

# High-Bandwidth Interleaved Memories for Vector Processors—A Simulation Study

Gurindar Singh Sohi, *Member, IEEE*

**Abstract**—Sustained memory bandwidth for a range of access patterns is a key to high-performance vector processing. Interleaving is a popular way of constructing a high-bandwidth memory system. However, for some access patterns conflicts reduce the bandwidth of a standard, low-order interleaved memory. To improve memory bandwidth for a wide range of access patterns, alternate interleaving schemes must be considered. This paper studies a family of alternate interleaving schemes called permutation-based interleaving schemes. Permutation-based interleaving schemes can be implemented with a small amount of additional hardware and with a minimal time overhead. A detailed simulation analysis has been carried out in this paper. The simulation analysis suggests that, with adequate buffering, permutation-based interleaving schemes similar to those studied in this paper can be used to implement a high-bandwidth memory system for vector processors. The resulting memory system sustains its bandwidth for a wide variety of access patterns and for large bank busy times far better than a memory system with standard interleaving.

**Index Terms**—Interleaved memories, permutation-based interleaving, simulation analysis, sustained throughput, vector processing.

## I. INTRODUCTION

IT is well understood that the rate at which a processor can process information is limited by the rate at which the processor can access information from the memory. An increase in the processor speed must be coupled with a corresponding increase in the bandwidth of the memory. Several schemes to improve the bandwidth of memory exist. Cache memories provide an excellent means of improving the latency as well as the bandwidth of the memory as seen by the processor. However, their utility is limited by their size. In vector processing machines, cache memories have, to date, not been proven to be useful in providing high-bandwidth access to elements of large data structures. To achieve such high-bandwidth access, vector processing machines routinely use *parallel* or *interleaved* memories. A parallel or interleaved memory consists of several memory *modules* or *banks*. Parallel and interleaved memories improve memory bandwidth by allowing elements in distinct banks to be accessed in an overlapped manner. However, the actual bandwidth of the memory system may be less than the peak bandwidth because of *collisions* or *conflicts*. A bank conflict occurs when the

processor makes a request to a memory bank that is already busy servicing a previous request.

For our discussion in this paper, we distinguish between parallel and interleaved memories. By parallel memories, we mean the memory modules of a SIMD array processor such as the Burroughs Scientific Processor [11]. The memory modules of a parallel memory are connected to  $P$  processing elements (PE's) via an interconnection network. In the array processing paradigm, each PE operates on a different element of the same vector in the same cycle. The parallel memory system must, therefore, be capable of supplying  $P$  elements of the same vector in every cycle. By interleaved memories, we mean the memory modules of a SIMD vector processor such as the Cray-1 [19], or the follow-on Cray X-MP and the Cray Y-MP. In the SIMD vector processing paradigm, a single PE operates on the elements of a single vector, one at a time. Ideally, an interleaved memory system for a SIMD vector processor would have a throughput of  $V$  elements (possibly from  $V$  different vectors) per cycle where  $V$  is the number of vector access streams that are active simultaneously.

The problem of conflicts in parallel and interleaved memories has been the subject of intensive study [3], [4], [6], [12], [13], [15], [20], [24]. For interleaved memories, most of the previous work has focused on conflicts in the memory system for a single vector access stream though recent work has also investigated conflicts in multiport memory systems that allow simultaneous, multiple vector access streams [4], [15].

This paper is concerned with high-bandwidth interleaved memories for vector processors. We restrict our discussion to single port interleaved memory systems with a single vector access stream. An example of a computer with such a memory system is the Cray-1. Of course, the ideas presented in this paper can be extended to the design of a multiport memory system that allows multiple streams of access (such as the Cray X-MP and the Cray Y-MP). Our goal is to design a memory system that can sustain its peak throughput for a wide range of vector access patterns.

The outline of this paper is as follows. Section II discusses the basic concepts in the design of interleaved memory systems for vector processors. Section III reviews alternate interleaving schemes that have been studied previously for both parallel and interleaved memories. Section IV discusses permutation-based interleaving schemes that use bit-wise Boolean operations to generate memory addresses. Section V contains a detailed, comparative simulation analysis of various interleaving schemes. Section VI contains a discussion and Section VII presents some concluding remarks.

Manuscript received June 21, 1988; revised October 8, 1991. This work was supported in part by NSF Grants CCR-8706722 and CCR-8919635.

The author is with the Computer Sciences Department, University of Wisconsin, Madison, Madison, WI 53706.

IEEE Log Number 9204952.

## II. 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. 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. Before proceeding further, some definitions are in order.

*Definition 1:* The *stride* of a vector access is the difference in the linear memory address between successive elements of the vector.

*Definition 2:* An interleaved memory system of  $M$  banks and  $M \times N$  words is composed of  $N$  linear subvectors of  $M$  elements each. The  $M$  elements  $\{X_j\}$  of a linear subvector  $i$  are  $\{X_j | X_j = i \times N + j; 0 \leq j \leq M - 1\}$ .

*Definition 3:* The *bank busy time* of an interleaved memory system is the number of processor clock cycles that a memory bank is busy servicing a request.

Let  $M$  be the number of banks in the interleaved memory. In most interleaved memories,  $M$  is a power of 2, i.e.,  $M = 2^n$  where  $n$  is an integer. However,  $M$  need not be restricted to a power of 2. Indeed, memory systems that have a *prime* number of banks have been studied [13] and built [11]. However, in a memory system with an arbitrary value of  $M$ , the process of determining the location of a desired data element is quite complicated [13]. By restricting  $M$  to be a power of 2, this process is greatly simplified. We shall, therefore, restrict our discussion to memory systems with a power-of-two number of banks.

The simplest and most common interleaving scheme, i.e., a *low-order* or *standard* 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. For such an interleaving scheme, the bank number in which an arbitrary address  $i$  is located is specified by  $i \bmod M$  and the word within the bank for the address  $i$  is  $i \div M$ . Successive elements in the linear memory address space are placed in successive banks.

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 for vector processors because they do not allow for peak-throughput access to a vector of stride 1.

If an interleaving scheme uses the low-order  $p$  bits to select the bank, address  $(2^p \times l + m)$  where  $l$  is an integer greater than 0 and  $0 \leq m \leq 2^p - 1$  will map on to the same bank as address  $m$  and, therefore, the distribution pattern of addresses amongst the banks will repeat after  $2^p$  elements. In a standard

interleaved memory, successive components of a vector with stride 1 fall in successive banks and can be accessed in a conflict-free fashion. However, successive components of a vector with stride  $M$  lie in the same bank and will result in conflicts. Conflict situations can also arise for other strides depending upon the bank busy time.

If the access patterns and the size of the data structures are known *a priori* at compile time, *array reshaping* techniques can be used to minimize the possibility of memory bank conflicts. 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 bank conflicts for common reference patterns are reduced at the expense of wasted memory. However, such techniques are of limited utility if sufficient compile-time information is not available and we shall not consider them any further in this paper. The interested reader is referred to [6] for some discussion on such techniques.

