PSEUDO-RANDOMLY INTERLEAVED MEMORY

B. Ramakrishna Rau

Hewlett Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303

ABSTRACT

Interleaved memories are often used to provide the high bandwidth needed by multiprocessors and high performance uniprocessors. The manner in which memory locations are distributed across the memory modules has a significant influence on whether, and for which types of reference patterns, the full bandwidth of the memory system is achieved. The most common interleaved memory architecture is the sequentially interleaved memory in which successive memory locations are assigned to successive memory modules. Although such an architecture is the simplest to implement and provides good performance with strides that are odd integers, it can degrade badly in the face of even strides, especially strides that are a power of two. This happens because all the memory references are concentrated on a subset of the memory modules.

Pseudo-randomly interleaved memory architectures are ones in which the memory locations are assigned to the memory locations in some pseudo-random fashion in the hope that the sequences of references, that are likely to occur in practice, will end up being evenly distributed across the memory modules. The notion of polynomial interleaving modulo an irreducible polynomial is introduced as a way of achieving pseudo-random interleaving with certain attractive and provable properties. The theory behind this scheme is developed and the results of simulations are presented.

Keywords: supercomputer memory, parallel memory, interleaved memory, hashed memory, pseudo-random interleaving, memory buffering.

1. INTRODUCTION

There has always existed a gap between the cycle time of the processor and that of high-density DRAM memory chips. This performance mismatch appears to be growing and is exacerbated by the use of multiple processors. The conventional solution is to provide a cache memory constructed out of SRAM. However, if one considers multiprocessor systems constructed out of the next generation of processors, which will be capable of issuing multiple memory requests per cycle, it will not be trivial to maintain cache coherency across multiple private caches at such high request rates. The alternative is to use a shared cache. Assuming that the additional delay incurred in going through the processor-cache interconnect is acceptable, we find ourselves in a situation where the bandwidth of even SRAM chips is inadequate unless some form of interleaving is employed in the cache.

Data cache anomalies. In the context of data references, caches are also susceptible to another problem. Many numerical applications (e.g., simulations involving the solution of partial differential equations) often sweep through large data structures in such a way that a particular element is re-referenced only after all of the other elements have been referenced. In many important problems, the arrays tend to be of a size comparable to the physical size of the main memory, and considerably larger than any realistic cache. Consequently, each word is displaced from the cache before it is next referenced, resulting in a low hit rate. The processor is now working directly out of the main memory which, typically, is under-designed for this situation since the design assumption was that only a small fraction of the references would miss the cache.

Worse yet, if the stride with which the processor is referencing memory is equal to or greater than the cache line size, the cache will fetch an entire line for each reference that the processor makes, of which all but one word is wasted. Far from helping the situation, the cache is now compounding the problem by amplifying the request rate to an already under-designed main memory. This phenomenon has been studied and reported elsewhere, e.g., in [1,2]. However, since data caches are important in achieving good performance on "scalar" computations with little parallelism, the right compromise, probably, is to provide a data cache that can be bypassed when referencing data structures which have poor locality.

Interleaved memory systems. Thus, whether or not a data cache is employed, it can be important to provide a memory system that can satisfy the request rate of the processors.

The solution generally used is to organize the memory system as multiple memory modules which can operate in parallel. The manner in which memory locations are distributed across the memory modules has a significant influence on whether, and for which types of reference patterns, the full bandwidth of the memory system is achieved.

Most engineering and scientific applications include computations such as matrix operations (on both dense and sparse matrices), single- and multi-dimensional fast Fourier transforms (FFT), interpolation and table lookup. As a result, although vector accesses with unit stride are important, a number of more complex access patterns must also be handled well. Dense matrix operations in FORTRAN that access two-dimensional matrices by columns, diagonals or anti-diagonals lead to accesses with a constant but non-unit stride. Sparse matrix, interpolation and table lookup operations make array accesses using subscripts that are either the result of some non-trivial computation or are derived by accessing another array of indices. The result is access sequences that are irregular and apparently random. The FFT algorithm's bit-reversed addressing pattern has structure to it, but it is by no means a constant stride. The situation becomes more complex yet when one considers multiple such access streams proceeding simultaneously. The relative strides of the various access streams and, if they have the same stride, their offsets relative to one another can have a marked effect on the memory system's performance [3,4].

Simple schemes like low-order interleaving, where the memory is structured so that, with M memory modules, the location with address a is in memory module (a mod M), are fine (in fact optimal) if the memory references are sequential. In this case, the processors' references will be uniformly distributed over all the memory modules, thereby allowing them all to operate in parallel and match the bandwidth requirements imposed by the processors. However, such a "sequentially interleaved" memory is subject to dramatic performance degradation when the memory references have certain patterns, for instance, if the references have a stride which is a multiple of M. In this case, all the references are directed to the same memory module and the performance is as bad as it could possibly be. Memory architectures that are optimized for unit-stride vector accesses rely heavily on the programmer's ability to develop algorithms which generate only such accesses. When more complex accesses do occur, memory performance degrades, sometimes very significantly.

In a "pseudo-randomly interleaved" memory system, the mapping between the address, a, and the memory module is pseudo-random. Such a memory architecture, if properly designed, is more robust than the sequentially-interleaved type and is relatively insensitive to the address reference pattern. However, what is not immediately clear is how one designs, evaluates and selects a good pseudo-randomization scheme. In this paper, use is made of the mathematical theory of Galois fields permitting the design of fairly robust pseudo-random interleaving schemes which are, at the same time, random as well as predictable in their behavior.

In Section 2, we look at conventional interleaving schemes such as sequential interleaving and prime degree interleaving, and study their shortcomings. In Section 3, we introduce the idea of pseudo-random interleaving with particular emphasis on XOR-based permutation schemes. Here, we also derive some desirable features that such a scheme should provide. In Section 4, we introduce and develop the concept of polynomial interleaving, establish its connections to XOR-based permutation schemes, and state a number of theorems regarding polynomial interleaving. Section 5 presents the results of some simulation runs that were undertaken to characterize the behavior of polynomial interleaving schemes. In Section 6, we outline the procedure for designing an irreducible polynomial interleaving scheme. The proofs of the theorems in Section 4 are provided in the Appendix.

2. CONVENTIONAL INTERLEAVING SCHEMES

We shall consider interleaved memory systems with M memory modules. By and large, M is a power of 2, and in that case $M = 2^m$. Let the module index, corresponding to a particular memory location, be defined to be the integer between 0 and M-1 which specifies the module in which the location is to be found, and the word address is the address of that location within the module.

Although the memory architectures discussed in this paper have applicability to multiprocessors consisting of more complex processors, we shall restrict our discussion to the case of a single processor that is capable of making one reqnest on every processor cycle. Figure 1 shows the type of processor-memory system that is considered in this paper. Furthermore, we shall restrict our consideration, in this paper, to either a reference sequence that consists of accesses that are randomly directed to the M memory modules or to one that constitutes an arithmetic sequence with some fixed stride that is not necessarily 1.

