerably.

Word	Bank							
	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	1	2	3	4	5	6	7
1	15	8	9	10	11	12	13	14
2	22	23	16	17	18	19	20	21
3	29	30	31	24	25	26	27	28
4	36	37	38	39	32	33	34	35
5	43	44	45	46	47	40	41	42
6	50	51	52	53	54	55	48	49
7	57	58	59	60	61	62	63	56

Figure 2: Distribution of Elements Using 1-Skew Storage

More recently, researchers have investigated the use of *logical* or *Boolean* skewing schemes [11, 12]. In[12] Norton and Melton investigate the use of Boolean schemes to allow conflict-free access to a power-of-two stride superword in a parallel processor. The attractiveness of logical skewing schemes lies in the fact that they use only logical manipulations on the address bits to locate the desired word in the memory system.

We believe that in order to decrease the probability of conflicts for vectors of arbitrary strides, a data skewing scheme should distribute the data across the banks in a non-regular fashion. We also believe that the computation of the bank number should not involve arithmetic manipulations that involve the rippling of signals (for example a carry signal). Computation of the bank number should rely solely on the logical manipulation of the address bits. To achieve these goals, we shall use a family of data skewing schemes which we refer to as *logical skewing schemes* in this paper.

3. Logical Skewing Schemes

Recall that in order to locate a desired word in an interleaved memory system, we need to specify: (i) the bank number and (ii) the address or position within the selected bank. In a memory system with $M = 2^n$ banks and 2^{N-n} words in each bank, we need n bits to specify the bank number

and $N-n$ bits to specify the address within each bank.

In the logical skewing schemes that we use in this paper, the position of the word within each bank is determined as in a standard interleaved memory, i.e., by using the high-order $N-n$ bits of address. However, any $N-n$ bits of the N-bit address could be used to determine the address within a bank; the remaining n bits of address, suitably modified as discussed below, would be used to determine the bank number.

The elegance of logical skewing schemes lies in the determination of the bank number. Rather than using only n bits of address to determine the bank number, we can potentially use all N bits of address to determine the bank as follows. If $\bar{X}$ is the N-bit address of the word and $\bar{Y}$ is the n-bit vector that represents the bank number, then $\bar{Y}$ is calculated as:

$$\bar{Y} = \mathbf{A}\bar{X} \tag{1}$$

where $\mathbf{A}$ is an $n \times N$ matrix of 0's and 1's. The inner product is a logical inner product with the "multiplication" being a logical AND operation and the "addition" being a logical EXCLUSIVE-OR operation. Element Y_i of $\bar{Y}$ is, therefore:

$$Y_i = (A_{i,0}.X_0) \oplus (A_{i,1}.X_1) \oplus \ldots . \oplus (A_{i,N-1}.X_{N-1}) \tag{2}$$

where X_j is the jth bit of the address.

In order to clarify the scheme, consider again the example of distributing the elements of a 64-element vector in 8 banks. If the bank number $Y_2Y_1Y_0$ is given by:

$$\begin{bmatrix} Y_2 \\ Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_5 \\ X_4 \\ X_3 \\ X_2 \\ X_1 \\ X_0 \end{bmatrix} \tag{3}$$

i.e., $Y_0 = X_4 \oplus X_3 \oplus X_0$, $Y_1 = X_4 \oplus X_3 \oplus X_1$, and $Y_2 = X_4 \oplus X_3 \oplus X_2 \oplus X_1$, the distribution of the elements of a 64-element vector in 8 banks is given in Figure 3.