## III. DATA SKEWING ALTERNATE INTERLEAVING SCHEMES

Reserchers have long realized the inadequacy of a standard interleaving scheme for parallel memory systems. To reduce the probability of conflicts, *data skewing* schemes were introduced. Several data skewing schemes have been proposed and analyzed in the literature [3], [5], [10], [12], [24]. In a memory system with a data skewing scheme, the mapping from the linear address space to the banks of the memory system is specified by the skewing scheme. In this paper, any interleaving scheme which uses a mapping function from the linear address space to the memory banks other than the mapping function used in standard interleaving shall be called an *alternate interleaving scheme*.

One of the first alternate interleaving schemes, *linear data skewing* schemes, were proposed for parallel memories in [10] and analyzed in [12] and [24]. In a linear skewing scheme, element  $a_{jk}$  of a two-dimensional array  $A$  is stored in memory bank  $j * \delta_1 + k * \delta_2$  where  $\delta_1$  and  $\delta_2$  are integers [12]. The scheme generalizes for higher dimension arrays. However, to implement a linear skewing scheme, the bank selection hardware needs the capability to carry out arithmetic operations. The time taken to carry out arithmetic operations can easily degrade the latency of each memory request.

To simplify the implementation (eliminate the need for arithmetic operations), Frailong, Jalby, and Lenfant presented a family of interleaving schemes called *XOR schemes* or *Boolean schemes* for parallel memories [5]. Such schemes rely on cheaper, bit-wise Boolean operations to determine the memory module for an arbitrary address. Melton and Norton have shown how such schemes can be used to achieve conflict-free access to power-of-two strides in the IBM RP3 processor [14].

Traditionally, data skewing schemes were studied for parallel memories in SIMD array processors. Recently, however, Harper and Jump proposed and studied a linear data skewing scheme for an interleaved memory in a vector processor [6]. In the *I-Skew* interleaving scheme proposed by Harper and Jump, the bank number ( $M_i$ ) for an arbitrary address  $i$  in the

linear memory is calculated as:

$$M_i = \left( i + \left\lfloor \frac{i}{M} \right\rfloor \right) \bmod M.$$

If  $M$  is a power of two ( $M = 2^n$ ), the division and modulo operations are trivial and the bank number calculation involves a single  $n$ -bit addition.

Consider the distribution of 128 elements (16 linear sub-vectors of 8 elements each) from the linear address space amongst 8 banks using a 1-Skew storage scheme (see Fig. 1). The elements of a subvector that comprises word  $k$  in the memory banks are distributed in a regular fashion, i.e., the only difference between word 0 and word  $k$  is that the first element of word  $k$  is placed in bank  $k \pmod{2^n}$ . Furthermore, since only  $2n = 6$  bits of the address are used to compute the bank, the distribution pattern repeats after  $2^{2n} = 2^6 = 64$  elements (see Fig. 1).

When compared to a standard interleaving scheme, the 1-Skew scheme eliminates conflicts for strides such as stride 8 but introduces conflicts for some other strides, for example, stride 14. However, the degradation due to many of the conflicting strides can be reduced or even eliminated if buffers are provided at the input and the output of each memory module. For example, if 3 buffer elements are provided at the input and the output of each memory bank, stride 14 accesses can proceed at the rate of 1 element per cycle (see analysis in Section VI). Likewise, stride 7 accesses can proceed at the rate of 1 element per cycle if 6 buffer elements were provided at the input and output of each memory module. Harper and Jump provide a detailed analysis on the number of buffers that is needed in [6].

Unfortunately, the 1-Skew interleaving scheme (or any other scheme that uses linear data skewing) has an inherent disadvantage—it involves arithmetic manipulation of the address bits. This arithmetic manipulation can degrade latency of each memory operation. However, the performance advantage of 1-Skew interleaving over standard interleaving presented in [6] suggests that an alternate interleaving scheme in memory systems for vector processors may be worthwhile.

Before proceeding further, let us distinguish between *conflict-free* and *peak-throughput* access. *Conflict-free* access occurs when references to the same memory bank do not occur within a time window determined by the bank busy time. Indeed, most alternate interleaving schemes presented in the literature are concerned with conflict-free access to a particular set of strides.

If no conflicts occur in accessing data, the data accesses can proceed at the maximum rate allowed by the memory system, i.e., with *peak-throughput*. However, peak-throughput access is possible even if the accesses are not conflict-free (see, for example, analysis for strides 7 and 14 of Fig. 1 in Section V-B). If a conflict situation occurs and the conflicting request can be buffered, succeeding requests might be able to proceed allowing peak-throughput access [2], [6].

Since conflict-free access to a wide variety of strides is not possible [3], one can attempt to improve the throughput for a wide variety of strides by using alternate interleaving schemes. Ideally, the interleaving scheme would minimize the

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
8	64	65	66	67	68	69	70	71
9	79	72	73	74	75	76	77	78
10	86	87	80	81	82	83	84	85
11	93	94	95	88	89	90	91	92
12	100	101	102	103	96	97	98	99
13	107	108	109	110	111	104	105	106
14	114	115	116	117	118	119	112	113
15	121	122	123	124	125	126	127	120

Fig. 1. Distribution of elements using 1-Skew storage.

possibility of a long sequence of accesses to the same bank. Then, by providing buffers, memory requests can be allowed to proceed even though previous requests may be waiting (in the buffers) for a memory bank to become free. However, keep in mind that the interleaving scheme should be simple enough that its implementation does not degrade the latency of each memory request.

Fortunately, interleaving schemes that involve bit-wise logical operations on the address bits seem to fit the above-mentioned requirements and therefore we shall investigate their use in a high-bandwidth interleaved memory system. We shall refer to such schemes as *Permutation-Based Interleaving (PBI)* schemes and the resulting memory system as a *Permutation Interleaved (PI)* memory system.

#### IV. PERMUTATION-BASED INTERLEAVING (PBI) SCHEMES

In order to locate a desired word in an interleaved memory system with  $M = 2^n$  banks and  $2^{N-n}$  words in each bank, we need to specify: i) an  $n$ -bit bank number and ii) an  $N - n$  bit address that indicates the position of the word in the selected bank. In the PBI schemes that we use in this paper, the position of the word within each bank (or the linear subvector number) is determined in the same way as it is determined in a standard low-order interleaved memory, i.e., by using the high-order  $N - n$  bits of address.

The power and the elegance of PBI schemes lies in the calculation 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 number 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:

$$\bar{Y} = \mathbf{A}\bar{X} \quad (1)$$

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 ( $\oplus$ )" being a logical Exclusive-OR operation. Element  $Y_i$  of  $\bar{Y}$  is, therefore:

$$Y_i = (A_{i,0} * X_0) \oplus (A_{i,1} * X_1) \oplus \cdots \oplus (A_{i,N-1} * X_{N-1}) \quad (2)$$

where  $X_j$  is the  $j$ th bit of the address.

A memory system using the PBI schemes described above can be viewed as a memory system consisting of  $2^{N-n}$

subvectors each of  $2^n$  elements. The subvectors are distributed in a regular, ordered fashion amongst the memory banks with one component of a subvector in each memory bank. However, the distribution of the elements of a subvector amongst the memory banks can be quite irregular thereby leading to an overall randomness. The name “PBI schemes” is derived from the fact that the elements of the subvectors are permuted amongst the banks in different ways.

Notice that by modifying the contents of the  $A$  matrix, we can obtain PBI schemes that distribute the elements of a subvector across the memory banks in a nonregular fashion. That is, the distribution of the elements of subvector (word)  $i$  have no clear relationship to the distribution of elements of subvector (word)  $j$ .

