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 such as vector and VLIW processors. 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.

In a pseudo-randomly interleaved memory architecture, memory locations are assigned to the memory modules in some pseudo-random fashion in the hope that those sequences of references, which 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

The gap that has always existed between the processor cycle time and that of high-density DRAM memory chips continues to grow. This leads to a considerable performance mismatch between the rate at which the processor can make data requests and the rate at which DRAM can service those requests. This problem is exacerbated by the use of multiple processors and uniprocessors, such as vector [1] and VLIW [2,3] processors, which are capable of making multiple requests per cycle.

The conventional solution is to provide each processor with a data cache constructed out of SRAM. The problem is maintaining cache coherency, at high request rates, across multiple private caches in a multiprocessor system. The alternative is to use a shared cache if the additional delay incurred in going through the processor-cache interconnect is acceptable. The problem here is that the bandwidth, even with SRAM chips, is inadequate unless some form of interleaving is employed in the cache. So once again, the interleaving scheme used is an issue. Furthermore, data caches are susceptible to problems arising out of the lack of spatial and/or data locality in the data reference pattern of many applications. This phenomenon has been studied and reported elsewhere, e.g., in [4,5]. Since data caches are essential to achieving good performance on scalar computations with little parallelism, the right compromise is to provide a data cache that can be bypassed when referencing data structures with poor locality. This is the solution employed in various recent products such as the Convex C-1 and Intel's i860.

Interleaved memory systems. Whether or not a data cache is present, it is important to provide a memory system with bandwidth to match the processors. This is done by organizing 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.

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. These generate access patterns with constant stride (both unit and non-unit), patterns with structure but which do not have constant stride and access sequences that are irregular and apparently random. Generally, most applications generate multiple such access streams that proceed simultaneously in an interleaved fashion. A good interleaving scheme should be robust enough to deliver high bandwidth across all such access patterns.

In the case of sequentially interleaved memory (SIM) with M memory modules, the location with address A is in memory module (A mod M). SIM works well (in fact optimally) if the memory references have a stride that is prime relative to M. In this case, the processor's references will be uniformly distributed over all the memory modules, thereby allowing them to operate in parallel and match the bandwidth requirements imposed by the processor. On the other hand, SIM is subject to dramatic performance degradation when the

memory 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 that of a non-interleaved memory. SIM relies heavily on the programmer's ability to develop algorithms which generate only odd strides.

In a pseudo-randomly interleaved memory (PRIM) system, the mapping between the address, A, and the memory module is pseudo-random. PRIM, when properly designed, can be considerably more robust (i.e., insensitive to the address reference pattern) than SIM. 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 study the shortcomings of conventional interleaving schemes such as SIM, prime degree interleaving and skewed-storage schemes. In Section 3, we discuss pseudo-random interleaving with particular emphasis on XOR-based permutation schemes. In Section 4, we introduce and develop the concept of polynomial interleaving, establish its connections to XOR-based permutation schemes, and present a number of theorems regarding polynomial interleaving. Section 5 presents the results of simulation runs that were undertaken to validate the behavior of polynomial interleaving schemes and compare them with other schemes. We conclude in Section 6 by outlining a procedure for designing an irreducible polynomial interleaving scheme.

2. CONVENTIONAL INTERLEAVING SCHEMES

Terminology and assumptions. We shall consider interleaved memory systems with M memory modules. By and large, M is a power of 2, and in that case $M = 2^m$. The module index, corresponding to a particular memory location, is 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.

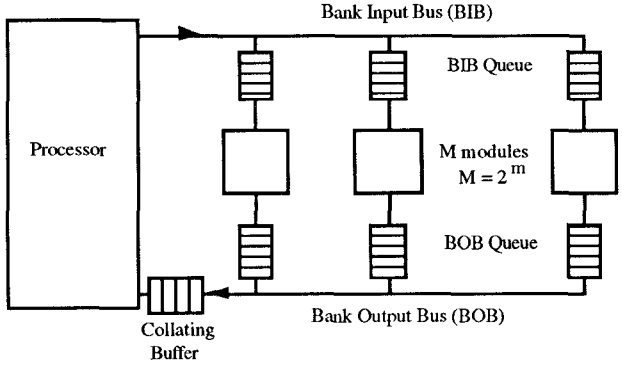

Figure 1. Structure of the processor-memory system that is considered in this paper. The memory modules receive requests over the Bank Input Bus (BIB) and return data over the Bank Output Bus (BOB). Each memory module has a queue on the input side (BIB Queue) to buffer requests when the module is busy and a queue on the output side (BOB Queue) to buffer returning data while it is arbitrating for the BOB. Since requests to distinct modules can be serviced out of order, the Collating Buffer holds returning data until they can be returned to the processor in order.