If, on the other hand, the $\mathbf{A}$ matrix is modified so that the bank number $Y_2Y_1Y_0$ is given by:

Word	Bank							
	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	1	6	7	4	5	2	3
1	11	10	13	12	15	14	9	8
2	19	18	21	20	23	22	17	16
3	24	25	30	31	28	29	26	27
4	32	33	38	39	36	37	34	35
5	43	42	45	44	47	46	41	40
6	51	50	53	52	55	54	49	48
7	56	57	62	63	60	61	58	59

Figure 3: Distribution of Elements Using the Logical Skewing Scheme of Equation (3)

$$\begin{bmatrix} Y_2 \\ Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_5 \\ X_4 \\ X_3 \\ X_2 \\ X_1 \\ X_0 \end{bmatrix} \tag{4}$$

i.e., $Y_0 = X_4 \oplus X_3 \oplus X_2 \oplus X_1 \oplus X_0$, $Y_1 = X_5 \oplus X_4 \oplus X_2 \oplus X_1$, and $Y_2 = X_5 \oplus X_3 \oplus X_2$, the distribution of the elements of a 64-element vector in 8 banks is given in Figure 4.

By modifying the contents of the **A** matrix, we can obtain logical skewing schemes or "hash functions" that distribute the elements of a linear vector across the memory banks in a non-regular fashion[3]. That is, the distribution of the elements of word i have no clear relationship to the distribution of elements of word j. To specify the logical skewing scheme, all that we have to do is to specify the contents of the **A** matrix.

Before proceeding further, let us discuss some issues the design and the properties of the **A** matrix. However, since the main thrust of the paper is to present the idea of logical skewing schemes for interleaved memories in vector processors and to evaluate their performance potential, we shall

[3]We should mention that such "hash functions" have been used before in a different context - in the set selection process of some cache memories to enhance the ability of the cache memory to exploit temporal locality. An example of such an application for TLBs can be found in [13].

Word	Bank							
	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	1	3	2	6	7	5	4
1	15	14	12	13	9	8	10	11
2	18	19	17	16	20	21	23	22
3	29	28	30	31	27	26	24	25
4	37	36	38	39	35	34	32	33
5	42	43	41	40	44	45	47	46
6	55	54	52	53	49	48	50	51
7	56	57	59	58	62	63	61	60

Figure 4: Distribution of Elements Using the Logical Skewing Scheme of Equation (4)

keep theoretical discussions to a bare minimum.

We can rewrite equation (1) as:

$$\bar{Y} = \left[\mathbf{A}_H \mid \mathbf{A}_L \right] \begin{bmatrix} \bar{X}_H \\ - \\ \bar{X}_L \end{bmatrix}$$

where $\bar{X}_H$ is the high-order $N-n$ bits of the address and $\bar{X}_L$ is the low-order n bits of the address. The reader should note that the standard low-order interleaving scheme is a special case of our more general logical skewing schemes. For a standard interleaving scheme, $\mathbf{A}_H$ is the zero matrix and $\mathbf{A}_L$ is the unity matrix.

Using a general logical skewing scheme, 2^n elements of the linear memory address space map into the same word within a bank. Our logical skewing scheme must make sure that these 2^n elements all fall into distinct banks, i.e., no 2 distinct elements from the linear address space map into the same word within the same bank in the interleaved memory system. Mathematically, we can say that the logical skewing scheme must guarantee a unique mapping from each set of 2^n integral addresses $2^n.\beta+0$, $2^n.\beta+1$, ..., $2^n.\beta+2^n-1$, where β is an arbitrary integer that represents the word in the bank, to the 2^n banks numbers 0, 1, ..., 2^n-1. It can easily be shown that this condition is satisfied if and only if

the matrix $\mathbf{A}_L$ is of rank n. Therefore, any design of the **A** matrix must make sure that the submatrix $\mathbf{A}_L$ is of full rank. Note that if an arbitrary $N-n$ bits of the address were chosen to determine the word within the bank, this condition would be modified to state that the submatrix corresponding the n address bits not chosen to determine the word number must be of full rank.

3.1. Bank Number Calculation in Logical Skewing Schemes

In this section, we present the hardware needed to determine the position of an arbitrary word in the interleaved memory system. Recall that our skewing scheme uses the high-order $N-n$ bits of address to determine the word in each bank. These bits are passed directly to the decoding logic within each bank and no additional hardware (as compared to a standard interleaved memory) is needed. The additional hardware needed for a memory system with a logical skewing scheme is, therefore, the hardware needed to compute the bank number.