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

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

i.e.,  $Y_0 = X_4 \oplus X_3 \oplus X_1$ ,  $Y_1 = X_5 \oplus X_2 \oplus X_1$ , and  $Y_2 = X_5 \oplus X_4 \oplus X_1 \oplus X_0$ , the distribution of 64 elements of the linear address space amongst 8 banks is given in Fig. 2. In this case, the bank selection process involves 6 bits and, therefore, the distribution pattern repeats after 64 elements.

Before proceeding with an analysis of PBI schemes, let us discuss some issues in the design and the properties of the  $A$  matrix. However, the main thrust of the paper is to present the use of PBI schemes in a high-bandwidth PI memory and evaluate their effectiveness using a detailed simulation analysis. Therefore, we shall keep theoretical discussion to a bare minimum.

We can rewrite (1) as:

$$\bar{Y} = [A_H | A_L] \begin{bmatrix} \bar{X}_H \\ \bar{X}_L \end{bmatrix} = A_H \bar{X}_H + A_L \bar{X}_L \quad (4)$$

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 PBI schemes. For a standard interleaving scheme,  $A_H$  is the zero matrix and  $A_L$  is the identity matrix.

A general PBI scheme must make sure that each memory address maps on to a distinct location in the memory system, i.e., is a distinct word within a distinct bank. Since the word selection process of our PBI schemes does not involve any Boolean operations other than simple bit selection, the word within the selected memory bank for each memory address is unique and exactly  $2^n$  elements (the elements of a linear subvector) have the same word within the memory banks. The PBI 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 memory system. In order to do so, the  $A_L$  matrix must be

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

Fig. 2. Distribution of elements using the permutation based interleaving scheme of (3).

chosen properly. The following theorem guides the selection of the  $A_L$  matrix.

**Theorem:** A PI memory system has a unique location for each addressed element iff the associated matrix  $A_L$  is of full rank with respect to the Boolean matrix multiplication operation.

**Proof:** We shall prove the necessary and sufficient conditions separately by making use of well-known results in linear algebra.

For a given linear subvector  $i$ ,  $\bar{X}_H$  is constant and  $\bar{X}_L$  can take on  $2^n$  distinct values. Therefore, the term  $A_H \bar{X}_H$  in (4) is a constant. Since  $\bar{Y} = A_H \bar{X}_H + A_L \bar{X}_L$ , the submatrix  $A_L$  is solely responsible for guaranteeing a one-one mapping between the  $2^n$  elements of the linear subvector  $i$  and the  $2^n$  memory banks. To do so,  $A_L$  must be of rank  $n$ . If the rank of  $A_L$  is  $m$ ,  $0 \leq m < n$ , then the  $2^n$  elements of the subvector would be mapped to a subset of the memory which contains  $2^m$  banks. This proves the necessary condition.

Now we prove the sufficient condition. Since  $A_L$  is of rank  $n$ , the term  $A_L \bar{X}_L$  will generate  $2^n$  distinct bank numbers for the  $2^n$  elements of a linear subvector  $i$  (word within bank). Since  $\bar{X}_H$  is a constant for a given linear subvector  $i$ ,  $A_H \bar{X}_H$  is a constant and does not alter the one-one mapping. Therefore, the bank selection process will generate a unique bank number for each element of a linear subvector and consequently guarantee a unique location in the memory system for each addressed data element. Q.E.D.

While designing a PBI scheme, the only consideration is that  $A_L$  should be of full rank with respect to the Boolean matrix multiplication operation.  $A_H$  participates in the interleaving scheme by permuting the elements of different subvectors in different ways (see the distribution of the elements from different words in Fig. 2). Any choice of  $A_H$  will result in a valid PBI scheme. Of course, PBI schemes with different  $A_H$  submatrices will have different vector access performance for different strides.

#### A. Bank Number Calculation in a PI Memory

Let us now consider the hardware needed to calculate the position of an arbitrary word in the PI memory system. Keep in mind that unless the hardware needed to implement the PI memory is extremely simple, cheap and fast, its use in a high-performance system is unlikely. Recall that our PBI schemes use 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.

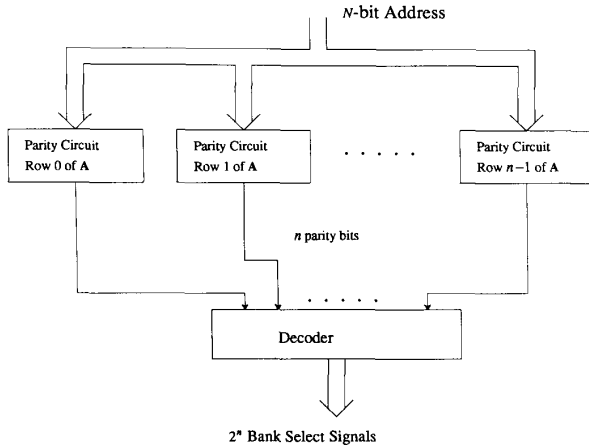


Fig. 3. Hardware organization for determining the bank number in a permutation interleaved memory.

The additional hardware needed for a PI memory system is, therefore, the hardware needed to compute the bank number.

From (2) we see that each bit of the  $n$ -bit vector  $\bar{Y}$  is determined by computing an appropriate Boolean sum (parity) of the input address bits. The bits that participate in the Boolean sum for 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 Fig. 3.  $N$  address bits are fed into  $n$  parity computation circuits that compute the appropriate Boolean sum. 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 addressed 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 for the PI memory is, therefore, the delay through the parity computation circuits. For a simple PBI scheme such as the one described by (3), this overhead can be as little as the delay through a single 4-input XOR gate.

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  $\lceil \log_k N \rceil$  levels of XOR gates where  $k$  is the fan-in of each XOR gate. For  $k = 4$ , the parity of up to 64 input bits can be computed with only 3 levels of XOR logic.

The reader should note that in some cases, it may be possible to merge part or all of the parity computation logic with the decoding logic. In other cases, it may be possible to incorporate some of the parity computation 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 PI memory has no additional time overhead over a standard interleaved memory. In any case, the time overhead for an arbitrary PBI scheme is not more than a few levels of logic and is far less than the overhead for interleaving schemes that involve arithmetic

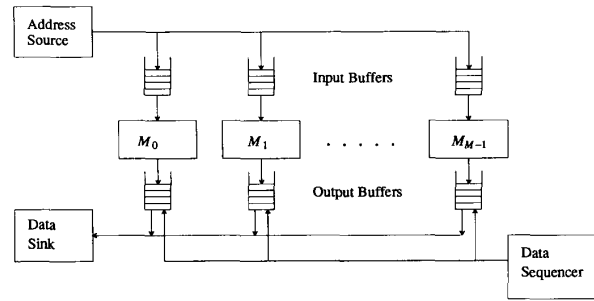


Fig. 4. An interleaved memory system with input and output buffers.

calculations (a carry ripple) in the calculation of the bank number. Furthermore, the additional hardware is of the order of a few XOR gates and is less than the additional hardware needed to implement an interleaving scheme that uses linear data skewing.