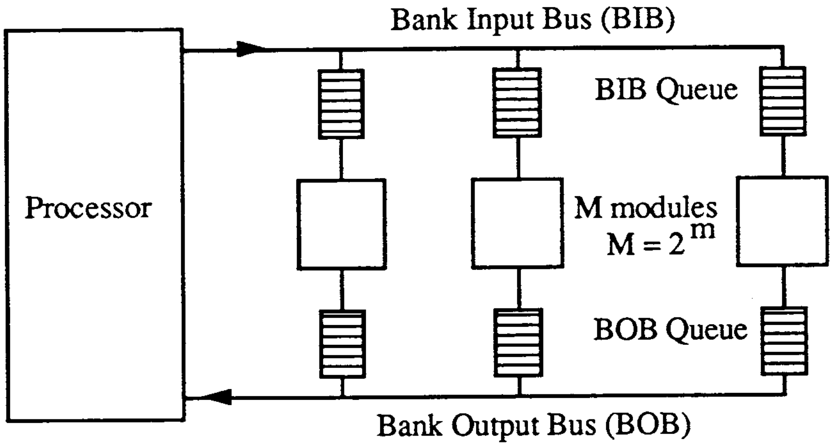
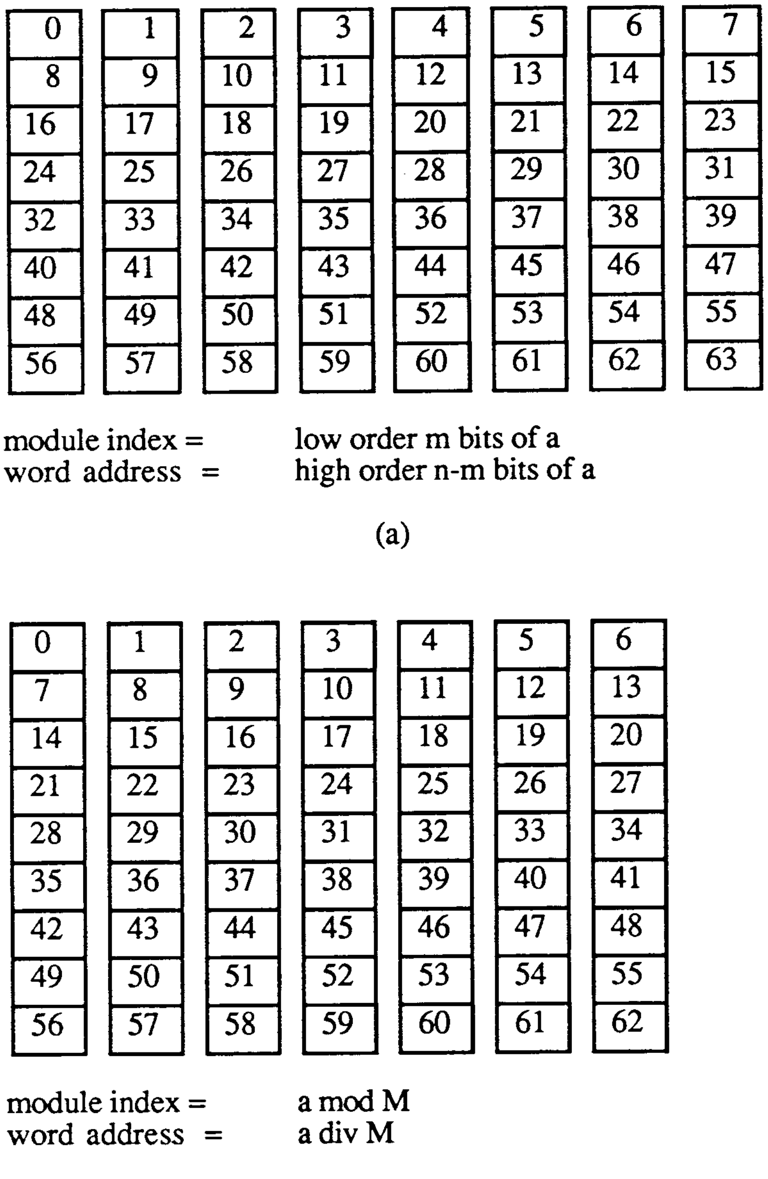


Figure 1. Structure of the processor-memory system that is considered in this paper.

We shall use the processor cycle time as the unit of measure and, for the purposes of this paper, we shall make the (generally true) assumption that the cycle time of the RAM chip has been rounded up to a multiple of the processor cycle time. The ratio between these two cycle times is defined to be the memory cycle time. Also, we shall assume that the buses between the processors and the memory are capable of transmitting one request and/or datum every cycle. The average number of requests per cycle that the processor-memory combination are able to actually sustain will be termed the achieved bandwidth or, more simply, the bandwidth. If the processor is attempting to make a memory request every cycle, the achieved bandwidth is also equal to the processor utilization, which is the fraction of time that the processor is not stalled.

Sequentially interleaved memory (SIM) architectures. The most common style of interleaved memory architecture is a sequentially interleaved memory (Figure 2a), consisting of $M = 2^m$ modules, in which the location whose address is a, has a module index of (a mod M) and a word address of (a div M). In practice, no division is required; since M is a power of 2, the module index is the low order m bits of the address and the word address is the remaining high order bits. In the case of a sequential reference stream, this ensures high bandwidth since all the modules are referenced before the same module is referenced again. If the degree of interleaving is at least as large as the memory cycle time, the memory module will be ready to handle another request by the time it is referenced again. In this case, the memory system is able to accept one request every cycle and to keep up with the processor. On the other hand, if the reference sequence has a stride which is a multiple of M, every reference is to the same memory module and we get no benefit from the interleaving.



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

module index = low order m bits of a
word address = high order n-m bits of a

(a)

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

module index = a mod M
word address = a div M

(b)

Figure 2. Assignment of memory locations to memory modules with (a) sequential 8-way interleaving and (b) prime degree 7-way interleaving

Prime degree interleaving. In general, the achieved bandwidth, when the reference sequence has a stride of s, is given by M/gcd(M,s), where gcd stands for the greatest common divisor. Whenever s is not relatively prime to M, the bandwidth is degraded. This motivates the use of prime degree interleaving (Figure 2b) in which the number of memory modules, M, is a prime number [5]. Making M prime maximizes the number of strides that are relatively prime to M. Except for strides which are a multiple of M (in which case bandwidth degrades by a factor of M), peak bandwidth is consistently achieved. Unfortunately, the computation of the module index and the word address are no longer trivial since they involve true division, although the judicious choice of the prime number (e.g., of the form $M = 2^n \pm 1$) can simplify the computation somewhat.

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

word address, r = a div 8
skew = r mod 8
module index = (a + skew) mod 8

Figure 3. A skewed-storage 8-way interleaving scheme

Skewed-storage schemes. The problem with the above two approaches to interleaving is that, due to the regular pattern with which memory locations are assigned to memory modules, it is easy to find plausible sequences of references, all of which map to the same module. To address this problem, skewed-storage schemes have been suggested in which each successive set of M memory locations is assigned to the M memory modules with a skew relative to the previous set, e.g., [6-8]. One example of such a scheme (Figure 3) is to compute the module index for location a as ((a + ((a div M) mod M)) mod M). The word address is (a div M) as before. Thus, while locations 0 through M-1 are assigned to modules 0 through M-1, respectively, locations M through 2M-1 are assigned with a skew of 1 to modules 1 through M-1 and then 0. Strides that are a multiple of M-1 still suffer since sets of M consecutive references will be to the same memory module. Furthermore, sequences with strides that are a multiple of M^2 will still all map to the same module. However, this basic idea, whereby each set of M consecutive locations is permuted across the modules in a different fashion, is valuable and is employed in the pseudo-random interleaving schemes that are developed in Sections 3 and 4.