The memory architecture developed in this paper has applicability to multiprocessors and uniprocessors capable of multiple requests per cycle. However, for brevity we shall restrict our discussion in three ways. Firstly, we only consider uniprocessors capable of making one request per processor cycle. Secondly, we restrict our discussion to memory systems with a single memory bank, i.e., a set of memory modules that share a single pair of input and output buses. Figure 1 illustrates the class of processor-memory systems that is considered in this paper. Thirdly, we only consider either a reference sequence that consists of accesses that are randomly directed to the M memory modules or one that constitutes an arithmetic sequence with some fixed stride that is not necessarily unity.

The processor cycle time is used as the unit of measure. For the purposes of this paper, we 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 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 SIM, consisting of $M = 2^m$ modules, such that the location with address 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 maximal 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 can accept one request every cycle and can 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.

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 in which the number of memory modules, M, is a prime number [6]. 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. The drawback is that the computations of the module index and the word address are no longer trivial. 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.

Skewed-storage schemes. The problem with the two previous approaches 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. The skew for each set of M locations could have an obvious pattern to it [7-9] or it could be pseudo-random [10]. One example of the former type of scheme 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. Whereas 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 0, respectively (Figure 2). 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. Nevertheless, this basic idea, of permuting each set of M consecutive locations differently across the modules, is valuable and is employed in the pseudo-random interleaving schemes that are developed in Sections 3 and 4.

Buffering. No interleaving scheme, by itself, can guarantee high bandwidth when the reference sequence has a random or irregular pattern of accesses to the memory modules. 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 [11]. However, if the memory system has adequate buffering to queue up references to busy modules, the full bandwidth of M modules can be achieved. Figure 3 shows the result of a simulation, for the system in Figure 1 with 16-way interleaving, which confirms this. Adequate buffering, at all points of contention for a resource such as a memory module or a bus, is needed for full bandwidth with random access patterns.

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 2. A skewed-storage 8-way interleaving scheme

Conversely, buffering alone is insufficient 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 greater than the rate at which the module can service them. This causes the queues to fill up almost immediately and, thereafter, the processor will be stalled repeatedly.

3. PSEUDO-RANDOM INTERLEAVING

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. By providing adequate buffering to queue the short clusters, the full interleaved bandwidth would be achieved.

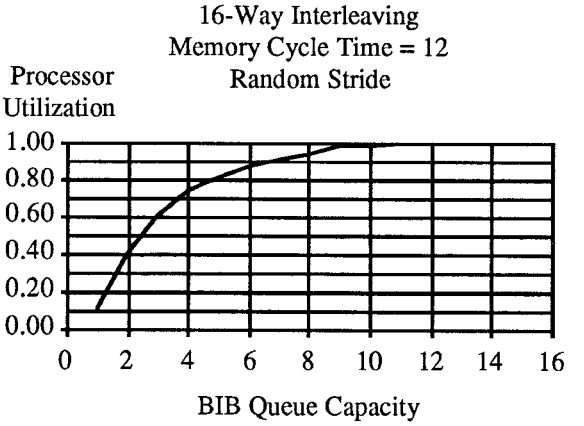

Figure 3. Performance of a sequentially interleaved memory, with buffering per module, for a random reference stream.

In this paper, 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 $A = \langle a_{n-1}, .., a_0 \rangle$ and the randomized address by $B = \langle b_{n-1}, .., b_0 \rangle$. (The bits that determine the byte address within a word are not relevant to this discussion). 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. B_H and B_L are similarly defined for B. 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). It is highly desirable that h(•) possess two properties. Firstly, it should be a bijection and, secondly, 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.