## V. A SIMULATION ANALYSIS

To evaluate the relative performance of a PI memory system, we carried out a detailed simulation analysis. Our model of the memory system is the same as the one proposed by Harper and Jump [6] and is shown in Fig. 4. Before proceeding further with the experiments, let us describe the memory system in some more detail.

### A. 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 input buffer of the selected bank for service. All requests pass through the input buffer; no request is forwarded to the bank directly even if the bank is free. The address source also provides a sequence number for each memory request. The sequence number is used by the data sequencer to return the data back to the data sink in the same order as the addresses generated by the address source.

Each memory bank services requests from its input buffer in the order that they were submitted to the buffer. A serviced request, along with its sequence number, is placed in the output buffer of the bank.

The data sequencer keeps track of the sequence number of the next element to be returned to the data sink. It monitors the sequence numbers in the output buffers of the banks. When a match results, the corresponding data item is sent to the data sink, the relevant output buffer is updated and the sequence number in the data sequencer is also updated. Both the address source and the data sink maintain their sequence numbers by starting out with a zero value and incrementing their current sequence number modulo a given number. To ensure correct operation, the number should be larger than the maximum number of data requests that can be outstanding at any given instant.

Note that the input and output address latches of a standard interleaved memory can be regarded as buffers of size one. Indeed, in our simulations, we treat the input address latch of

each memory bank as an input buffer of size 1. Likewise, a latch at the output of each bank is treated as an output buffer of size 1.

The throughput of the memory system can be calculated in two ways. The first method of calculating throughput is to divide the number of data elements accessed by the number of cycles taken by the address source to generate all the addresses and submit them to the banks for service. Thus, if the address source took 1024 cycles to supply the addresses of 1024 elements, the throughput would be 1 element per cycle even though the access operation takes more than 1024 cycles to *complete*. In this method, the time taken to flush the memory banks and buffers and return the last data item to the data sink is ignored.

If no buffers are present in the memory system, the time taken to flush the memory system is constant; the address source blocks when a bank is busy and the access operation is complete  $B$  cycles after the last request is issued where  $B$  is the bank busy time. However, if buffers are present *and* the access pattern is not highly deterministic (as is the case in the 1-Skew and PBI schemes), requests can queue up at the input of a single bank (consider, for example, stride 14 access in Fig. 1) and the time taken to flush a sequence of requests may be quite significant. A more realistic measure of the throughput would, therefore, use the *total* time taken to access the data elements. The second method for calculating throughput is to divide the number of cycles that would ideally be taken to access the vector in a conflict-free fashion (vector length + bank busy time + time through buffers) by the total number of cycles that are actually taken to access the vector. We shall use this method to calculate the throughput for all our experiments since we believe it is more realistic in a vector processing environment.

### B. Simple Interleaving Schemes

Our first set of experiments compares standard interleaving with 1-Skew interleaving and a simple PBI scheme. We do not attempt to analyze various PBI schemes in this paper. (However, see Section VII for related work.) For the experiments of this section, we use the 6-bit PBI scheme described by (3). Our choice of the PBI scheme for this experiment was quite arbitrary; the main criterion was to involve 6 bits of address in the computation of the bank number, the same as in a 1-Skew scheme (indeed, several other 6-bit logical skewing schemes that we considered had very similar performance). In Section V-C, we investigate other PBI schemes that use more address bits to compute the bank number.

To prevent an explosion in the number of experiments, we do not attempt to evaluate the orthogonal effect of the number of memory banks on the throughput. Clearly, if more banks are available with the same bank busy time, the probability of bank conflicts is reduced and the average throughput is increased. All the experiments in this paper are carried out on a memory system with 8 banks.

In the first experiment, the bank busy time is set to 4 clocks (we consider other bank busy times in Section V-B1), the stride of the input vectors was varied from 1 to 64, and the

TABLE I  
THROUGHPUT VERSUS STRIDE FOR VARIOUS INTERLEAVING SCHEMES

Stride	Interleaving Scheme								
	Standard			1-Skew			PBI		
	Input/Output Buffer Size			Input/Output Buffer Size			Input/Output Buffer Size		
	1	2	3	1	2	3	1	2	3
1	—	—	—	—	—	—	—	—	—
2	—	—	—	—	—	—	—	—	—
3	—	—	—	—	—	—	—	—	—
4	0.50	0.50	0.50	—	—	—	0.94	—	—
5	—	—	—	—	—	—	—	—	—
6	—	—	—	—	—	—	—	—	—
7	—	—	—	0.31	0.36	0.44	0.80	0.99	0.99
8	0.25	0.25	0.25	—	—	—	—	—	—
9	—	—	—	—	—	—	0.76	0.99	0.99
10	—	—	—	—	—	—	—	—	—
11	—	—	—	—	—	—	0.89	—	—
12	0.50	0.50	0.50	—	—	—	—	—	—
13	—	—	—	—	—	—	0.76	0.99	0.99
14	—	—	—	0.40	0.67	0.99	0.80	—	—
15	—	—	—	—	—	—	0.91	—	—
16	0.25	0.25	0.25	—	—	—	—	—	—
17	—	—	—	—	—	—	0.94	—	—
18	—	—	—	—	—	—	—	—	—
19	—	—	—	0.86	0.99	0.99	0.80	0.99	0.99
20	0.50	0.50	0.50	—	—	—	—	—	—
21	—	—	—	0.57	0.99	0.99	0.89	—	—
22	—	—	—	—	—	—	0.73	0.99	0.99
23	—	—	—	—	—	—	0.89	—	—
24	0.25	0.25	0.25	—	—	—	—	—	—
25	—	—	—	0.73	0.99	0.99	0.89	—	—
26	—	—	—	—	—	—	0.80	—	—
27	—	—	—	—	—	—	0.80	—	—
28	0.50	0.50	0.50	—	—	—	—	—	—
29	—	—	—	—	—	—	0.91	—	—
30	—	—	—	—	—	—	—	—	—
31	—	—	—	—	—	—	—	—	—
32	0.25	0.25	0.25	0.50	0.50	0.50	0.50	0.50	0.50

Stride	Interleaving Scheme								
	Standard			1-Skew			PBI		
	Input/Output Buffer Size			Input/Output Buffer Size			Input/Output Buffer Size		
	1	2	3	1	2	3	1	2	3
33	—	—	—	—	—	—	—	—	—
34	—	—	—	—	—	—	0.91	—	—
35	—	—	—	—	—	—	—	—	—
36	0.50	0.50	0.50	—	—	—	0.80	—	—
37	—	—	—	—	—	—	0.80	—	—
38	—	—	—	—	—	—	0.89	—	—
39	—	—	—	0.73	0.99	0.99	—	—	—
40	0.25	0.25	0.25	—	—	—	—	—	—
41	—	—	—	—	—	—	0.89	—	—
42	—	—	—	—	—	—	0.73	0.99	0.99
43	—	—	—	—	—	—	0.89	—	—
44	0.50	0.50	0.50	0.57	0.99	0.99	—	—	—
45	—	—	—	0.86	0.99	0.99	0.80	0.99	0.99
46	—	—	—	—	—	—	—	—	—
47	—	—	—	—	—	—	0.94	—	—
48	0.25	0.25	0.25	—	—	—	—	—	—
49	—	—	—	—	—	—	0.91	—	—
50	—	—	—	0.40	0.66	0.99	0.80	—	—
51	—	—	—	—	—	—	0.76	0.99	0.99
52	0.50	0.50	0.50	—	—	—	—	—	—
53	—	—	—	—	—	—	0.89	—	—
54	—	—	—	—	—	—	—	—	—
55	—	—	—	—	—	—	0.76	0.99	0.99
56	0.25	0.25	0.25	—	—	—	—	—	—
57	—	—	—	0.31	0.36	0.44	0.80	0.99	0.99
58	—	—	—	—	—	—	—	—	—
59	—	—	—	—	—	—	0.94	—	—
60	0.50	0.50	0.50	—	—	—	—	—	—
61	—	—	—	—	—	—	—	—	—
62	—	—	—	—	—	—	—	—	—
63	—	—	—	—	—	—	—	—	—
64	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