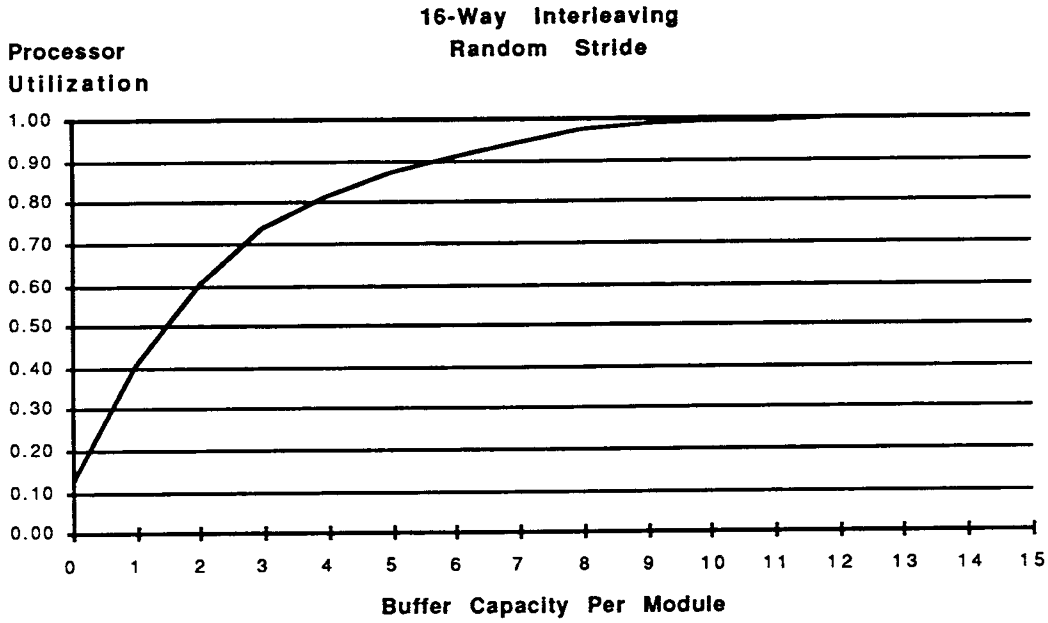


Figure 4. Performance of a sequentially interleaved memory, with buffering per module, for a random reference stream.

Buffering. None of these interleaving schemes, by themselves, can guarantee high bandwidth when the reference sequence has a random or irregular pattern of accesses to the memory modules. In the case of the unit-stride vector access, every M-th request is to the same module. Thus, the time between successive references to a given module is a constant M cycles. In contrast, successive references to a given module are irregularly spaced in time for random access patterns. Every so often, the processor will reference a module that is still busy and, in the absence of buffering, the processor must stall until the module is available. An M-way interleaved memory with a random request sequence will only achieve a bandwidth that is approximately proportional to $\sqrt{M}$ modules instead of getting the full benefit of the M modules [9]. However, if the memory system has facilities to queue up references to busy modules, the processor need not stall. With sufficient buffering, the full bandwidth of M modules can be achieved [10,11]. Figure 4 shows the result of a simulation, for the system in Figure 1 with 16-way interleaving, which confirms that this is, indeed, the case. Adequate buffering at points of contention for the same resource, such as a memory module or a bus, is needed for full bandwidth with random access patterns.

Conversely, buffering alone is not enough if the reference stream has a stride that is not relatively prime to M. Since only a subset of the modules are being referenced, references to any given module (which is being referenced) arrive at an average rate that is a multiple of 1 every M cycles, causing the buffers to fill up almost immediately. Thereafter, the processor will not be able to make a reference every cycle.

3. PSEUDO-RANDOM INTERLEAVING

As we have seen, any assignment of locations to modules with an obvious pattern is suspect. This suggests the assignment of memory locations to modules in a pseudo-random fashion in the hope that no non-artificial sequence of references will exhibit more than a very short-term concentration to an individual module. We shall refer to an architecture of this kind as a pseudo-randomly interleaved memory (PRIM). By providing adequate buffering to queue the short clusters, the full interleaved bandwidth would be achieved.

For the purposes of this discussion, the term physical address is used to refer to the address presented to the memory system after virtual address translation but prior to randomization and the term randomized address is used to refer to the address after the randomizing mapping. The physical address is represented by $A = <a_{n-1}, .., a_0>$ and the randomized address by $B = <b_{n-1}, .., b_0>$. (The bits that determine the byte address within a word are not relevant to this discussion). Let A_H and A_L refer to $<a_{n-1}, .., a_1, a_m>$ and $<a_{m-1}, .., a_1, a_0>$, (i.e., the high-order and low-order bits), respectively. Let B_H and B_L be similarly defined for $<b_{n-1}, .., b_1, b_0>$. In an M-way interleaved memory (where $M = 2^m$), the low-order m bits of the randomized address, B_L, determine the module index. The remaining high-order bits, B_H, determine the word address within the selected module. The randomizing function h(•) maps a to b, i.e., B = h(A).

The randomizing function must, obviously, possess the following properties to be even minimally acceptable:

Property 1. The mapping should be repeatable, i.e., the same physical address should always map to the same randomized address. In other words, the mapping cannot be truly random. Rather, it is deterministic and pseudo-random.

Property 2. The mapping should be one-to-one, i.e., no more than one physical address should map to the same randomized address.

However, for the randomization scheme to be desirable, it should have the following additional properties:

Property 3. The mapping should be onto, i.e., some physical address will map into any given randomized address. This avoids wastage resulting from memory that is physically present but which cannot be addressed.

Property 4. There should never be a greater concentration of references in time to any given memory module than would be the case with a truly random sequence of randomized addresses, regardless of the physical address sequence that is presented to the memory system. Amongst other things, this property implies that as many as possible, preferably all, of the physical address bits should be used in determining the module address. One might expect then that regardless of where in the physical address word the "activity" is (in terms of the address bits changing), the randomized module address will continue to change, thereby minimizing clustering.

Permutation schemes. Another property yields some simplification. There is no benefit derived from the randomizing function modifying $<a_{n-1}, .., a_m>$ since these address bits do not determine the module selected but merely permute the memory locations within the same module. Hardware cost may be avoided if the randomizing function does not alter these bits. However, if this is the case, an additional property must exist.

Property 5. When $<a_{m-1}, .., a_0>$ go through all 2^m combinations holding $<a_{n-1}, .., a_m>$ constant, $<b_{m-1}, .., b_0>$ should also go through all 2^m combinations, i.e., the randomization scheme must apply a permutation to the 2^m addresses.

This is a necessary condition to guarantee the one-to-one property (Property 2 above). However, to supply Property 4, for each combination of $<a_{n-1}, .., a_m>$, the permutation should be different.

Permutation using the XOR function. It is clearly preferable if the computation of the randomized address is inexpensive both in the amount of hardware required as well as in the time taken to do it. From this viewpoint, the idea of randomizing the physical address by XOR-ing it with another bit pattern is very attractive [2,12-14]. This mapping is a permutation and, so, satisfies requirements 2 and 3. Obviously, the bit pattern that is XOR-ed with the physical address must keep changing, else all that we have accomplished is a renaming of the memory modules.

Assume that the m low order bits, A_L, of the physical address are to be randomized to yield the low order m bits, B_L, of the randomized address, where $n \geq m$ and n is the number of physical address bits. Due to the associativity and commutativity of the XOR function, any randomization scheme, that is based solely on the XOR function, can be viewed, with no loss of generality, as being implemented using a set of m multiple-input XOR gates whose outputs constitute $<b_{m-1}, .., b_0>$. The inputs to each XOR gate are some subset of $<a_{n-1}, .., a_0>$. The randomization scheme is completely specified by a boolean matrix H[i,j], (i = 0,..,n-1, and j = 0,..,m-1), where H[i,j] = 1 if and only if a_{m-i-1} is an input to the XOR gate whose output is h_j.

$$\begin{bmatrix} H(n-1,m-1) & \cdots\cdots & H(n-1,0) \\ \vdots & & \vdots \\ H(0,m-1) & \cdots\cdots & H(0,0) \end{bmatrix}$$

A_H is unaltered to yield B_H. If $<b_{m-1}, .., b_0>$ is viewed as a vector, then it is the result of the vector-matrix product $<a_{n-1}, .., a_0> * H$ (where multiplication and addition are to be done modulo 2 and are equivalent to the AND and XOR functions, respectively).

For this randomization scheme to satisfy requirements 2 and 3, when $<a_{m-1}, .., a_0>$ go through all 2^m combinations holding $<a_{n-1}, .., a_m>$ constant, $<b_{m-1}, .., b_0>$ should also go through all 2^m combinations, i.e., the randomization scheme should be a permutation. Define S(q,i) to be the following square sub-matrix of H:

$$\begin{bmatrix} H(i+q-1,q-1) & \cdots & H(i+q-1,0) \\ \vdots & & \vdots \\ H(i,q-1) & \cdots & H(i,0) \end{bmatrix} .$$

where $1 \leq q \leq m$ and $0 \leq i \leq n-q$. Properties 2 and 3 are provided if S(m,0) is non-singular, i.e., the bottom m rows of the H-matrix are linearly independent. For any fixed value of A_H, either A_L or B_L can each be computed uniquely from the other. Thus B_L is a permutation of A_L. However, for each A_H, the permutation is different if the remaining rows of the H-matrix are suitably chosen. This approach can provide property 4 if H is well designed. Factors to be considered when designing the H-matrix are discussed below.