There is no benefit derived from the randomizing function modifying $\langle a_{n-1}, .., a_m \rangle$ 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, viz., when $\langle a_{m-1}, .., a_0 \rangle$ go through all 2^m combinations with $\langle a_{n-1}, .., a_m \rangle$ fixed, $\langle b_{m-1}, .., b_0 \rangle$ 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 for h(•) to be a bijection.

Permutation using the XOR function. It is desirable that the computation of the randomized address be inexpensive both in the amount of hardware required as well as in the time taken to do it. Hence, the idea of randomizing the physical address by XOR-ing it with another bit pattern is very attractive [5,12-15] and, in fact, two machines [5,16] have been built using such randomization, the former as a commercial product. Such a mapping is a permutation. 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 m≤n 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 the boolean matrix

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

where H(i,j) = 1, (i = 0,..,n-1, and j = 0,..,m-1), if and only if a_i is an input to the XOR gate whose output is h_j. (Note that contrary to convention, the rows are numbered from the bottom upward and the columns from the right to the left). 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).

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 the square sub-matrix of H,

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

where 1≤q≤m and 0≤i≤n-q. The mapping is a bijection if D(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. A few of the factors to be considered in designing an H-matrix are discussed below.

Any H-matrix with a non-singular M(m,0) is minimally acceptable. However, this does not define a unique H-matrix. Certain H-matrices may be expected to be 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. 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 access pattern 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 D(m,i) be non-singular for all i, 1≤i≤n-m.

This still does not guarantee a good H-matrix. Figure 4b is an example of an H-matrix, every D(m,i) of which is non-singular, but which, nevertheless, is a poor choice. The rows repeat themselves with a 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 physical address come in pairs that are k bit positions apart (Figure 4a). Identical rows, k apart in the H-matrix, are XOR-ed together, cancelling each other. Clusters of up to 2^k consecutive references will be to the same module (Figure 4c).

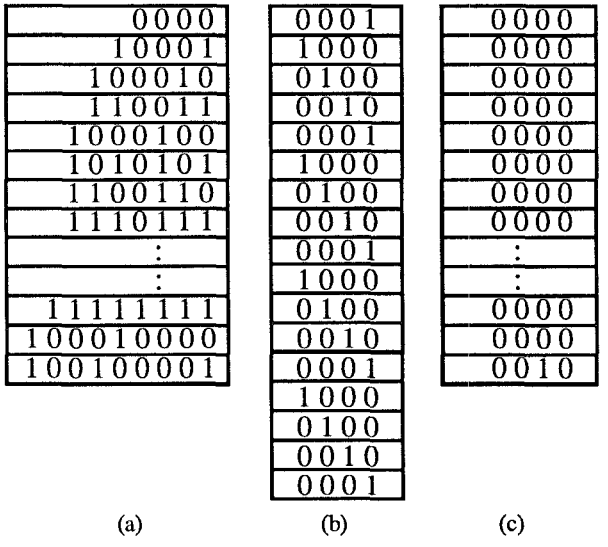

Figure 4. (a) A physical address sequence with a stride of 17. (b) An H-matrix with repetitive rows. (c) The output sequence of addresses; long sequences of addresses get mapped to the same module.

To avoid such problems, and in the absence then of any better theory at that time, the H-matrix for the Cydra 5 [5] was designed to be random. The rows of the H-matrix were selected randomly, without replacement, from the set of 2^5 bit patterns with odd parity (m was equal to 6, n was less than 32). The non-singularity of D(q,0), for 2≤q≤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 5. More extensive measurements have been conducted on the Cydra 5 memory system. They demonstrate the robustness of pseudo-randomly interleaved memory in the face of multiple concurrent requests streams with

different strides or different offsets. Some of these measurements have been reported in [5].

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. Before describing an interleaving strategy that comes closer to allowing such prediction, it is instructive to 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.