length of each vector was 1024 elements (in Section V-B2 we consider the effect of different vector lengths). The number of input and output buffers with each bank was varied from 1 to 3 (the number of input buffers is equal to the number of output buffers). The results for this set of experiments are presented in Table I. The cases for which a throughput of 1 cannot be achieved have been shown (the unfilled entries have a throughput of 1).

With 8 memory banks, a standard interleaving scheme uses only the low-order 3 bits of address to select the bank and,

therefore, the pattern of throughput repeats after a stride of 8. 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 throughput of 0.5. Since the pattern of bank conflicts is regular, there are no transients and buffers are of no use.

Both the 1-Skew and PBI schemes use 6 address bits to compute the bank number and the throughput pattern repeats after a stride of 64. In the 1-Skew scheme, many of the conflicts that arise 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 Fig. 1). Addresses 40 and 55 both lie in bank 5 and will result in a conflict if the bank busy time is more than 4 clock cycles. 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 until bank 5 is available. The address source can proceed with generating addresses 60, 65, 70, etc., without waiting for bank 5 to become free, thereby allowing the memory system to have a throughput of 1 element per cycle. Likewise, the throughput for other strides can be increased. For example, the throughput for strides 7 and 57 peaks out at 0.98 if the number of buffers is equal to 6. Therefore, if 1-Skew interleaving is used and adequate buffering is provided, strides that are multiples of 64 have a throughput of 0.25 elements per cycle, strides that are multiples of 32 (but not 64) have a throughput of 0.5 elements per cycle, all other strides have a peak or near-peak throughput.

The simple PBI scheme used in our experiments does not perform very well without buffers in addition to the address and data latches. However, this is not in conflict with our goals. As with the 1-Skew scheme, the conflicts that occur are of a transient nature and the degradation due to such conflicts can be alleviated by the use of additional buffers. As we can see from Table I, with only 2 buffers at the input and output of each bank, the overall throughput is superior to the throughput of both the standard and the 1-Skew interleaving schemes. For the simple 6-bit PBI scheme, with adequate buffering, strides that are 32 modulo 64 have a throughput of 0.5, strides that are a multiple of 64 have a throughput of 0.25 and all other strides have peak or near-peak throughput.

*1) Effect of Bank Busy Time:* It is quite obvious that as the bank busy time is increased, the throughput of the memory system degrades [1]. However, because of the nonregular access pattern and the presence of buffers, there is a possibility that the degradation will not be as severe in the 1-Skew and PI memories as in the case of standard interleaving.

In order to evaluate the effect of bank busy time on throughput, we carried out a second set of experiments. For this set of experiments, we considered a memory with 8 banks and evaluated the average throughput achieved in accessing 1024 element vectors for the different interleaving schemes. The average throughput is the average of the throughputs for each stride (note that since the number of elements accessed is constant, the arithmetic and harmonic means are the same). We realize that stride 1 accesses are most important. However, all the interleaving schemes achieve peak-throughput access for stride 1 and, therefore, the average throughput provides

a good metric for measuring the performance of the memory system for the remaining strides. We should point out that the weighted average throughput would be greater if more weight is given to stride one access as in [6].

Fig. 5 presents the results of our experiments for bank busy times varying from 1 to 8 clock cycles. A bank busy time of 1 clock cycle is the trivial case since all memory requests can proceed with peak throughput. Bank busy times of greater than 8 clock cycles were not considered because there is no hope of achieving a throughput of 1 element per clock cycle if the bank busy time is greater than the number of memory banks, regardless of the interleaving scheme used.

From Fig. 5 one can see that, while the performance of a standard interleaved memory degrades considerably as the bank busy time is increased, both the 1-Skew and PI memories retain their performance for large bank busy times. The result is not counter-intuitive. If the access pattern is not regular, one can tolerate larger bank busy times by increasing the amount of buffering available. We consider this aspect of alternate interleaving schemes to be very important for memory system design in current vector processing supercomputers. With clock speeds shrinking more rapidly than memory speeds, the relative bank busy times are increasing. To allow for adequate vector access throughput using standard interleaving, the number of memory banks would have to be increased considerably [1]. Increasing the number of banks arbitrarily is not an attractive option. Alternate interleaving schemes that can achieve peak throughput with a relatively large bank busy time are, therefore, quite attractive. This trend has been verified for a memory system with 64 banks and with bank busy times ranging from 16 to 64 clock cycles.

*2) Effect of Vector Length:* In the 1-Skew and PBI schemes, several elements may be queued at the input of a memory bank when the address source finishes generating requests. Depending upon the number of queued requests and the relative vector length, the time taken to flush the memory system can have a significant impact on performance. In the experiments so far, we have used a vector length of 1024 elements and because of the relatively long vector length, the flush time penalty is minimal. However, the flush time overhead can be significant if the vectors are short. To evaluate the flush time overhead, we carried out another set of experiments. For this set of experiments, we consider a memory system with 8 banks and 4 clock cycle bank busy time. Table II presents the average throughput of the memory system using different interleaving schemes as the vector length is varied from 64 to 1024 elements.

From Table II we observe that the standard interleaving scheme has a near-uniform throughput for varying vector lengths. The minor difference arises because of the flush penalty for strides with bank conflicts. For example, a stride 8 access of a 64-element vector will have a throughput of  $((64 + 4 + 2)/(64 \times 4 + 4 + 2) = 0.27)$  (64 elements plus 4 cycles through bank plus 2 cycles through the buffers) while a stride 8 access of a 1024-element vector will have a throughput of  $((1024 + 4 + 2)/(1024 \times 4 + 4 + 2) = 0.25)$ .