Any XOR-based randomization function that adheres to the above rules is acceptable. Nevertheless, these rules by no means define a unique H-matrix and certain H-matrices are better than others. The purpose of an interleaved memory is to increase the bandwidth of the memory system by directing successive references to distinct memory modules, thereby achieving the bandwidth of multiple memories operating in parallel. The success of the interleaving scheme is measured by the extent to which "long" sequences of "clustered" (i.e., closely spaced in time) requests to the same memory module are avoided. The better an H-matrix is, the less likely is it that some particular stride will result in long clusters of references to the same module.

One of the benefits of requiring that the first m rows of the H-matrix be linearly independent is that clustering is greatly reduced for the unit stride (which is the single most important stride). Since each set of 2^m consecutive references is uniformly distributed across the 2^m modules, the clustering that occurs is less than that which would occur with a truly random sequence. It is desirable that this benefit be extended to other strides as well. Good behavior for strides that are a power of 2 is ensured by requiring that S(m,i) be non-singular for all i, $1 \leq i \leq n-m$.

However, this does not yet guarantee a good H-matrix. An example of an H-matrix, every S(m,i) of which is non-singular, but which, nevertheless, is a poor design, is shown in Figure 5b. The rows repeat themselves with some relatively small period k (=4). If one considers the first 2^k addresses in a request sequence with stride 2^k+1 starting at 0, the 1's in the address come in pairs that are k bit positions apart (Figure 5a). This means that identical rows that are k apart in the H-matrix are being XOR-ed together and cancelling each other out. Clusters of upto 2^k consecutive references will be to the same module (Figure 5c).

(a)	(b)	(c)
0000	0001	0000
10001	1000	0000
100010	0100	0000
110011	0010	0000
1000100	0001	0000
1010101	1000	0000
1100110	0100	0000
1110111	0010	0000
:	0001	:
:	1000	:
11111111	0100	0000
100010000	0010	0000
100100001	0001	0010
	1000	
	0100	
	0010	
	0001	

Figure 5. (a) A physical address sequence with a stride of 17. (b) An H-matrix with repetitive rows. (c) The output sequence of addresses; long sequences of addresses get mapped to the same module.

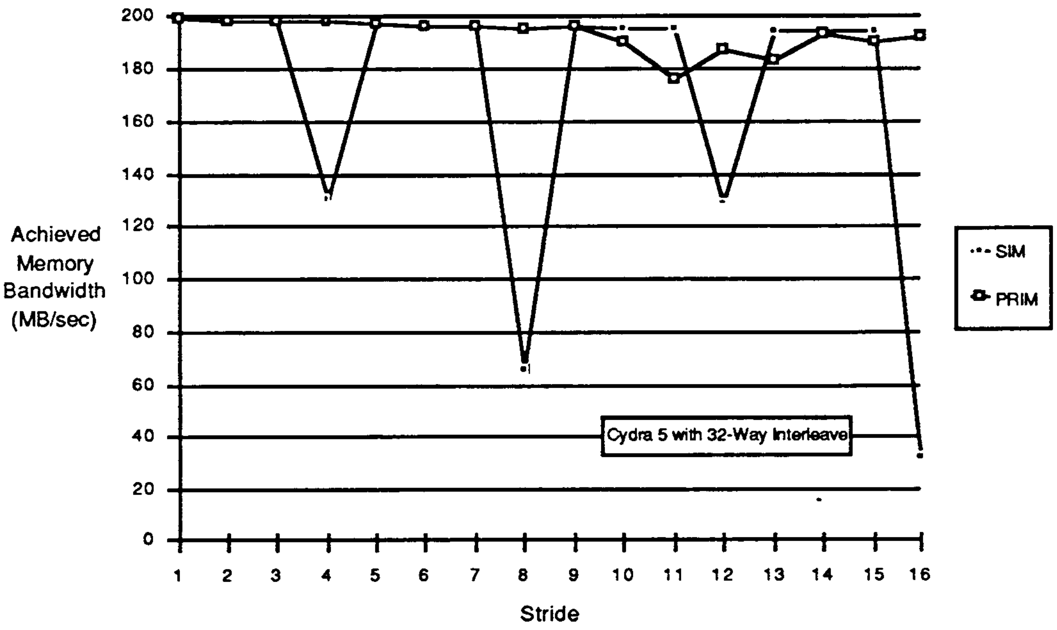


Figure 6. Performance of the Cydra 5 pseudo-randomly interleaved memory system with and without randomization using a random H-matrix.

To avoid such problems, and in the absence then of any better theory at that time, the H-matrix for the Cydra 5 [2], in which m was equal to 6, was designed by selecting the rows randomly, without replacement, from the set of 2^6-1 bit patterns. (The all-zero row was excluded). The non-singularity of S(q,0), for $2 \leq q \leq 6$, was obtained by perturbing this random ordering of rows to the minimum extent necessary. Although this procedure was rather ad hoc, the randomizing function implemented in the Cydra 5 works quite well as can be seen in Figure 6. More extensive measurements on the Cydra 5 memory system, that demonstrate the robustness of pseudo-randomly interleaved memory, are reported in [2].

4. IRREDUCIBLE POLYNOMIAL INTERLEAVING

The unsatisfactory aspect of the ad hoc design of the Cydra 5's H-matrix is that it is very difficult, if not impossible, to develop any theory that predicts performance as a function of stride. (For instance, it is difficult to explain why there is a dip in performance for strides in the range of 11 through 13 in Figure 6). Obviously, the brute-force approach of measuring its behaviour for every stride is impractical. Before describing an interleaving strategy that allows such prediction, let us summarize the lessons learned from the previously described interleaving schemes. Prime degree interleaving is very effective except for strides that are a multiple of M but the required division makes it unattractive. Skewed-storage schemes, which are one example of permutation schemes, partially reduce the sensitivity to bad strides, but are less attractive from an implementation viewpoint than are XOR-based permutation schemes. XOR-based permutation schemes are simple to implement but lack enough underlying theory to assist in their design. In contrast, with sequential and prime degree interleaving it is quite straightforward to specify the performance for any stride. It would be nice to be able to combine such predictability with the relative stride insensitivity of permutation schemes and the implementation attractiveness of the H-matrix. An approach that comes close to meeting these goals is developed next.

Consider the class of polynomials whose coefficients are in the Galois Field GF(2) [15], i.e., the coefficients take on the values 0 or 1 and addition, subtraction and multiplication are performed modulo 2. (Note that this makes addition and subtraction the equivalent of the XOR operation, and multiplication the equivalent of the AND operation). Such polynomials are said to be defined over the field GF(2) and the addition, subtraction, multiplication and division of such polynomials is similar to that for conventional polynomials except that the coefficient arithmetic is determined by that for GF(2). (Note also that the implementation of arithmetic with these polynomials is the same as that for binary numbers except that there is no carry or borrow). An irreducible polynomial is a polynomial over GF(2) that is divisible by no other polynomial over GF(2) which is of order greater than 0. As a matter of convention, we shall refer to the integer, obtained by setting x = 2 in the polynomial A(x), as A, and vice versa, i.e., if $A = <a_{n-1}, .., a_1, a_0>$, then $A(x) = a_{n-1}x^{n-1}+ ..+a_1x+a_0$. When no confusion can arise, we shall resort to the somewhat sloppy, but convenient, practice of referring to a polynomial A(x) by its associated integer A (e.g., polynomial 19 is x^4+x+1) and ascribing to polynomials the properties of their associated integers (e.g., an even polynomial is one in which the coefficient of x^0 is 0).