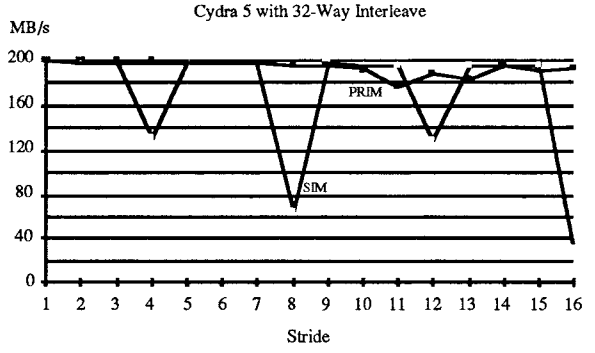

Figure 5. Performance of the Cydra 5 pseudo-randomly interleaved memory system with and without randomization using a random H-matrix.

Consider the class of polynomials whose coefficients are in the Galois Field GF(2) [17], 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 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 $A = \langle a_{n-1}, .., a_1, a_0 \rangle$, 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 [17] 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 much the same reason 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 $R_i(x) = x^i \bmod P(x)$. Bearing in mind that $A(x) = a_{n-1}x^{n-1} + ..+ a_0$,

$$\begin{aligned} R(x) &= A(x) \bmod P(x) \\ &= [a_{n-1}x^{n-1} + ..+ a_0] \bmod P(x) \\ &= [\,(a_{n-1}x^{n-1}) \bmod P(x) + .. + (a_0x^0) \bmod P(x)\,] \bmod P(x) \end{aligned}$$

however, since each of the terms in the square brackets is a polynomial of order less than n, so must their sum. Consequently,

$$\begin{aligned} R(x) &= (a_{n-1}x^{n-1}) \bmod P(x) + .. + (a_0x^0) \bmod P(x) \\ &= a_{n-1}(x^{n-1} \bmod P(x)) + .. + a_0(x^0 \bmod P(x)), \text{ since } a_i = 0,1 \\ &= a_{n-1}R_{n-1}(x) + .. + a_0R_0(x) \end{aligned}$$

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 following theorems are useful in proving certain important properties of polynomial interleaving.

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.

Proof: 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 $<b_{n-1}, .., b_1, b_0>$. 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 ■

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

Figure 6 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.

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, D(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 D(m,i) consist of the coefficients of $x^j \bmod P(x)$, for j = i,..,i+m-1. D(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 D(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., $B_i(x) \bmod P(x) = R_i(X) \neq B_j(x) \bmod P(x) = R_j(X)$. In this case, $A_i(x) \bmod P(x) = [R_i(x)*x^k] \bmod P(x) \neq A_j(x) \bmod P(x) = [R_j(x)*x^k] \bmod P(x)$ since $R_i(x)*x^k \neq R_j(x)*x^k$ and D(m,k) is non-singular for all $k \geq 0$. 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 ■

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. In a virtual memory system, the high order bits of the virtual address are replaced by the corresponding bits of the physical address. Unless a conscious effort is made to avoid it, this mapping has the effect of randomizing the high order bits of the address. Because of this, only those strides need be considered that are less than the page size divided by the word size.

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.

Proof: Since polynomial interleaving is a permutation scheme (Theorem 1), the reference sequence with a stride of 1 is short-term equi-distributed. Therefore, by Theorem 2, all strides of the form 2^k, for $k \geq 0$, 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 experiments conducted to date confirm the above hypothesis and it is believed to be true although a proof has not yet presented itself. 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), $0 \leq i \leq K$, mapped into a subset of the M modules.

Theorem 4. With the 2^m-way polynomial interleaving scheme defined by a (not necessarily irreducible) 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 $0 \leq i \leq K$, 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 $0 \leq i \leq K$, $A_i(x) \bmod P(x) = (Q_i(x)*G(x)) \bmod (T(x)*G(x)) = Q_i(x) \bmod T(x)$. 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 $\{A_i\}$ ■

Such a situation would arise if $\{A_0,A_1,..,A_K\}$ constituted an arithmetic sequence with stride S, such that $A_i(x) = S(x)*i(x)$, $0 \leq i \leq K$, 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 contains 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, $0 \leq i \leq 2^k$, $A_i(x) = S(x)*i(x)$. This is because, over the stated range of i, the partial binary products, when added, do not generate any carry. Binary and polynomial arithmetic are identical over this range. By Theorem 4, 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 $\partial^i \bmod P(x)$, 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.