Both the 1-Skew and PBI schemes suffer a slight penalty if the vector length is small. In particular situations, if the

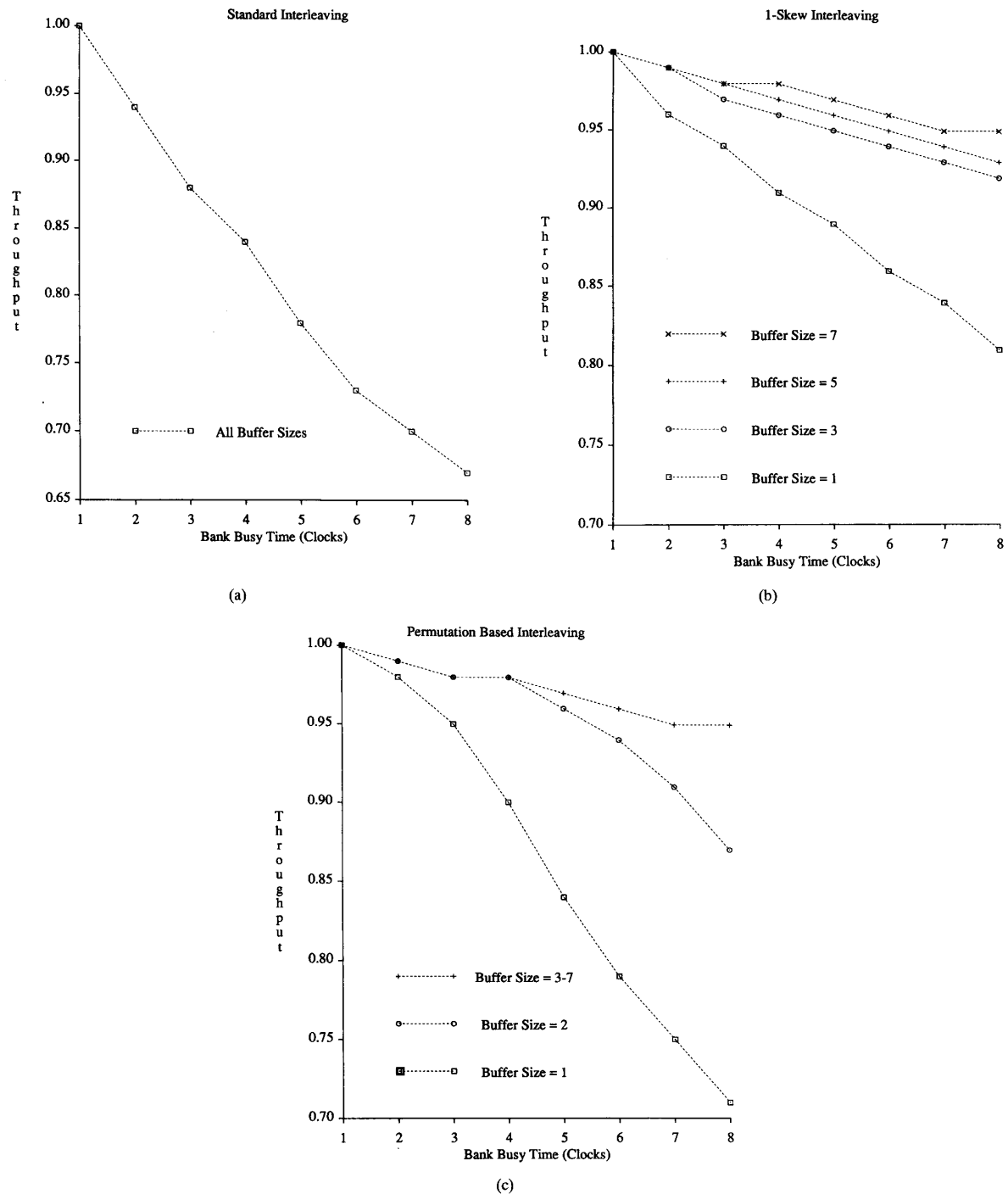


Fig. 5. Average throughput for various bank busy times vector length = 1024 elements. (a) Standard interleaving, (b) 1-Skew interleaving, (c) Permutation based interleaving.

vector length of the machine is not adequate, the flush penalty might be of concern. However, for large vector lengths, the flush penalty is minimal. Also note that the flush penalty is important because we calculate the throughput in *completing* the operation. Of course, the effects of this penalty can be

minimized even for small vector lengths by overlapping consecutive vector accesses since the address source can proceed with requests from another vector access while requests from a previous access are still queued in the buffers of the memory system.



TABLE II  
AVERAGE THROUGHPUT FOR DIFFERENT VECTOR  
LENGTHS: BANK BUSY TIME = 4 CLOCK CYCLES

Interleaving Scheme	Vector Length	Number of Buffers						
		1	2	3	4	5	6	7
Standard	64	0.84	0.84	0.84	0.84	0.84	0.84	0.84
	128	0.84	0.84	0.84	0.84	0.84	0.84	0.84
	256	0.85	0.84	0.84	0.84	0.84	0.84	0.84
	512	0.85	0.85	0.85	0.85	0.85	0.85	0.85
	1024	0.85	0.85	0.85	0.85	0.85	0.85	0.85
1-Skew	64	0.91	0.93	0.94	0.95	0.95	0.95	0.95
	128	0.91	0.94	0.95	0.96	0.96	0.97	0.97
	256	0.91	0.95	0.96	0.96	0.97	0.97	0.97
	512	0.91	0.95	0.96	0.96	0.97	0.98	0.98
	1024	0.91	0.95	0.96	0.97	0.97	0.98	0.98
PBI	64	0.88	0.93	0.93	0.93	0.93	0.93	0.93
	128	0.89	0.95	0.95	0.95	0.95	0.95	0.95
	256	0.90	0.97	0.97	0.97	0.97	0.97	0.97
	512	0.90	0.97	0.97	0.97	0.97	0.97	0.97
	1024	0.90	0.98	0.98	0.98	0.98	0.98	0.98

### C. Higher Order Interleaving Schemes

The 1-Skew scheme and the 6-bit PBI scheme discussed above make use of only  $2n = 6$  address bits to determine the bank number. Since the distribution pattern repeats after  $2^P$  elements, where  $P$  is the number of low-order address bits used in the bank selection process, by involving more address bits in the bank selection process, i.e., by increasing  $P$ , we can reduce the number of strides for which peak throughput access cannot be achieved.

It is possible to extend the 1-Skew scheme to a linear skewing scheme that uses more than  $2n$  bits to select the bank. For example, a linear skewing scheme in which the bank number ( $M_i$ ) for an arbitrary address  $i$  is calculated as:

$$M_i = \left( i + \left\lfloor \frac{i}{2^n} \right\rfloor + \left\lfloor \frac{i}{2^{2n}} \right\rfloor + \left\lfloor \frac{i}{2^{3n}} \right\rfloor \right) \bmod 2^n$$

uses  $4n$  address bits to compute the  $n$ -bit bank number. Unfortunately, both the hardware and the time overheads for linear skewing schemes increase as more and more bits are involved in the bank selection process. This is because several  $n$  bit arithmetic operations must be carried out (3  $n$ -bit additions). Therefore, higher-order linear skewing schemes are of limited utility in a high-performance memory system.

In a PI memory, involving more bits in the bank selection process is straightforward. As pointed out in Section IV-A, a PBI scheme that uses  $N$  bits of address can be implemented in  $\log_k N$  levels of XOR logic where  $k$  is the fan-in of each XOR gate. Therefore, we investigated a PBI scheme that uses 12 address bits (4 sets of 3 bits) to compute the bank number. The  $3 \times 12$   $A$  matrix for the interleaving scheme was:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

i.e.,  $Y_0 = X_{11} \oplus X_{10} \oplus X_8 \oplus X_5 \oplus X_4 \oplus X_3 \oplus X_0$ ,  $Y_1 = X_{11} \oplus X_8 \oplus X_7 \oplus X_6 \oplus X_5 \oplus X_4 \oplus X_1$  and  $Y_2 = X_{11} \oplus$

$X_{10} \oplus X_9 \oplus X_8 \oplus X_7 \oplus X_5 \oplus X_2$ . Many  $3 \times 12$   $A$  matrices are possible, each affecting the overall stride access pattern in a different way. We are mainly concerned with a scheme that allows peak-throughput access in the presence of buffers and, as we shall see, the above matrix fulfills the criterion.