From equation (2) we see that each bit of $\bar{Y}$, i.e., the n-bit vector that represents the encoded bank number, is determined by computing an appropriate parity of the input address bits. The bits that are used in calculating parity bit Y_i are indicated by row i of the **A** matrix. The n-bit vector $\bar{Y}$ then needs to be decoded to generate the appropriate bank select signals. The overall organization of the hardware needed to carry out this task is shown in Figure 5.

In Figure 5, the N address bits are fed into n parity computation circuits. Each parity circuit calculates the parity of a select number of input bits as determined by the **A** matrix. These parity bits represent the n-bit bank number that contains the addresses word. These parity bits are then input to a decoder which is responsible for generating the 2^n bank select signals.

In a standard interleaved memory, n bits of the address are directly fed to the decoder. The overhead of our logical skewing scheme is, therefore, the delay through the parity computation circuits. For a simple skewing scheme such as the one described by equation (3), this overhead can be as little as the delay through a single EXOR gate.

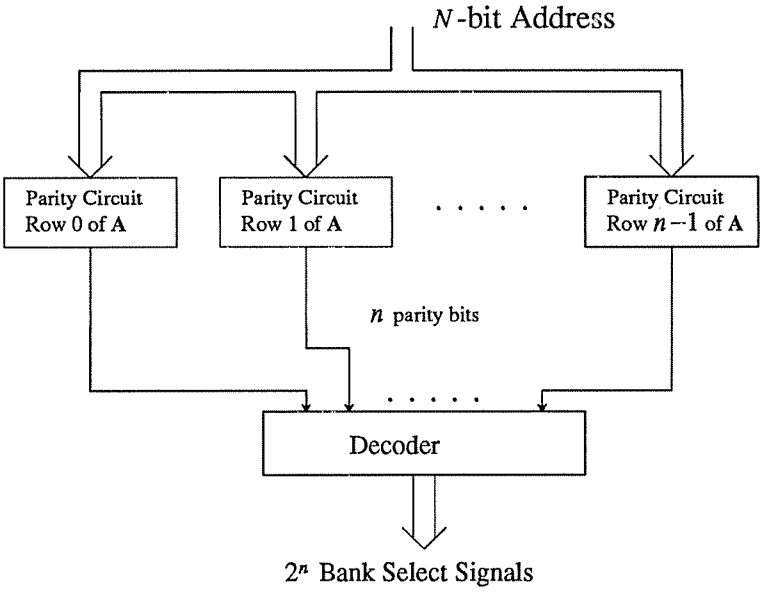

Figure 5: Hardware Organization for Determining the Bank Number in a Logical Skewing Scheme

For more complicated schemes that make use of an arbitrary number (a maximum of N) bits of address, the N-bit parity circuit can be implemented with $\log_k N$ levels of EXOR gates where k is the fan-in of each EXOR gate. For k=4, the parity of up to 64 input bits can be computed with only 3 levels of EXOR logic. Therefore, we do not expect the degradation in memory latency to be a very significant factor for logical skewing schemes.

The reader should note that in some cases, it may be possible to merge part or all of the EXOR logic with the decoding logic. In other cases, it may be possible to incorporate some of the EXOR logic within the address generation hardware itself (for example in the address calculation adder) without increasing the length of the critical path. In such cases, the logical skewing scheme has *no additional overhead* over a standard interleaved memory. In any case, the overhead for an arbitrary skewing scheme is not more than a few levels of logic and is far less than the overhead for skewing

schemes that involve arithmetic calculations (a carry ripple) in the computation of the bank number.

4. Programmable Interleaved Memories

In the hardware of Figure 5, the parity circuits were designed specifically for a particular **A** matrix, i.e., for a particular skewing scheme. In some situations, it may be useful to alter the skewing scheme under program control depending upon the access patterns of the particular program to be run.