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 $x^d \bmod P(x) = x^0 \bmod P(x) = 1$, 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 [17]) 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 $k = 2^m-1$ ■

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.

Lack of space prevents proof of further properties. For instance, in the case of an I-poly interleaving scheme defined by an irreducible polynomial P(x) of order m, and a K*K

matrix, ($K = 2^k$, $k < 2^m - 1$), which is aligned on a 2^{2k} boundary, the following access patterns demonstrate short-term equi-distribution: (a) all row access (b) all column access (c) access along the main diagonal (d) access along the main anti-diagonal.

5. MEASUREMENTS

Measurements were performed by simulating a processor-memory system of the form shown in Figure 1 with one processor and 16-way interleaved memory. The memory cycle time was 12 with an access time of 8 cycles which presents a lower bound on the latency of a memory request. Each simulation was run for 16,384 cycles with the processor making a memory reference every cycle unless stalled. The reference sequence consisted either of an arithmetic sequence with a constant stride or a random sequence. The objectives were to validate the theory of Section 4, demonstrate the effectiveness of I-poly interleaving and to compare it with sequential interleaving and the random H-matrix scheme of the Cydra 5. In addition, the series of simulations described below, and the order in which they are performed, is intended to be illustrative of the procedure that one might employ in designing a pseudo-randomly interleaved memory system. The memory system design parameters that affect performance the most are the degree of interleaving, the memory cycle time, the interleaving scheme and the amount of buffering provided per memory module. The first two are fixed for the purposes of this discussion. There is one other parameter of importance. Due to program dependences or hardware limitations, there is a limit to the memory latency that a processor can tolerate. Sophisticated compilers can increase the program's tolerance for long latencies by various forms of instruction re-ordering, but when this limit is reached, the processor stalls. VLIW processors [2,3] impose an explicit upper bound on the acceptable latency. In the Cydra 5, this latency bound can be varied under program control by depositing the desired value in the memory latency register (MLR) [5]. The compiler determines the most appropriate value based on its success in re-ordering the code to make it latency-tolerant. For architectures other than VLIW, the MLR value may be interpreted as a fuzzy measure of the latency tolerance of the processor-program pair.

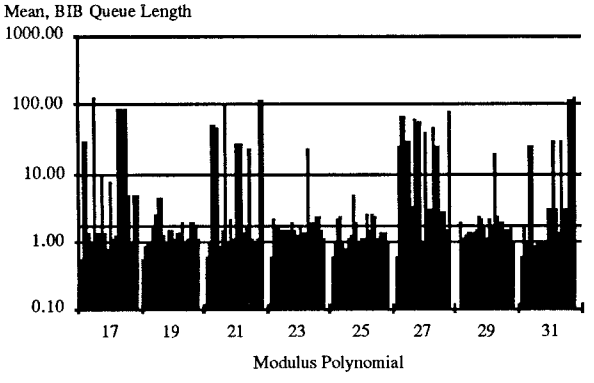

Figure 7. Average BIB Queue length (assuming unbounded buffer capacity per memory module) for all odd strides between 1 and 63 and all odd fourth-order modulus polynomials.

The first set of simulations were performed to demonstrate the merit of selecting an irreducible polynomial as the modulus. They were conducted assuming unbounded BIB (Bank Input Bus) Queue capacity and an unbounded MLR value. For these simulations alone, the memory cycle time was set at 16 cycles so that, with the processor making a request every cycle, the memory system would be operating in a state of saturation. This was done to exaggerate and thereby highlight the effects of the temporal clustering of references. The figure of merit used, as a measure of the extent of temporal clustering, is the average BIB Queue length. Figure 7 shows this statistic for all odd fourth-order polynomials and for all 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. Polynomial 19 was used as the modulus polynomial for I-poly interleaving in the rest of the simulations. Also, so as to correspond to a more realistic design point, the memory cycle time was selected to be 12 (less than the degree of interleaving) since it is well understood from queuing theory that it is inadvisable to operate any queueing system at, or close to, saturation.

