

Brief Contributions

Nonprime Memory Systems and Error Correction in Address Translation

Rajendra S. Katti

Abstract—Using a prime number p of memory banks on a vector processor allows a conflict-free access for any slice of p consecutive elements of a vector stored with a stride not multiple of p . To reject the use of a prime number of memory banks, it is generally advanced that address computation for such a memory system would require systematic Euclidean division by the number p . The Chinese Remainder Theorem allows a simple mapping of data onto the memory banks for which address computation does not require any Euclidean division. However, this requires that the number of words in each memory module m and p be relatively prime. We propose a method based on the Chinese Remainder Theorem for moduli with common factors that does not have such a restriction. The proposed method does not require Euclidean division and also results in an efficient error detection/correction mechanism for address translation.

Index Terms—Address translation, error correction, error detection, logical address, memory systems, physical address, vector processors.

1 INTRODUCTION

A vector is an ordered set of words whose addresses form an arithmetic series. In most cases, the bottleneck for performance on vector applications is the parallel access to vectors in memory. Conflicts arise when accessing vectors when two elements of a vector are stored in the same memory bank. A lot of studies ([1], [2], [3], [4], [5], [6]) have addressed the problem of reducing conflicts for vector access. Using a prime number p of memory banks allows conflict-free access for any slice of p consecutive elements of a constant-strided vector when the stride is not a multiple of p [1]. When the number of memory banks is not a prime number, then address computation requires an Euclidean division [3].

In this paper, we present a method for mapping data to memory banks for the conflict-free access of vectors. The method proposed does not require any Euclidean division during address translation and a prime number of memory banks are no longer required. The proposed method also results in an efficient mechanism for error detection/correction during address translation.

The Self-Testing and Repairing (STAR) computer was one of the first major efforts in the design of a fault-tolerant system for space applications [11]. Error detection in address translation in the STAR computer was implemented using error detecting codes. Address translation is important due to the high frequency at which memory is accessed in vector processors. For example, the SAXPY benchmark requires 1.5 memory accesses for every floating point instruction. If a processor were to execute 20 million floating point instructions per second (MFLOPS), then memory is accessed 30 million times per second. Therefore, it is critical for the address translation unit to function correctly. There are two techniques that could be used to detect errors in the address translation unit. The first would rely on fault detection techniques used

in ALU design [10]. This is the technique used in the STAR computer. The other technique would be to use algorithm-based fault tolerance (ABFT) [12], which has very low overhead compared to the former technique. This scheme has three characteristics:

- 1) encode data at a higher level.
- 2) redesign algorithms to operate on encoded data.
- 3) distribute the computation steps of the redesigned algorithm among computational processors, such that failure of a processor affects as little data as possible.

We propose a modification of the scheme that uses the Chinese Remainder Theorem (CRT) to map addresses to memory modules. The new scheme uses the CRT for moduli with common factors. This leads to an efficient technique for fault detection in address translation that is similar to ABFT. Another advantage of the new mapping scheme is that the number of memory modules do not have to be prime for obtaining higher memory bandwidth. It has been shown that memory bandwidth is higher when the number of memory modules is prime [13]. We show that even though the number of memory modules is not prime the higher bandwidth obtained using a prime number of moduli can still be obtained by using the CRT for moduli with common factors. This paper therefore has two main contributions:

- 1) A technique to obtain higher memory bandwidth with a nonprime number of modules is presented (Section 3).
- 2) An algorithmic technique to detect/correct errors in address translation is presented with very little overhead (Sections 4 and 5).

Section 2 describes existing memory systems with a prime number of memory banks and Section 3 describes the proposed memory system with no restriction on the number of memory banks. Theorem 2 of this section suggests that higher memory bandwidths can be obtained with a nonprime number of modules. Section 4 describes error detection during address translation and Section 5 describes error correction during address translation. Section 6 concludes the paper.

2 THE PRIME MEMORY SYSTEM

In this section, we describe the three existing approaches for mapping logical addresses to physical addresses. Here, p denotes the number of memory modules and m the number of words in each module. The physical address of a word is a pair of integers (u, v) , which denotes the u th word in the v th memory module, where $0 \leq u \leq m - 1$, and $0 \leq v \leq p - 1$. The logical address of a word is an integer d ($0 \leq d \leq pm - 1$). The physical and logical addresses must satisfy the following two conditions,

- 1) $v = d \bmod p$.
- 2) There is a one to one correspondence between a logical and physical address.

We assume that p is prime.

The first mapping scheme obtains the physical address from the logical address as follows [7],

$$v = d \bmod p, u = \lfloor d/p \rfloor.$$

This scheme assumes that $d = up + v$. The drawback of this scheme is that it requires Euclidean Division.

The second scheme obtains the physical address as follows ([2], [3]):

$$v = d \bmod p, u = \lfloor d/(p-1) \rfloor.$$

• The author is with the Department of Electrical Engineering, North Dakota State University, Fargo, ND 58105. E-mail: katti@plains.nodak.edu.