We used the following reasoning in its construction. Starting from a standard interleaved memory with 8 banks, involving  $X_3$  in the calculation of  $Y_0$  eliminates stride 2 conflicts. Likewise, involving  $X_4$  in the calculation of  $Y_0$  and  $Y_1$  eliminates stride 4 conflicts and involving  $X_5$  in the calculation of  $Y_0, Y_1$  and  $Y_2$  eliminates stride 8 conflicts. This process can continue in several ways. The path we chose is to involve address bits in such a manner that each bit  $Y_i$  of the bank number computes parities of approximately the same number of address bits. This is done to make sure that the bank calculation circuit is not lopsided. The resulting matrix eliminates conflicts for strides that are powers of two (up to  $2^{11}$ ). While conflict situations do occur for other strides, the conflict pattern is of a transient nature and, by using buffers to smooth out the transients, near-peak throughput access can still be obtained.

Using the above matrix, we calculated the performance of a PBI memory with 8 banks and a bank busy time of 4 cycles. Since 12 bits of address are used, the throughput pattern repeats after a stride of  $2^{12} = 4096$ ; strides that are multiples of  $2^{12} = 4096$  have a throughput of 0.25 elements per cycle and strides that are a multiple of  $2^{11} = 2048$  but not of 4096 have a throughput of 0.5 elements per cycle. With adequate buffering, the remaining 4094 strides (and their multiples) have peak or near-peak throughput. Rather than present the throughput pattern for 4096 strides, we present the results in a slightly different fashion in Table III. The table presents the number of strides that have a throughput of less than 0.95 elements per cycle (number of strides row) and also the average throughput of the memory system as the buffer size is varied from 1 to 7 elements. The average is calculated assuming that each stride has an equal weight and the vector length is 1024 elements.

Consider the results for the three interleaving schemes for a buffer size of 6 elements. In a standard interleaving scheme, all strides that are a multiple of 4 have a throughput of less than 0.95 elements per cycle. There are 1024 such strides in the range 1-4096 (4, 8, 12, 16, ..., 4096). For 1-Skew interleaving, strides that are multiples of 32 have a throughput of less than 0.95. There are 128 such strides in the range 1-4096 (32, 64, 96, ..., 4096).

For the 12-bit PBI scheme, only 2 strides (2048 and 4096) have a throughput of less than 0.95. All other strides have a throughput of greater than 0.95. A throughput value of 0.95 was chosen as a cut off. Indeed, of the 4094 strides with a throughput greater than 0.95, 4091 had a throughput of 0.97 or greater and 4049 had a throughput of 0.98 or greater with a buffer size of 6 elements. These results indicate that, with adequate buffering, higher-order PBI schemes can be used to construct a memory system that achieves peak or near-peak throughput vector accesses for a wide variety of strides. More address bits can easily be incorporated into the PBI scheme. We do not do so in this paper.

TABLE III  
COMPARATIVE PERFORMANCE OF A 12-bit PBI SCHEME

Interleaving Scheme	Metric	Number of Buffers						
		1	2	3	4	5	6	7
Standard	# of Strides	1024	1024	1024	1024	1024	1024	1024
	Throughput	0.85	0.85	0.85	0.85	0.85	0.85	0.85
1-Skew	# of Strides	768	384	256	256	128	128	
	Throughput	0.91	0.95	0.96	0.97	0.98	0.98	
PBI	# of Strides	3901	1050	168	21	6	2	2
	Throughput	0.85	0.96	0.98	0.99	0.99	0.99	0.99

## VI. DISCUSSION

The memory system that we have used throughout this paper (Fig. 4) has 3 places which have additional hardware as compared to a standard interleaved memory: i) the address source, ii) the input/output buffers and iii) the data sequencer. As we have emphasized throughout this paper, the cost of each of these components is not very high and the time penalty introduced by them is minimal. However, let us discuss in more detail some of the issues involved in the design of each one of these components.

The address source has an address generation mechanism and a control mechanism that monitors the input buffers of the banks. Address generation (bank number calculation) is carried out using a set of parity computation circuits (see Fig. 3). The parity computation circuits that we have discussed so far implement a particular PBI scheme, i.e., compute parities based using a given  $A$  matrix. The address generation can be made more flexible by having a *programmable* PBI scheme, i.e., by allowing the  $A$  matrix to be altered under program control. This feature can be useful if some *a priori* knowledge about the access patterns is available (for example, we would like to alter the interleaving to standard interleaving if we know that all the strides of access are odd).

Fortunately, extending the bank-selection hardware of Fig. 3 to allow for arbitrarily programmable schemes is quite straightforward and does not have much additional hardware overhead. All that we have to do is to 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 as shown in Fig. 6. The mask is simply a row of the  $A$  matrix and can be altered under program control. We should point out that the memory must start out in a "clean state" before the  $A$  matrix can be altered, i.e., the memory must be flushed and reloaded with the data if the  $A$  matrix is altered.

The control mechanism for the address source needs to determine the status of the input buffer associated with the selected bank before it can submit the request to the banks for service. It does so by monitoring a busy bit associated with each input buffer. The busy bit is set if all the elements in the buffer are full. In a standard interleaved memory, the busy bit is associated with the memory bank. Because of the precise, pre-determined timing of a standard interleaved memory, it

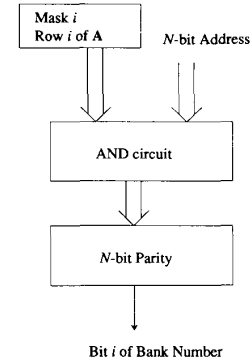


Fig. 6. Bank selection in a programmable interleaved memory.

is possible to incorporate the busy bits associated with the banks in the address source control logic itself. For example, in the Cray-1, the address generation logic determines whether it should generate full-, half- or quarter-speed accesses simply by looking at the stride of the vector access. This is not possible in a PI memory (it is also not possible in a more sophisticated memory system such as the memory system of the Cray X-MP or the Cray Y-MP).

The additional overhead for the buffers includes the latches for the buffer elements, multiplexors and the control logic needed to enforce a queue mechanism. Of course, the width of each buffer element would be increased by the size of the tag needed to hold the sequence number for the data reference.

The data sequencer by far requires the most amount of additional hardware. However, the additional hardware is straightforward. The data sequencer needs to compare its current sequence number with the sequence numbers of the data elements at the head of each output buffer and it also needs an incrementer. Other control logic associated with the data sequencer is trivial.

## VII. RELATED WORK

Since the work reported in this paper was carried out and submitted for publication [21], [22], several papers have appeared that describe related work. Weiss considers an aperiodic scheme, very similar to the PBI schemes presented in this paper [23]. Harper discusses how XOR-based interleaving schemes can be used for dynamic storage schemes to reduce memory conflicts [7]. Rau, Schlansker, and Yen describe the stride-insensitive memory of the Cydra<sup>TM</sup> 5 departmental supercomputer [17]. The interleaving strategy used in the Cydra<sup>TM</sup> 5 (later called pseudo-random interleaved memory, or PRIM, by Rau [18]), is an example of the PBI schemes discussed in this paper. It is worth mentioning that the Cydra<sup>TM</sup> 5 interleaved memory was conceived and built before this work was carried out, however, it was not made public until later.