Consider for example a matrix multiplication program that multiplies 2 matrices **X** and **Y**. Suppose that the dimensions of **X** and **Y** are are $P \times Q$ and $Q \times R$, respectively. Also suppose that the program has been written assuming that the matrices are stored in row-major order in the linear address space. The skewing scheme is responsible for distributing the elements of the matrices amongst the banks.

The normal matrix multiplication algorithm access the elements of each row of **X** and each column of **Y**, i.e., makes stride 1 accesses to matrix **X** and stride Q accesses to matrix **Y**. If the logical skewing scheme were hardwired, then we could not be sure if conflict-free access was available to the matrices for an arbitrary Q. However, if we could alter the skewing scheme under program control, we could attempt to achieve conflict-free access to the access patterns made by the program. The program can be written assuming that the array is stored in row- or column-major format, but the logical skewing scheme can be varied to distribute the elements amongst the memory banks under program control depending upon the value of Q.

Fortunately, extending the bank-selection hardware of Figure 5 to allow for arbitrarily programmable schemes is quite straight forward and does not have much additional hardware overhead. Recall that the skewing scheme is specified completely by specifying the contents of the **A** matrix. Therefore, all that we have to do to modify the skewing scheme is to modify the contents of the **A** matrix, i.e., modify the operation of the parity computation circuits. This modification is easily implemented if we design each parity computation circuit to compute the parity of all N input bits but modify the input bits to the parity circuit by ANDing the address bits with a mask. The mask is simply a row of the **A** matrix and can be altered under program control. The modified hardware for each bit

of the encoded bank number is shown in Figure 6.

For this programmable interleaved memory system, the latency of each memory request is increased by the time it takes to pass through the AND circuit and the N-bit parity logic. The AND circuit is simply a linear array of AND gates, one for each bit, and therefore the maximum delay through this circuit is the delay through a single AND gate. The delay through the N-bit parity computation circuit is the delay through $\log_k N$ levels of EXOR logic, where k is the fan-in of each EXOR gate. Therefore, the overhead for having a programmable memory system is not very significant when compared to a memory system with a data skewing scheme that involves arithmetic operations.

5. A Simulation Analysis

In order to evaluate the performance of a vector memory system with a logical skewing scheme, we decided to carry out a simulation analysis. Our model of the memory system is the same as the one

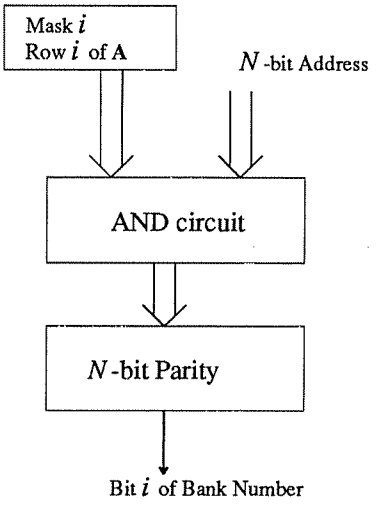

Figure 6: Bank Selection in a Programmable Interleaved Memory

proposed by Harper and Jump[9] and is shown in Figure 7.

In the memory system, an address is generated by the address source at a maximum rate of one address per clock cycle. The addresses are transmitted to the selected bank for service. If a bank is busy, the address is queued in the input buffer of the bank. The bank services requests from its input buffer and places the results in its output buffer. From the output buffer, the requests are returned to the data sink in the same order as the requests generated by the address source. The address source stops generating requests when the input buffer of a desired bank is full and resumes generating requests when the buffer is available. The buffers are useful in smoothing out transient effects.