Let P(x) be a polynomial of order m, and let A(x) be the polynomial of order n that is associated with the address, a, of a memory location. Then A(x) can be uniquely represented [15] as

$$A(x) = V(x)*P(x) + R(x)$$

where V(x) and R(x) are polynomials over GF(2) and R(x) is of order less than m. With the polynomial interleaving scheme defined by P(x) for M (= 2^m) memory modules, the integer r is used as the module index. (We shall, shortly, present an inexpensive technique for computing r). The integer v could be used as the word address within the module. However, this would require the unnecessary computation of the polynomial quotient. Instead, we choose to use, as the word address, the integer q defined by the polynomial Q(x), where

$$A(x) = Q(x)*x^m + R'(x),$$

i.e., the word address is merely the high order bits of the physical address a. Thus, the randomized address b that is the result of applying the randomizing function to the physical address a is given by the integer associated with the polynomial

$$B(x) = Q(x)*x^m + R(x).$$

Thus polynomial interleaving is analogous to conventional interleaving except that we use polynomial arithmetic modulo a polynomial rather than integer arithmetic modulo an integer to compute the module index. For many of the same reasons that it is attractive to choose a prime as the modulus integer, we shall find it desirable to choose an irreducible polynomial as the modulus polynomial.

Computation of R(x). Let $R_i(x) = x^i \bmod P(x)$. Since $A(x) = a_{n-1}x^{n-1}+ ..+ a_1x+ a_0$,

$$\begin{aligned} R(x) = A(x) \bmod P(x) &= [R_{n-1}(x)+..+R_1(x)+R_0(x)] \bmod P(x) \\ &= R_{n-1}(x)+..+R_1(x)+R_0(x) \end{aligned}$$

since each $R_i(x)$ is of order less than m and, consequently, so is the sum of all the $R_i(x)$. Thus, if the $R_i(x)$ are pre-computed, R(x) can be computed by adding up those $R_i(x)$ for which the corresponding a_i is 1. This is equivalent to using an H-matrix in which the i-th row (from the bottom) consists of the coefficients of $R_i(x)$.

It is interesting to note that the rows of such an H-matrix constitute the successive states of a feedback shift register. Since $x^i = x^{i-1}*x$, for $i \geq 1$, $R_i(x) = (R_{i-1}(x)*x) \bmod P(x)$. If the coefficient of x^{m-1} in $R_{i-1}(x)$ is 0, then $R_i(x)$ is of order less than m and $R_i(x) = R_{i-1}(x)*x$, i.e., the contents of the i-th row (from the bottom) are obtained by shifting the contents of the (i-1)-th row to the left. If, however, the coefficient of x^{m-1} in $R_{i-1}(x)$ is 1, then $R_i(x)$ is of order m and $R_i(x) = (R_{i-1}(x)-x^{m-1})*x + (x^{m-1} \bmod P(x))$, i.e., the contents of the i-th row are obtained by shifting the contents of the (i-1)-th row to the left, ignoring the bit that shifts out to the left, and then XOR-ing in the coefficients of the polynomial $P(x)-x^{m-1}$.

Properties of polynomial interleaving. The proofs for the theorems in this Section are provided in the Appendix.

Theorem 1. Let P(x) be a polynomial of order m over GF(2). The polynomial interleaving scheme in which the physical address $A = <a_{n-1}, .., a_1, a_0>$ is mapped into the randomized address $B = <b_{n-1}, .., b_1, b_0>$, where

$$B(x) = (A(x) \text{ div } x^m)*x^m + (A(x) \bmod P(x)),$$

is a permutation scheme ■

Figure 7 shows the manner in which memory locations are assigned to memory modules with the polynomial interleaving defined by the polynomial 19 for m = 4. Note that each set of 16 memory locations is permuted in a different way.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
19	18	17	16	23	22	21	20	27	26	25	24	31	30	29	28
38	39	36	37	34	35	32	33	46	47	44	45	42	43	40	41
53	52	55	54	49	48	51	50	61	60	63	62	57	56	59	58
76	77	78	79	72	73	74	75	68	69	70	71	64	65	66	67
95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
106	107	104	105	110	111	108	109	98	99	96	97	102	103	100	101
121	120	123	122	125	124	127	126	113	112	115	114	117	116	119	118
139	138	137	136	143	142	141	140	131	130	129	128	135	134	133	132
152	153	154	155	156	157	158	159	144	145	146	147	148	149	150	151

Figure 7. Assignment of memory locations to memory modules in the polynomial interleaving scheme defined by the polynomial 19.

Theorem 2. Consider the H-matrix corresponding to a polynomial interleaving scheme defined by the polynomial P(x). If P(x) is odd (i.e., the associated integer P is odd), then all of the square sub-matrices, S(m,i), of the H-matrix are non-singular and all strides of the form $s = s_0 * 2^k$ for any $k \geq 0$ are statistically identical in their behavior to that of the stride s_0 ■

As a result of this theorem, we need only examine odd strides to fully understand the behavior of a polynomial interleaving scheme that is defined by an odd modulus polynomial.

Definition: The references of a reference sequence $\{a_0, a_1, ..\}$ are said to be short-term equi-distributed over the M memory modules if it is possible to define an integer k such that the references in any sub-sequence $\{a_{k+i*M}, ..., a_{k+(i+1)M-1}\}$, for $i \geq 0$, are all to distinct memory modules.

Theorem 3. In the polynomial interleaving scheme defined by an odd polynomial P(x), all strides that are of the form 2^k are short-term equi-distributed ■

Definition: Consider the reference sequence $\{a_0,a_1,..,a_K\}$. Let K_i be the number of references in the reference sequence to memory module i. The reference sequence is said to be long-term equi-distributed over the M memory modules if K_i/K tends to 1/M as K tends to ∞.

Hypothesis. In the polynomial interleaving scheme defined by an odd polynomial P(x), all odd strides are long-term equi-distributed ■

All the experiments that have been conducted to date confirm the above hypothesis and it is believed to be true although a proof has not presented itself thus far. If this hypothesis is correct, then by Theorem 2, all even strides, too, are long-term equi-distributed. However, even if the hypothesis is true, it does not preclude the possibility of extensive clustering of references to memory modules, i.e., the absence of short-term equi-distribution. Analogous to conventional interleaving, we would expect to see marked clustering if a reference sequence $\{A_0,A_1,..,A_K\}$ were such that all the polynomials $A_i(x)$ modulo P(x), $0 \le i \le K$, mapped into a subset of the M modules.

Theorem 4. With the 2^m-way polynomial interleaving scheme defined by a polynomial P(x) and with a reference sequence $\{A_0,A_1,..,A_K\}$ such that the greatest common divisor of P(x) and $A_i(x)$, for all $0 \le i \le K$, is the polynomial G(x) of order q, only 2^{m-q} memory modules are referenced over the whole reference sequence ■

For such a situation to arise, given that $\{A_0,A_1,..,A_K\}$ constitutes an arithmetic sequence with stride s, would require that $A_i(x) = S(x)*i(x)$, $0 \le i \le K$, where i(x) is the polynomial associated with the integer i. One way in which this can occur is if the binary representation of s consists of sparse 1's, i.e., 1's separated by relatively long sequences of 0's. Let k be the shortest run length in s of 0's between two consecutive 1's. We shall term a stride of this type a k-sparse stride. With a k-sparse stride, for all i, $0 \le i \le 2^k$, $A_i(x) = S(x)*i(x)$. This is because, over the stated range of i, the partial binary products, when added, do not generate any carry. Thus binary and polynomial arithmetic are identical over this range. By the previous theorem, the performance of the polynomial interleaving scheme over the sequence $\{A_0,A_1,..,A_K\}$, where $K = 2^k$, is determined by the order of G(x), the greatest common divisor of S(x) and P(x). In particular, if S(x) is a multiple of P(x), G(x) = P(x), the order of G(x) = m, and only a single module is referenced over the sequence $\{A_0,A_1,..,A_K\}$. The possibility that n, the order of G(x), is greater than 0 but less than m is eliminated if P(x) is an irreducible polynomial, thus minimizing the opportunity for strides that concentrate their references over significant periods of time to a subset of the M memory modules. On the positive side, if a k-sparse stride polynomial is relatively prime to P(x), the reference sequence $\{A_0, A_1, .., A_K\}$, where $K = 2^k$, is short-term equi-distributed.