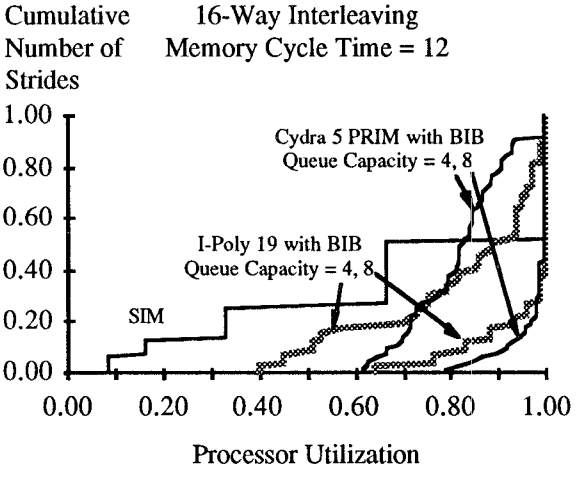

Figure 8. Frequency distribution of processor utilization with SIM, I-poly interleaving and Cydra 5 PRIM for strides 1 through 64 and for BIB Queue capacities of 4 and 8. (The buffer capacity is irrelevant with SIM).

In a realizable system, the BIB Queue capacity is finite and whenever a BIB Queue fills up, the processor stalls. Ultimately, the most meaningful figure of merit is the processor utilization, i.e., the fraction of time that the processor is not stalled. So, the objective of the second set of simulations was to understand the amount of BIB Queue capacity needed in order to avoid stalling the processor too frequently. To isolate the influence of this parameter, the MLR value was left unbounded. A subset of the simulation results are shown in Figure 8 which compares sequential interleaving (SIM) with I-poly 19 interleaving and Cydra 5 PRIM for BIB Queue capacities of 4 and 8 and for strides from 1 through 64. With SIM, the BIB Queue capacity is irrelevant; the queue never builds up for odd strides and the BIB Queue will overflow, regardless of its capacity, for even strides. 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. Consider first a BIB Queue capacity of 4. Cydra 5 PRIM has the least variability in its performance and SIM has the most. The processor utilization for Cydra 5 PRIM is between 0.65 and 0.90 for the middle 80% of the strides but is between 0.16 and 1.00 for SIM (for the same percentage of strides). The other side of the coin is that SIM achieves full performance for 50% of all strides whereas Cydra 5 PRIM does so for only 10% of all strides. I-poly interleaving is intermediate between SIM and Cydra 5 PRIM in every respect. With both PRIM schemes, the worst 50% of the strides behaved better than the worst 50% of the strides for SIM by factors of between 1.3 and 4. The best 50% of PRIM strides were never worse than the best 50% of SIM strides by more than a factor of 1.2.. Notice that even the worst stride with I-poly interleaving is better than a quarter of all the strides with SIM.

Interleaving schemes with low variability in performance are favored as the BIB Queue capacity increases. Both PRIM schemes yield better performance for all strides as capacity increases. However, the advantage that the higher variability scheme had, on the well-behaved strides, over the lower variability scheme disappears because processor utilization cannot increase beyond 1.0. So, whereas on the poorly-behaved strides Cydra 5 PRIM does somewhat better than I-poly 19, and both do significantly better than SIM, SIM and I-poly 19 never do much better than Cydra 5 PRIM. Almost all of the strides for PRIM are better than 50% of all the strides with SIM. With a buffer capacity of 8 or more, almost all strides yield a processor utilization of better than 0.8. Although not evident from Figure 8, the simulation results show that perfect processor utilization is achieved for stride 1 as long as there is the ability to buffer at least two requests per memory module (the request currently being served and a second one that causes the processor to stall). With a BIB Queue capacity of 8, both PRIM schemes appear to be unquestionably better than SIM, and Cydra 5 PRIM is better than I-poly 19. (The lack of adequate buffering is probably the cause for the relatively disappointing behavior of a random H-matrix scheme evaluated in [10]). Figure 3 shows that if a 16-way interleaved memory system is to perform well for a random stride, a BIB Queue capacity of about 8 is needed. So, the BIB Queue capacity was fixed at 8 in the remaining experiments.