After the initial experimental work, there has been a fair amount of recent theoretical work that attempts to provide a mathematical foundation for XOR-based interleaving schemes. Raghavan and Hayes consider another interleaving strategy, one that uses multiplicative hashing to randomize the distribution of data amongst the banks, and show that their scheme has

some interesting theoretical properties [16]. Harper attempts to provide a theoretical understanding of the basics of XOR-based interleaving schemes in [9], and provides a metric to evaluate various XOR-based schemes in [8]. Last, Rau provides a very solid theoretical basis for the construction of an XOR-based memory interleaving scheme by showing that schemes that use irreducible polynomials to construct the  $A$  matrix have certain attractive and provable properties [18].

### VIII. SUMMARY AND CONCLUSIONS

In this paper, we discussed the design of high-bandwidth 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 wide variety of strides and for a relatively large bank busy time for a single vector access stream. In order to do so, alternate interleaving schemes that use several address bits to determine the bank number must be used.

This paper discussed permutation-based interleaving schemes and their application to the design of a permutation interleaved memory. Permutation-based interleaving schemes allow the distribution of data in the memory banks in a nonregular fashion so that the probability of a regular pattern of conflicting requests is reduced. By buffering conflicting requests, the throughput of the memory system can be enhanced considerably. The elegance of permutation-based interleaving 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.

We carried out a detailed simulation analysis of an example permutation interleaved memory system and compared it to an equivalent memory system with standard interleaving and a memory system with 1-Skew interleaving. The simulation results indicate that the performance of a permutation interleaved memory system is superior to other memory systems that have been proposed for vector processors. Moreover, permutation interleaved memories can maintain near-peak throughput access as the relative bank busy time is increased. Based upon the results presented in this paper, we believe that permutation interleaved memories can be designed to provide a sustained peak throughput for a wide variety of strides and their use should be considered for high-performance vector processors.

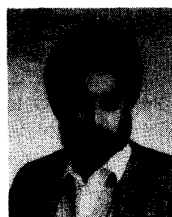
### ACKNOWLEDGMENT

The author is thankful to J. Smith for his discussions throughout this research and to M. Hill for his comments on an earlier draft of this paper.

### REFERENCES

- [1] D. H. Bailey, "Vector computer memory bank contention," *IEEE Trans. Comput.*, vol. C-36, pp. 293–298, Mar. 1987.

- [2] F. A. Briggs and E. S. Davidson, "Organization of semiconductor memories for parallel-pipelined processors," *IEEE Trans. Comput.*, vol. C-26, pp. 162–169, Feb. 1977.
- [3] P. Budnick and D. J. Kuck, "The organization and use of parallel memories," *IEEE Trans. Comput.*, vol. C-20, pp. 1566–1569, Dec. 1971.
- [4] T. Cheung and J. E. Smith, "A simulation study of the CRAY X-MP memory system," *IEEE Trans. Comput.*, vol. C-35, pp. 613–622, July 1986.
- [5] J. M. Frailong, W. Jalby, and J. Lenfant, "XOR-schemes: A flexible data organization in parallel memories," in *Proc. 1985 Int. Conf. Parallel Processing*, Aug. 1985, pp. 276–283.
- [6] 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, pp. 1440–1449, Dec. 1987.
- [7] D. T. Harper III, "Address transformations to increase memory performance," in *Proc. 1989 Int. Conf. Parallel Processing*, St. Charles, IL, Aug. 1989, pp. 1-237–1-241.
- [8] D. T. Harper III and Y. Costa, "Analytical estimation of vector access performance in parallel memory architectures," Tech. Rep. TR-91005, Dep. of EE, Univ. of Texas at Dallas, Dallas, TX, 1991.
- [9] D. T. Harper III, "Increased memory performance during vector access through the use of linear address transformations," *IEEE Trans. Comput.*, vol. 41, pp. 227–230, Feb. 1992.
- [10] D. J. Kuck, "ILLIAC IV software and application programming," *IEEE Trans. Comput.*, vol. C-17, pp. 758–770, Aug. 1968.
- [11] D. J. Kuck and R. A. Stokes, "The Burroughs Scientific Processor (BSP)," *IEEE Trans. Comput.*, vol. C-31, pp. 363–376, May 1982.
- [12] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145–1155, Dec. 1975.
- [13] D. H. Lawrie and C. R. Vora, "The prime memory system for array access," *IEEE Trans. Comput.*, vol. C-31, pp. 435–442, May 1982.
- [14] A. Norton and E. Melton, "A class of Boolean linear transformations for conflict-free power-of-two access," in *Proc. 1987 Int. Conf. Parallel Processing*, Aug. 1987, pp. 247–254.
- [15] W. Oed and O. Lange, "On the effective bandwidth of interleaved memories in vector processor systems," *IEEE Trans. Comput.*, vol. C-34, pp. 949–957, Oct. 1985.
- [16] R. Raghavan and J. P. Hayes, "On randomly interleaved memories," in *Proc. Supercomput. '90*, Nov. 1990, pp. 49–58.
- [17] B. R. Rau, M. S. Schlansker, and D. W. L. Yen, "The Cydra™ stride-insensitive memory system," in *Proc. 1989 Int. Conf. Parallel Processing*, St. Charles, IL, Aug. 1989, pp. 1-242–1-246.
- [18] B. R. Rau, "Pseudo-random interleaved memory," in *Proc. 18th Int. Symp. Comput. Architecture*, Toronto, Canada, May 1991, pp. 74–83.
- [19] R. M. Russel, "The CRAY-1 Computer system," *Commun. ACM*, vol. 21, pp. 63–72, Jan. 1978.
- [20] H. D. Shapiro, "Theoretical limitations on the efficient use of parallel memories," *IEEE Trans. Comput.*, vol. C-27, pp. 412–428, May 1978.
- [21] G. S. Sohi, "Logical data skewing schemes for interleaved memories in vector processors," Computer Sciences Tech. Rep. 753, Univ. of Wisconsin–Madison, Madison, WI 53706, Feb. 1988.
- [22] ———, "High-bandwidth interleaved memories for vector processors—A simulation study," Comput. Sciences Tech. Rep. 790, Univ. of Wisconsin–Madison, Madison, WI 53706, Sept. 1988.
- [23] S. Weiss, "An aperiodic storage scheme to reduce memory conflicts in vector processors," in *Proc. 16th Int. Symp. Comput. Architecture*, Jerusalem, Israel, June 1989, pp. 380–386.
- [24] H. A. G. Wijshoff and J. van Leeuwen, "The structure of periodic storage schemes for parallel memories," *IEEE Trans. Comput.*, vol. C-34, pp. 501–505, June 1985.



**Gurindar Singh Sohi** (S'85–M'85) received his B.E. degree in electrical engineering from the Birla Institute of Technology and Science, Pilani, India, in 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana–Champaign, in 1983 and 1985, respectively.

Since September 1985 he has been with the computer Sciences Department at the University of Wisconsin–Madison, where he is currently an Associate Professor. His research interests are in the area of computer architecture, parallel and distributed processing and fault-tolerant computing.