Definition: An element, ß, of the field of polynomials modulo the irreducible polynomial P(x) is said to be a primitive element or a generator of the field if for each element, μ, of the field, where $\mu \neq 0$, $\mu = ß^k$, for some k such that $0 \leq k \leq 2^m - 1$.

Definition: A polynomial interleaving scheme defined by the polynomial P(x) such that P(x) is irreducible and x is a primitive element is termed an irreducible polynomial (I-poly) interleaving scheme.

In cases where P(x) is irreducible but x is not a primitive element, all the benefits of I-poly interleaving can be achieved by using an H-matrix whose i-th row (from the bottom) is given by ∂^i mod P(x), where ∂ is a primitive element. However, for the sake of brevity, we shall not discuss this possibility any further.

Theorem 5. In an I-poly interleaving scheme defined by the irreducible polynomial P(x) of order m, x^k+1 is not divisible by P(x) for any $k < 2^m-1$ and is divisible by P(x) for $k = 2^m-1$, i.e., the rows of the H-matrix have a maximal period of 2^m-1 ■

Note that the successive states of a feedback shift register and, thus, the rows of the H-matrix, constitute a pseudo-random sequence which is of maximal period if P(x) is irreducible and x is a primitive element. Randomness in the rows of the H-matrix was called out as a desirable property in the discussion at the end of Section 3. Pseudo-randomness of this sort provides most of the benefits of randomness but also makes it possible to prove certain properties of the interleaving scheme which would be difficult or impossible with true randomness.

5. MEASUREMENTS

Measurements were performed by simulating a processor-memory system of the form shown in Figure 1 with one processor and a 16-way interleaved memory. Each simulation was run for 16,384 cycles with the processor making a memory reference every cycle. The reference sequence consisted either of an arithmetic sequence with some constant stride or a random sequence.

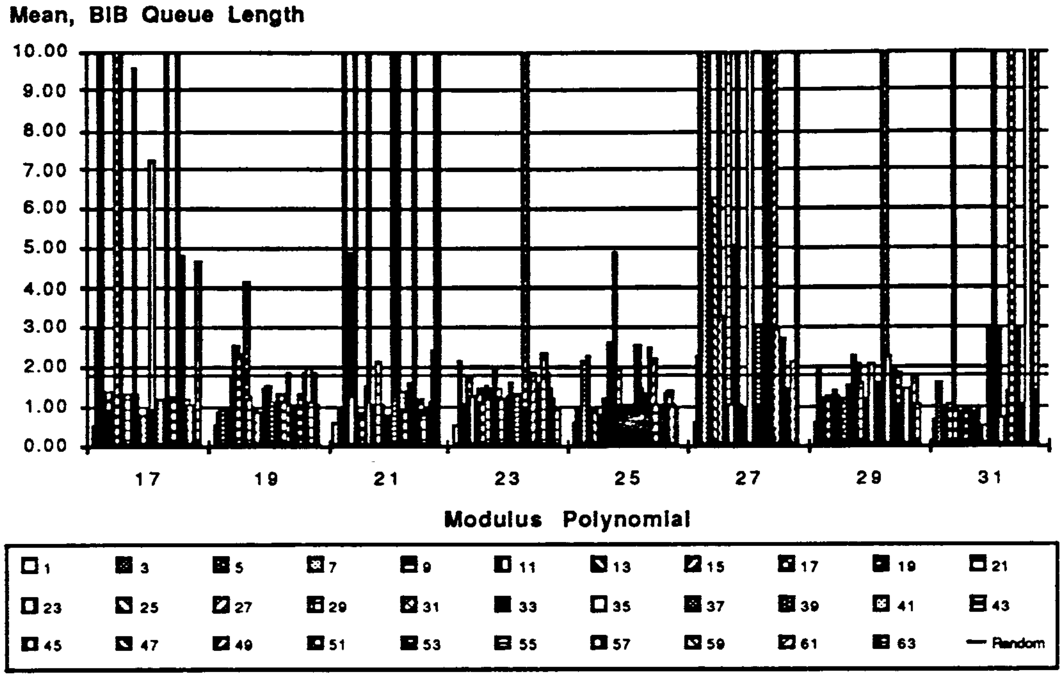


Figure 8. Average BIB Queue length (assuming unbounded buffer capacity per memory module) for all odd strides between 1 and 63 and all odd fourth-order modulus polynomials.

The first set of simulations were performed assuming unbounded BIB (Bank Input Bus) queues and a memory cycle time of 16 cycles. Thus, with the processor making a request every cycle, the memory system was saturated. This was done so as to exaggerate and thereby highlight the effects of the temporal clustering of references. Figure 8 shows the average BIB Queue length for all the odd fourth-order polynomials and for all the odd strides between 1 and 63. Notice the relatively short queue lengths for the irreducible polynomials 19 and 25. The other irreducible polynomial, 31, does not behave as well because x is not a primitive element when p = 31.

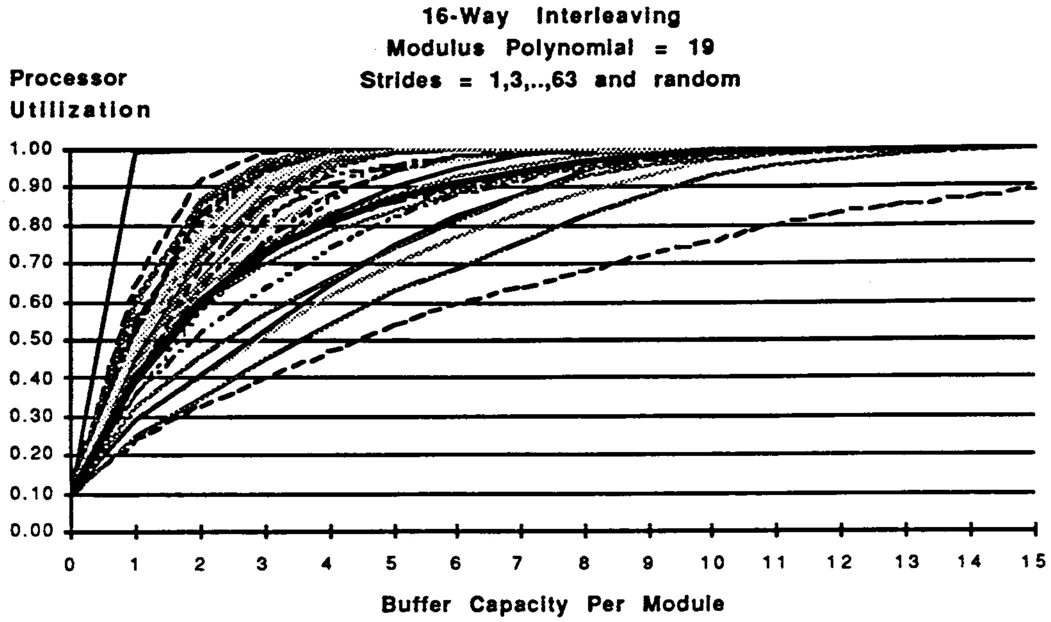


Figure 9. Processor utilization as a function of the amount of buffer capacity per memory module for odd strides between 1 and 63 as well as for the random stride (heavy line).