Since we are mainly interested in seeing *how* a logical skewing scheme performs in comparison to other interleaving schemes, we shall not attempt to evaluate the effect of the orthogonal issues of the number of banks and the relative cycle time of each memory bank. Rather, we use a memory system that has 8 banks and each bank has a cycle time of 4 clocks. We shall also not attempt to analyze various logical skewing schemes in this paper. For our experiment, we use a logical skewing scheme in

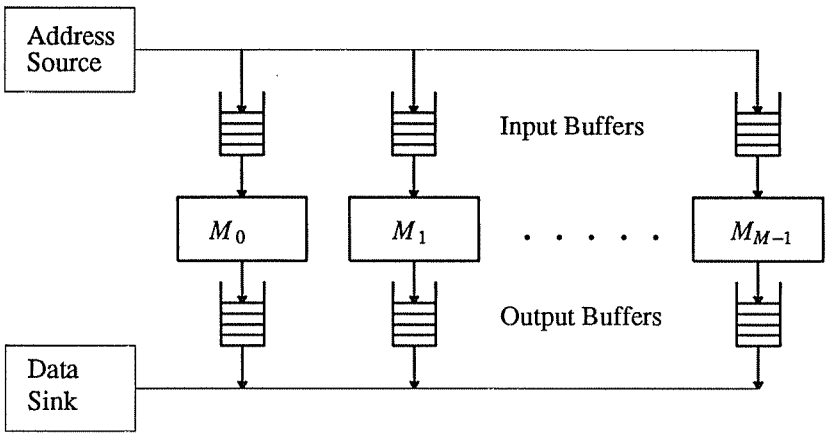

Figure 7: An Interleaved Memory System With Input and Output Buffers

which the bank number is determined as in equation (3), i.e., $Y_0 = X_4 \oplus X_3 \oplus X_0$, $Y_1 = X_4 \oplus X_3 \oplus X_1$, and $Y_2 = X_4 \oplus X_3 \oplus X_2 \oplus X_1$. Our choice of the logical skewing scheme was quite arbitrary; the main criterion was that vectors of stride 1 should have conflict free access even with no input/output buffers. (Indeed, several other logical skewing schemes that we considered had similar performance with the use of buffers).

For the memory system parameters described above, we evaluated the throughput for 3 different interleaving schemes: (i) the standard low-order scheme, (ii) the 1-skew scheme described in [9], and (iii) the logical skewing scheme described above. The stride of the input vectors was varied from 1 to 32. Note that since the standard interleaving scheme uses only the low-order 3 bits of address to select the bank, the throughput of a vector with stride $\alpha 1$ is the same as the throughput of a vector with stride $\alpha 2$ where $\alpha 2 = \alpha 1 \ mod \ 8$, i.e., the pattern of throughput repeats after a stride of 8. Likewise, the pattern of throughput repeats after a stride of 32 for our logical skewing scheme (since it uses 5 bits of the address) and after 64 for the 1-skew scheme (since it uses 6 bits of the address).

The throughput is calculated as the number of elements accessed per clock cycle, i.e., the total number of elements accessed divided by the total number of clock cycles taken. The vectors are long so that startup and flushing effects are ignored. We also assume that the number of entries in each input buffer is the same as the number of entries in each output buffer. The results of our simulations are presented in Table 1. The table presents the throughput versus the stride for the 3 interleaved memory systems with the number of entries in the input and output buffers ranging from 0 to 3. Since the throughput pattern repeats itself after stride 32 for our logical skewing scheme, we have restricted the maximum stride to 32. Note, however, that the throughput pattern for the 1-skew scheme does not repeat until after stride 64. In Table 1, we have also highlighted the cases for which a perfect throughput (ignoring startup) cannot be achieved.

5.1. Discussion of Results

For the standard low-order interleaving scheme, any stride that is a multiple of 8 has a throughput of 0.25 elements per cycle and any stride that is a multiple of 4 (but not of 8) has a