Manuscript received Dec. 12, 1994; revised Jan. 19, 1996.

For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96088.

The computation of u is simplified when $p = 2^t + 1$, but only $(p - 1)/p$ of the d addresses satisfy the constraint $u < m$. This implies that $1/p$ of the memory capacity is wasted.

The third approach for obtaining the physical address is [7],

$$v = d \bmod p, u = d \bmod m.$$

The above approach will work if p and m are relatively prime. This approach relies on the Chinese Remainder Theorem (CRT) to guarantee the fact that there is a one to one correspondence between logical and physical addresses. An example of such a system is shown in Table 1 with $p = 3$ and $m = 8$. Each column of Table 1 corresponds to a memory module and each entry in the table corresponds to the logical address d , of a word. For example, if $d = 10$, then $u = 2$ and $v = 1$.

TABLE 1
AN EXAMPLE OF A MEMORY SYSTEM
BASED ON THE CHINESE REMAINDER THEOREM
WITH $p = 3$ AND $m = 8$

	$v = 0$	$v = 1$	$v = 2$
$u = 0$	0	16	8
$u = 1$	9	1	17
$u = 2$	18	10	2
$u = 3$	3	19	11
$u = 4$	12	4	20
$u = 5$	21	13	5
$u = 6$	6	22	14
$u = 7$	15	7	23

The memory systems described so far do not result in any efficient way of detecting errors in the translation of the logical address to the physical address. In the next section, we develop a new technique of mapping logical addresses to physical addresses that results in an efficient error detection/correction technique in the address translation process. The new mapping also removes any restriction on the number of memory modules required. In other words, p and m need not be relatively prime. The address translation also does not require any Euclidean division.

3 THE NONPRIME MEMORY SYSTEM

In this section, we propose a method for mapping logical addresses to physical addresses that results in an efficient error detection mechanism for address translation. The proposed method is based on the Chinese Remainder Theorem for moduli with common factors. Let $G = \gcd(p, m)$ and $L = \text{lcm}(p, m)$ (\gcd is greatest common divisor and lcm is least common multiple). The logical address is now (d, w) where $0 \leq w \leq G - 1$ and $0 \leq d \leq L - 1$. Computing the physical address can be performed as follows:

$$v = d \bmod p, u = (d + w) \bmod m.$$

Let us consider an example system with $p = 6$ and $m = 9$. Note that $G = 3$ and $L = 18$. The logical address is now (d, w) where $0 \leq d \leq 17$ and $0 \leq w \leq 2$. Table 2 shows the mapping of the logical to the physical addresses. If the logical address is $(d, w) = (15, 2)$, then the physical address is computed as follows,

$$v = d \bmod p = 15 \bmod 6 = 3, u = (d + w) \bmod m = (15 + 2) \bmod 9 = 8.$$

The above mapping is based on the Chinese Remainder Theorem (CRT) for moduli with common factors (see [8] pp. 30-32). In the rest of the paper, $|x|_M$ denotes $x \bmod M$.

THEOREM 1. For the proposed nonprime memory system:

- 1) Every logical address (d, w) , with $0 \leq w < \gcd(p, m)$, $0 \leq d < \text{lcm}(p, m)$, corresponds to one and only one physical address (u, v) , $0 \leq u < m$, $0 \leq v < p$, and
- 2) Every physical address (u, v) corresponds to one and only one logical address d .

TABLE 2
AN EXAMPLE OF THE PROPOSED MEMORY SYSTEM
WITH $p = 6$ AND $m = 9$

	$v = 0$	$v = 1$	$v = 2$	$v = 3$	$v = 4$	$v = 5$
$u = 0$	(0, 0)	(7, 2)	(8, 1)	(9, 0)	(16, 2)	(17, 1)
$u = 1$	(0, 1)	(1, 0)	(8, 2)	(9, 1)	(10, 0)	(17, 2)
$u = 2$	(0, 2)	(1, 1)	(2, 0)	(9, 2)	(10, 1)	(11, 0)
$u = 3$	(12, 0)	(1, 2)	(2, 1)	(3, 0)	(10, 2)	(11, 1)
$u = 4$	(12, 1)	(13, 0)	(2, 2)	(3, 1)	(4, 0)	(11, 2)
$u = 5$	(12, 2)	(13, 1)	(14, 0)	(3, 2)	(4, 1)	(5, 0)
$u = 6$	(6, 0)	(13, 2)	(14, 1)	(15, 0)	(4, 2)	(5, 1)
$u = 7$	(6, 1)	(7, 0)	(14, 2)	(15, 1)	(16, 0)	(5, 2)
$u = 8$	(6, 2)	(7, 1)	(8, 0)	(15, 2)	(16, 1)	(17, 0)