The second set of simulations were performed for a more realistic design in which the memory cycle time was selected to be 12 (less than the degree of interleaving) since it is well understood that it is not advisable to operate any queueing system at, or close to, saturation. The objective of this set of simulations was to see how often the processor would be stalled due to the BIB Queues getting full in the case when the BIB Queue capacities are finite. Figure 9 plots the processor utilization as a function of the BIB Queue capacity for all the odd strides between 1 and 63 and for the random stride. Notice that only six or seven strides out of thirty-two behave worse than the random stride (the heavy line). This demonstrates the fact that a permutation scheme, by and large, behaves better than a truely random inteleaving scheme. When the buffer capacity is around four to six, the majority of the strides yield a processor utilization that is better than 80%. Notice, also, that for stride 1 (the top curve), perfect processor utilization is achieved as long as there is the ability to buffer one request (the one currently being served) per memory module.

Figure 10 compares sequential interleaving (SIM) with the I-poly interleaving scheme defined by the modulus polynomial 19 for BIB Queue capacities of 4, 8 and 12 and for strides from 1 through 64. The plots show the cumulative fraction of the 64 strides that yield a processor utilization that is less than or equal to a given value. Notice that even with a buffer capacity of 4 per module, the worst stride with I-poly interleaving is better than a quarter of all the strides with SIM and that with a buffer capacity of 8 per module, the worst stride with I-poly interleaving is better than half of all the strides with SIM. With a buffer capacity of 8 or more, almost all strides yield better than 80%. This demonstrates the robustness of I-poly interleaving and its advantage over conventional sequential interleaving.

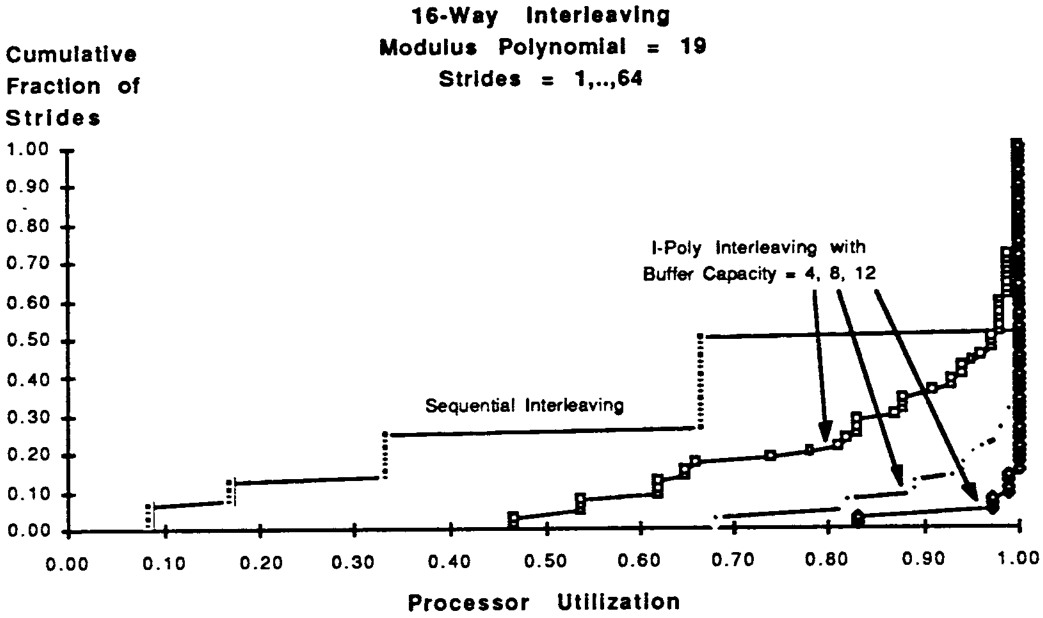


Figure 10. Processor utilization as a function of the stride with sequential interleaving and I-poly interleaving for various amounts of buffer capacity per memory module

6. CONCLUSION

I-poly interleaving, i.e., polynomial interleaving modulo an irreducible polynomial, can provide most of the properties that one might desire in an interleaving scheme: implementation simplicity, robustness in the face of varying address patterns, and the ability to prove important properties regarding its behavior. The major hardware cost associated with I-poly interleaving is the buffering that is required per memory module to achieve good performance. However, it could be argued that this buffering is required with other interleaving schemes as well if random or irregular access patterns are to be accomodated. The major performance penalty is that as a result of the pseudo-randomization, in cases where the processor is making memory requests at the peak rate that the memory can handle, queues will build up in front of the memory modules for most strides. The resulting increase in memory access time is not as significant an issue as it may seem at first since most high performance processor architectures tend to be designed to cope with long memory latencies. What such processors absolutely must have is high memory bandwidth, and this is provided far more consistently by I-poly interleaving than by conventional sequential interleaving. Note also that for the important unit stride, I-poly interleaving behaves as well as sequential interleaving in every respect.

The design of an M-way ($M = 2^m$) I-poly interleaved memory requires that all the irreducible polynomials of order m be tested to select the most desirable one. The testing consists, firstly, of checking whether x is a primitive element. (If not, some other primitive element should be used to define the H-matrix). Next, all of the of k-sparse multiples of the modulus polynomial should be generated for $k \geq 2$. It is desirable that this list consist only of very large integers, thereby reducing the number of troublesome strides. Lastly, it is desirable that as many as possible of these k-sparse multiples be prime numbers since this minimizes the number of integer sub-multiples of k-sparse strides, which have some of the bad properties of the k-sparse stride.

APPENDIX

Theorem 1. Let P(x) be a polynomial of order m over GF(2). The polynomial interleaving scheme in which the physical address $A = \langle a_{n-1}, .., a_1, a_0 \rangle$ is mapped into the randomized address $B = \langle b_{n-1}, .., b_1, b_0 \rangle$, where

$$B(x) = (A(x) \text{ div } x^m) * x^m + (A(x) \text{ mod } P(x)),$$

is a permutation scheme.

Proof: Let A_H and A_L refer to $\langle a_{n-1}, .., a_1, a_m \rangle$ and $\langle a_{m-1}, .., a_1, a_0 \rangle$, (i.e., the high-order and low-order bits), respectively. Let B_H and B_L be similarly defined for $\langle b_{n-1}, .., b_1, b_0 \rangle$. Consider the 2^m physical addresses which all have the same high order bits. Since for any physical address the high-order bits of the physical and randomized address are identical, B_H for all of the 2^m randomized addresses is identical. Let the randomized low-order bits for two distinct physical low-order bits, A_{L1} and A_{L2} be B_{L1} and B_{L2}, respectively. Let $A_D(x) = A_{L1}(x)-A_{L2}(x) \neq 0$, and let $B_D(x) = B_{L1}(x)-B_{L2}(x)$. Since $B_{L1}(x) = A_{L1}(x) \bmod P(x)$ and $B_{L2}(x) = A_{L2}(x) \bmod P(x)$, $B_D(x) = A_D(x) \bmod P(x)$. Since $A_D(x)$ is a lower order polynomial than P(x), $B_D(x) \neq 0$. Therefore, $B_{L1} \neq B_{L2}$ if $A_{L1} \neq A_{L2}$. In other words, all of the 2^m randomized indices are distinct and the randomization function is a permutation scheme■

Lemma. If P(x) is of order m and is odd, i.e., the coefficient of $x^0 \neq 0$, then $x^i \bmod P(x) \neq 0$ for any $i \geq 0$.

Proof: We know that x^i is not divisible by P(x) for any $i \leq m$. Assume that x^i is divisible by P(x) for some $i \geq m$. Therefore, $x^i = P(x)*Q(x)*x^j$ where Q(x) is odd and $j \geq 0$. Let $d = i-j \geq m$. Therefore, $x^d = P(x)*Q(x)$ where $d \geq m$. This means that the coefficient of x^0 in $P(x)*Q(x)$ is equal to 0. But this is impossible since P(x) and Q(x) are both odd polynomials. Therefore, x^i is not divisible by P(x) and $x^i \bmod P(x) \neq 0$ for any $i>0$.■

Theorem 2. Consider the H-matrix corresponding to a polynomial interleaving scheme defined by the polynomial P(x). If P(x) is odd (i.e., the associated integer P is odd), then all of the square sub-matrices, S(m,i), of the H-matrix are non-singular and all strides of the form $S = S_0*2^k$ for any $k \geq 0$ are statistically identical in their behavior to that of the stride S_0.