The penalty for buffering is the greater latencies incurred in making a memory request. The results shown in Figure 8 do not take this into account. The third set of simulations were performed to understand how processor utilization behaved as a function of the MLR value. The results for MLR values of 12 and 56 are shown in Figure 9 for all three interleaving schemes. Once again, SIM is insensitive to the MLR value as long as it is not less than 8. For odd strides, the request is back in 8 cycles. For even strides, the queues tend to grow in an unbounded fashion and it makes little difference whether the processor is stalled because the BIB Queue fills up or because the MLR limit is exceeded. Both PRIM schemes pay a significant penalty for an MLR value of 12 and a relatively small one for an MLR value of 56. (Note that with a memory cycle time of 12, 56 cycles corresponds to the delay experienced by a request which finds 4 requests in front of it in the BIB Queue). Most of the discussion about Figure 8 is applicable here at a qualitative level. Once again, Cydra 5 PRIM demonstrates less variability than I-poly 19 which, in turn, is less variable than SIM. Once again, as the MLR value is increased, schemes with higher variability lose whatever advantage they had in the case of well-behaved strides. If sufficiently high MLR values are acceptable, PRIM using a random H-matrix does best. For low values of MLR, the evidence is inconclusive; each scheme can be shown to be best depending on the relative importance assigned to each stride.

6. CONCLUSION

XOR-based PRIM, implemented using an H-matrix, provides most of the attributes of a good interleaving scheme: implementation simplicity and robustness in the face of varying address patterns. In addition, I-poly interleaving allows one to prove certain important properties regarding its behavior. The major hardware cost associated with PRIM is the buffering that is required per memory module for it to perform well. However, if one assumes the viewpoint that buffering is required with *any* interleaving scheme that wishes to perform well on random or irregular access patterns, then it is not a relative disadvantage of PRIM.

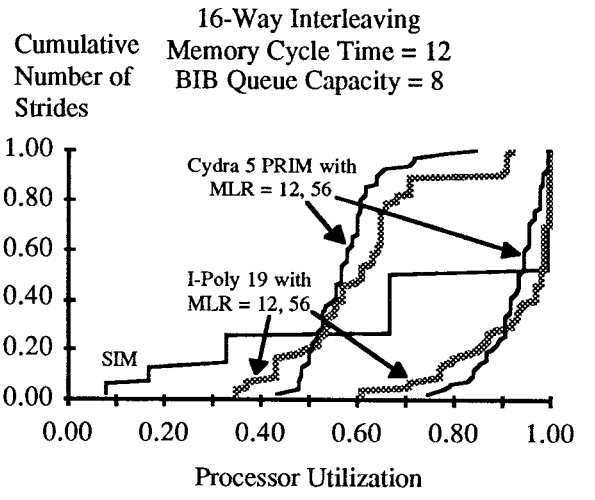

Figure 9. Frequency distribution of processor utilization with SIM, I-poly interleaving and Cydra 5 PRIM for strides 1 through 64 and for MLR values of 12 and 56. (The MLR value is irrelevant with SIM).

The major performance penalty of PRIM results from the increase in memory access time caused by pseudo-randomization. When the processor makes requests at the peak memory rate, queues will build up, to a greater or lesser extent, at the memory modules for all strides. If the processor and program, in conjunction, can tolerate the increase in latency, PRIM is the better solution; by arranging for the program to make requests well in advance of attempting to use the data, consistently good performance can be achieved across all strides. Most high performance processor architectures tend to be designed to cope with long memory latencies and, for them, PRIM is the preferred choice. On the other hand, if latency cannot be tolerated, PRIM offers mediocre performance for most strides whereas SIM provides either very good or very poor performance depending on the stride. However, it has been observed that, in practice, most programs are not latency-tolerant are unable to generate high request rates. Consequently, the memory system is not congested, and all interleaving schemes behave quite similarly. The data cache should be used for workloads which have adequate locality of reference, whether or not they are latency-tolerant. In this case

the interleaving scheme is irrelevant since memory is only accessed on a cache line basis.