Table 1: Throughput vs. Stride For The Example Memory System

Stride	Interleaving Scheme Used											
	Low-Order				1-Skew				Logical Skew			
	Input/Output Buffer Size				Input/Output Buffer Size				Input/Output Buffer Size			
	0	1	2	3	0	1	2	3	0	1	2	3
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.89**	1.00	1.00	1.00
4	**0.50**	**0.50**	**0.50**	**0.50**	1.00	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	**0.89**	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	1.00	**0.28**	**0.29**	**0.29**	**0.29**	**0.73**	1.00	1.00	1.00
8	**0.25**	**0.25**	**0.25**	**0.25**	1.00	1.00	1.00	1.00	**0.40**	**0.50**	**0.50**	**0.50**
9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.73**	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
11	1.00	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00	**0.73**	1.00	1.00	1.00
12	**0.50**	**0.50**	**0.50**	**0.50**	**0.80**	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00
13	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.57**	1.00	1.00	1.00
14	1.00	1.00	1.00	1.00	**0.31**	**0.33**	**0.34**	**0.34**	1.00	1.00	1.00	1.00
15	1.00	1.00	1.00	1.00	**0.73**	1.00	1.00	1.00	**0.57**	1.00	1.00	1.00
16	**0.25**	**0.25**	**0.25**	**0.25**	1.00	1.00	1.00	1.00	**0.50**	**0.50**	**0.50**	**0.50**
17	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.57**	1.00	1.00	1.00
18	1.00	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00	1.00	1.00	1.00	1.00
19	1.00	1.00	1.00	1.00	**0.76**	**0.80**	**0.80**	**0.80**	**0.57**	1.00	1.00	1.00
20	**0.50**	**0.50**	**0.50**	**0.50**	1.00	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00
21	1.00	1.00	1.00	1.00	**0.35**	**0.40**	**0.40**	**0.40**	**0.73**	1.00	1.00	1.00
22	1.00	1.00	1.00	1.00	**0.57**	1.00	1.00	1.00	1.00	1.00	1.00	1.00
23	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.73**	1.00	1.00	1.00
24	**0.25**	**0.25**	**0.25**	**0.25**	1.00	1.00	1.00	1.00	**0.40**	**0.50**	**0.50**	**0.50**
25	1.00	1.00	1.00	1.00	**0.57**	**0.67**	**0.67**	**0.67**	**0.73**	1.00	1.00	1.00
26	1.00	1.00	1.00	1.00	**0.80**	1.00	1.00	1.00	1.00	1.00	1.00	1.00
27	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.67**	1.00	1.00	1.00
28	**0.50**	**0.50**	**0.50**	**0.50**	**0.40**	**0.50**	**0.50**	**0.50**	**0.67**	1.00	1.00	1.00
29	1.00	1.00	1.00	1.00	**0.47**	**0.67**	**0.67**	**0.67**	**0.89**	1.00	1.00	1.00
30	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
31	1.00	1.00	1.00	1.00	**0.89**	1.00	1.00	1.00	1.00	1.00	1.00	1.00
32	**0.25**	**0.25**	**0.25**	**0.25**	**0.50**	**0.50**	**0.50**	**0.50**	**0.25**	**0.25**	**0.25**	**0.25**

throughput of 0.5. Since the pattern of bank conflicts is regular, there are no transients and buffers are of no use.

A 1-skew scheme actually results in conflicts for more strides than the standard interleaving scheme. However, many of the conflicts are transient in nature and the degradation due to such transients can be reduced by the use of buffers. Consider, for example, stride 5 accesses (refer Figure 2). Addresses 40 and 55 both lie in bank 5 and, therefore, will result in a conflict. However, other stride 5 accesses, i.e., 5, 10, 15, 20, 25, 30 and 35 are to distinct banks and do not cause a conflict. If a buffer is provided, the request to address 55 can be buffered till bank 5 is free. The address source can proceed with generating addresses 60, 65, 70, etc., without waiting for a busy bank to become free, thereby allowing the memory system to have a throughput of 1 element per cycle in the steady state.

The simple logical skewing scheme used in our experiments does not perform very well without buffers. However, this is not in conflict with our goals. Recall that the goal of the memory system is to improve the throughput for all strides is possible. We do this by "randomly" distributing data in the banks (thereby causing conflicts where none may have existed). Fortunately, the conflicts are of a transient nature and can be smoothed out with the use of buffers. As we can see from the table, with the use of a buffer with a single entry at the input and output of each bank, the overall throughput is superior to either the low-order or the 1-skew interleaving schemes. Further increasing the buffer size is of limited use for the given memory system parameters. With the use of buffers, strides that are 8, 16 or 24 modulo 32 have a throughput of 0.5, strides that are a multiple of 32 have a throughput of 0.25 and all other strides have a steady-state throughput of 1.