Proof: The rows of the matrix S(m,i) consist of the coefficients of $x^j \bmod P(x)$, for $j = i,..,i+m-1$. S(m,i) is singular if it is possible to find coefficients, c_j, $j = i,..,i+m-1$, over GF(2) such that $(c_i x^i+..+c_{i+m-1}x^{i+m-1}) \bmod P(x) = 0$. Since $(c_i x^i+..+c_{i+m-1}x^{i+m-1}) = (c_i x^0+..+c_{i+m-1}x^{m-1})*x^i$, $(c_i x^i+..+c_{i+m-1}x^{i+m-1}) \bmod P(x) = 0$ if either $(c_i x^0+..+c_{i+m-1}x^{m-1}) \bmod P(x) = 0$ or $x^i \bmod P(x) = 0$. The former cannot be true since $(c_i x^0+..+c_{i+m-1}x^{m-1})$ is of lower order than P(x) and the latter cannot be true by the above lemma. Therefore, all S(m,i) are non-singular.

Let $\{A_i\}$, $i = 0,..,j$, be a reference sequence with stride $S = S_0*2^k$ for some $k \geq 0$. Let $B_i = S_0*i$, i.e., $\{B_i\}$ is the reference sequence with stride S_0. Therefore, $A_i = B_i*2^k$, $A_i(x) = B_i(x)*x^k$, and $A_i(x) \bmod P(x) = [(B_i(x) \bmod P(x)) * x^k] \bmod P(x)$. Consider first any i and j such that B_i and B_j map into the same module, i.e., $B_i(x) \bmod P(x) = B_j(x) \bmod P(x) = R(x)$. In this case, $A_i(x) \bmod P(x) = A_j(x) \bmod P(x) = [R(x)*x^k] \bmod P(x)$. Consider next any i and j such that B_i and B_j do not map into the same module, i.e., $B_i(x) \bmod P(x)$

= $R_i(X) \neq B_j(x)$ mod $P(x) = R_j(X)$. In this case, $A_i(x)$ mod $P(x) = [R_i(x)*x^k]$ mod $P(x) \neq A_j(x)$ mod $P(x) = [R_j(x)*x^k]$ mod $P(x)$ since $R_i(x)*x^k \neq R_j(x)*x^k$ and S(m,k) is non-singular for all k≥0. Consequently, A_i and A_j map into the same module if and only if B_i and B_j map into the same module. In other words, the sequence $\{A_i\}$ is equivalent to the sequence $\{B_i\}$ except that the modules have been renamed by a permutation. Hence, the statistics for $\{A_i\}$ and $\{B_i\}$ are identical ■

Theorem 3. In the polynomial interleaving scheme defined by an odd polynomial P(x), all strides that are of the form 2^k, for k≥0, are short-term equi-distributed.

Proof: Since polynomial interleaving is a permutation scheme (Theroem 1), the reference sequence with a stride of 1 is short-term equidistributed. Therefore, by Theorem 2, all strides of the form 2^k, for k≥0, are short-term equi-distributed ■

Theorem 4. With the 2^m-way polynomial interleaving scheme defined by a polynomial P(x) of order m and with a reference sequence $\{A_0,A_1,..,A_K\}$ such that the greatest common divisor of P(x) and $A_i(x)$, for all $0 \leq i \leq K$, is the polynomial G(x) of order q, only 2^{m-q} memory modules are referenced over the whole reference sequence.

Proof: Define $Q_i(x)$ such that $A_i(x) = Q_i(x)*G(x)$, for all $0 \leq i \leq K$, and define T(x) such that $P(x) = T(x)*G(x)$. Then for all $0 \leq i \leq K$, $A_i(x)$ mod $P(x) = (Q_i(x)*G(x))$ mod $(T(x)*G(x)) = Q_i(x)$ mod $T(x)$. Since T(x) is of order m-q, all module indices corresponding to $\{A_i\}$ must be less than 2^{m-q}, i.e., only 2^{m-q} modules are referenced over the sequence $\{A_i\}$ ■

Theorem 5. In an I-poly interleaving scheme defined by the irreducible polynomial P(x) of order m, x^k+1 is not divisible by P(x) for any $k < 2^m-1$ and is divisible by P(x) for $k = 2^m-1$, i.e., the rows of the H-matrix have a maximal period of 2^m-1.

Proof: Let x^d be the smallest power of x such that x^d+1 is divisible by P(x). x^d+1 is divisible by P(x) if and only if x^d mod $P(x) = x^0$ mod $P(x) = 1$, i.e., if row d of the H-matrix is the same as row 0. Since the rows of the H-matrix constitute the successive states of a feedback shift register, this would correspond to the period of the shift register being of length d. From feedback shift register theory (page 316 of [15]) we know that the period, for a shift register corresponding to the irreducible polynomial P(x) of order m and when x is a primitive element, is of length 2^m-1. Therefore, x^k+1 is not divisible by P(x) for any $k < 2^m-1$ and is divisible by P(x) for $k = 2^m-1$ ■

REFERENCES

1. W. Abu-Sufah and A. D. Mahoney, "Vector processing on the Alliant FX/8 multiprocessor", Proceedings of the 1986 International Conference on Parallel Processing, pp. 559-563, 1986.

2. B. R. Rau, M. S. Schlansker and D. W. L. Yen, "The Cydra 5 Stride-Insensitive Memory System", Proceedings of the 1989 International Conference on Parallel Processing, Vol. 1, pp. 242-246, August 8-12, 1989.

3. W. Oed and O. Lange, "On the effective bandwidth of interleaved memories in Vector Processing Systems", IEEE Transactions on Computers, Vol. TC-34, No. 10, pp. 949-957, October 1985.

4. D. H. Bailey, "Vector computer memory bank contention", IEEE Transactions on Computers, Vol. TC-36, No. 3, March 1987.

5. D. H. Lawrie and C. R. Vora, "The Prime Memory System for Array Access", IEEE Transactions on Computers, Vol. TC-31, No. 5, pp. 435-442, May 1982.

6. P. Budnik and D. J. Kuck, "The Organization and Use of Parallel Memories", IEEE Transactions on Computers, Vol. TC-20, No. 12, pp. 1566-69, December 1971.

7. D. H. Lawrie, "Access and Alignment of Data in an Array Processor", IEEE Transactions on Computers, Vol. TC-24, No. 12, pp. 1145-55, December 1975.

8. D. T. Harper III and J. R. Jump, "Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme", IEEE Transactions on Computers, Vol. TC-36, No. 12, pp. 1440-1449, December 1987.

9. D. E. Knuth and G. S. Rao, "Activity in an interleaved memory," IEEE Transactions on Computers, Vol. TC-24, No. 9, pp. 943-944, September 1975.

10. F. A. Briggs and E. S. Davidson, "Organization of Semiconductor Memories for Parallel-Pipelined Processors", IEEE Transactions on Computers, Vol. TC-25, No. 2, pp. 162-169, February 1977.

11. D. Y. Chang, D. J. Kuck and D. M. Lawrie, "On the effective bandwidth of parallel memories," IEEE Transactions on Computers, Vol. TC-26, No. 5, pp. 480-489, May 1977.

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

13. A. Norton and E. Melton, "A Class of Boolean Linear Transformations for Conflict-Free Power-of-Two Stride Access", Proceedings of the 1987 International Conference on Parallel Processing, pp. 247-254, 1987.

14. G. S. Sohi, "Logical Data Skewing Schemes for Interleaved Memories in Vector Processors", Computer Sciences Technical Report #753, University of Wisconsin-Madison, September 1988.

15. H. S. Stone, Discrete Mathematical Structures, Science Research Associates, Chicago, 1973.