The comparison between the two PRIM schemes, Cydra 5's random H-matrix and I-poly interleaving, shows that in most ways I-poly interleaving is intermediate between the random H-matrix and SIM. For lower values of MLR, I-poly interleaving does better than the random H-matrix for a larger percentage of strides. For sufficiently high MLR values, the random H-matrix seems to perform best. It is indeed remarkable that the random H-matrix, that was designed as an act of desperation, does so well! However, there is still the nagging doubt that, perhaps, stumbling upon this H-matrix was pure luck and that the next random H-matrix designed would not fare as well.

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.

In this paper we have only considered single access patterns that have a constant or random stride. In reality, the reference patterns generated by a program consist of many interleaved sequences of this type and of finite length. Furthermore, in a multiprocessor system, there will be a number of such composite sequences of references going on simultaneously. A complete discussion of this is beyond the scope of this paper. Suffice it to say that as the sequences get more complex, they generally start behaving more and more like a random sequence. Thus the variability that is seen across all strides diminishes and converges towards the performance that the memory system is capable of with a random stride.

ACKNOWLEDGEMENTS

Ongoing discussions with Mike Schlansker and the constructive comments from Al Davis, Bill Worley and the anonymous referees have made a big difference to this paper. Only lack of space prevents the incorporation of all their suggestions.

REFERENCES

1. R. M. Russell, "The CRAY-1 Computer System", Comm. ACM, Vol.21, pp. 63-72, 1978.
2. B. R. Rau, D. W. L. Yen, W. Yen and R. A. Towle, "The Cydra 5 Departmental Supercomputer: Design Philosophies, Decisions and Trade-offs", Computer, Vol. 22, No. 1, January 1989.
3. R. P. Colwell, R. P. Nix, J. J. O'Donnell, D. B. Papworth and P. K. Rodman, "A VLIW Architecture for a Trace Scheduling Compiler", Proc. the Second Intl. Conf. on Arch. Support for Prog. Lang. and Oper. Sys., pp. 180-192, October 1987.
4. W. Abu-Sufah and A. D. Mahoney, "Vector processing on the Alliant FX/8 multiprocessor", Proc. the 1986 Intl. Conf. on Par. Proc., pp. 559-563, 1986.
5. B. R. Rau, M. S. Schlansker and D. W. L. Yen, "The Cydra 5 Stride-Insensitive Memory System", Proc. the 1989 Intl. Conf. on Par. Proc., Vol. 1, pp. 242-246, August 8-12, 1989.
6. D. H. Lawrie and C. R. Vora, "The Prime Memory System for Array Access", IEEE Trans. Comp., Vol. TC-31, No. 5, pp. 435-442, May 1982.
7. P. Budnik and D. J. Kuck, "The Organization and Use of Parallel Memories", IEEE Trans. Comp., Vol. TC-20, No. 12, pp. 1566-69, December 1971.
8. D. H. Lawrie, "Access and Alignment of Data in an Array Processor", IEEE Trans. Comp., Vol. TC-24, No. 12, pp. 1145-55, December 1975.
9. D. T. Harper III and J. R. Jump, "Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme", IEEE Trans. Comp., Vol. TC-36, No. 12, pp. 1440-1449, December 1987.
10. R. Raghavan and J. P. Hayes, "On Randomly Interleaved Memories", Proc. Supercomputing '90, pp. 49-58, November 1990.
11. D. E. Knuth and G. S. Rao, "Activity in an interleaved memory," IEEE Trans. Comp., Vol. TC-24, No. 9, pp. 943-944, September 1975.
12. J. M. Frailong, W. Jalby and J. Lenfant, "XOR-Schemes: A Flexible Data Organization in Parallel Memories", Proc. the 1985 Intl. Conf. on Par. Proc., 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", Proc. the 1987 Intl. Conf. on Par. Proc., 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. K. Kim and V. K. Prasanna Kumar, "Perfect Latin Squares and Parallel Array Access", Proc. the 16th Ann. Intl. Symp. on Computer Architecture, pp. 372-379, May 1989.
16. G. F. Phister, et. al, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture", Proc. the 1985 Intl. Conf. on Par. Proc., pp. 764-771, 1985.
17. H. S. Stone, Discrete Mathematical Structures, Science Research Associates, Chicago, 1973.