6. Summary and Conclusions

In this paper, we discussed the design of interleaved memory systems for vector processors. The goal of such a design is to achieve a throughput of 1 element per clock cycle for a variety of strides (for a single vector access stream). In order to achieve this goal, the memory system must skew the data across the banks and provide buffers to smooth out any transient effects.

The main contribution of this paper is the use of logical skewing schemes for interleaved memories in vector processors. Logical skewing schemes allow the distribution of data in the memory banks in a non-regular fashion so that the probability of a repeating sequence of conflicting requests is reduced. The elegance of logical skewing schemes lies in the fact that the process of locating an addressed data element in the memory system relies solely on the logical manipulation of the address bits and does not involve any arithmetic calculations. Such schemes can, therefore, be implemented without a significant increase in the memory latency.

Based on the idea of logical skewing schemes, we also introduced the concept of a programmable interleaved memory system. In a programmable interleaved memory, the distribution of data in the memory banks can be altered under program control without modifying the programs' view of data storage.

We carried out a simulation analysis of a memory system with a logical skewing scheme and compared it to an equivalent memory system with standard interleaving and a memory system with 1-skew storage. The simulation results indicate that the performance of a memory system with a logical skewing scheme is superior to other known memory systems.

References

[1] D. J. Kuck and B. Kumar, "A System Model for Computer Performance Evaluation," *Proc. International Symposium on Computer Performance Modeling, Measurement and Evaluation*, pp. 187-199, March 1976.

[2] P. Budnick and D. J. Kuck, "The Organization and Use of Parallel Memories," *IEEE Transactions on Computers*, vol. C-20, pp. 1566-1569, December 1971.

[3] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Transactions on Computers*, vol. C-24, pp. 1145-1155, December 1975.

[4] H. D. Shapiro, "Theoretical Limitations on the Efficient Use of Parallel Memories," *IEEE Transactions on Computers*, vol. C-27, pp. 412-428, May 1978.

[5] D. H. Lawrie and C. R. Vora, "The Prime Memory System for Array Access," *IEEE Trans. on Computers*, vol. C-31, pp. 435-442, May 1982.

[6] H. A. G. Wijshoff and J. van Leeuwen, "The Structure of Periodic Storage Schemes for Parallel Memories," *IEEE Transactions on Computers*, vol. C-34, pp. 501-505, June 1985.

[7] T. Cheung and J. E. Smith, "A Simulation Study of the CRAY X-MP Memory System," *IEEE Transactions on Computers*, vol. C-35, pp. 613-622, July 1986.

[8] W. Oed and O. Lange, "On the Effective Bandwidth of Interleaved Memories in Vector Processor Systems," *IEEE Trans. on Computers*, vol. C-34, pp. 949-957, October 1985.

[9] D. T. Harper and J. R. Jump, "Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme," *IEEE Transactions on Computers*, vol. C-36, pp. 1440-1449, December 1987.

[10] D. J. Kuck and R. A. Stokes, "The Burroughs Scientific Processor (BSP)," *IEEE Trans. on Computers*, vol. C-31, pp. 363-376, May 1982.

[11] J. M. Frailong, W. Jalby, and J. Lenfant, "XOR-Schemes: A Flexible Data Organization in Parallel Memories," *Proceedings 1985 International Conference on Parallel Processing*, pp. 276-283, August 1985.

[12] A. Norton and E. Melton, "A Class of Boolean Linear Transformations for Conflict-Free Power-Of-Two Access," *Proceedings 1987 International Conference on Parallel Processing*, pp. 247-254, August 1987.

[13] A. J. Smith, "Cache Memories," *ACM Computing Surveys*, vol. 14, pp. 473-530, Sept. 1982.