PROOF. From the logical address (d, w) , the physical address (u, v) can be obtained as follows,

$$v = d \bmod p, u = (d + w) \bmod m.$$

It is therefore obvious that every logical address corresponds to one and only one logical address. The number of memory modules p and the number of memory locations in each module m correspond to the moduli (a_1, a_2) , in the Chinese Remainder Theorem for moduli with common factors. The theorem states that there is one and only one residue representation $(|x|_m, |x|_p) = (b_1, b_2)$, for every integer x , $0 \leq x < \text{lcm}(p, m)$. If $G = \gcd(p, m)$ and $L = \text{lcm}(p, m)$, then $GL = pm$. For every physical address of the form $(|x + 0|_m, |x|_p)$, there is one and only one integer x in the range $[0, L - 1]$. For every physical address of the form $(|x + 1|_m, |x|_p)$, there is one and only one integer x in the range $[L, 2L - 1]$. For every physical address of the form $(|x + w|_m, |x|_p)$, there is one and only one integer x in the range $[wL, (w + 1)L - 1]$, where $0 \leq w \leq G - 1$. When $w = G - 1$, every physical address $(|x + G - 1|_m, |x|_p)$ is mapped to an integer in the range $[(G - 1)L, GL - 1]$. Since $GL - 1 = pm - 1$, there is a one to one correspondence between every physical address and logical address. Therefore, all logical addresses in the range 0 to $pm - 1 = GL - 1$ can be mapped to one and only one physical address and every physical address is mapped to one and only one logical address. The logical address is now given by an integer between 0 and $L - 1$ and w . \square

The following theorem is valid for the proposed mapping [1].

THEOREM 2. When the memory module number is given by $v = d \bmod p$, then any vector with p elements and stride R , can be accessed without conflicts if and only if $\gcd(p, R) = 1$. In other words, the p elements of the vector are stored in different memory modules if $\gcd(p, R) = 1$.

This implies that for any vector with a stride R , $p/\gcd(p, R)$ consecutive elements of the vector are stored in distinct memory banks. For example, let us consider accessing a vector of length 6 and stride 5 for the memory system of Table 2. Let us assume that the first element of the vector has a physical address of $(u, v) = (1, 4)$ or logical address $(d, w) = (10, 0)$. The second, third, fourth, fifth, and sixth elements of the vector are then stored in memory modules 3, 2, 1, 0, and 5, respectively. The logical addresses of these elements would be $(15, 0)$, $(2, 1)$, $(7, 1)$, $(12, 1)$, and $(17, 1)$. If two elements of a vector with stride 1 have logical addresses (d_1, w_1) , and (d_2, w_2) , then either

$$d_2 = d_1 + 1, w_2 = w_1$$

or

$$d_1 = L - 1, d_2 = 0, w_2 = w_1 + 1.$$

Therefore, if the logical address of the first element of a vector with stride 5 is (10, 0), then the logical address of the second element of the vector is (15, 0).

We now consider the problem of a logical address being of the form (d, w) . It can be argued that the logical address in a system with p memory modules and m words per module, must be an integer X , such that $0 \leq X < pm$. In the system defined by us, the logical address is (d, w) , where, $0 \leq d < L$ and $0 \leq w < G$ ($L = lcm(p, m)$ and $G = gcd(p, m)$). We can still think of the above system with logical address X such that $X = Lw + d$. If we were to use X as a logical address, then we would have to convert X first to (d, w) . This is not a problem as we could choose p and m such that w is as small as possible. For example, we could choose $p = 46$ and $m = 178$. Therefore, $G = 2$, $L = 4,094$, and $0 \leq X < 8,188$. This implies that w can be either 0 or 1. If $0 \leq X \leq 4,093$, then $w = 0$ and if $4,094 \leq X \leq 8,187$, then $w = 1$. Note that $d = X \bmod L$. Therefore, X is a 13-bit number whose most significant bit is w and the least significant 12 bits of X is d . This example demonstrates the ease with which (d, w) can be obtained from X . Therefore, using (d, w) as a logical address does not cause any problems.

4 ERROR DETECTION IN ADDRESS TRANSLATION

Let (r_1, r_2, \dots, r_n) be the residue representation of an integer x , defined by the moduli (m_1, m_2, \dots, m_n) . This implies that $r_i = |x|_{m_i}$, $0 \leq i \leq n$. Every residue representation need not correspond to an integer if the moduli in a residue number system (RNS) are not pairwise relatively prime. For example, consider the moduli $m_1 = 6$ and $m_2 = 8$, which are not relatively prime. The range of numbers that can be expressed using these moduli is $\{0, \dots, M - 1\}$, where $M = lcm(m_1, m_2) = 24$ (lcm is least common multiple). The residue representation $(r_1, r_2) = (3, 4)$ does not correspond to any integer in the range $\{0, \dots, 23\}$. It can be easily shown that a set of residues is consistent if and only if (see [8] pp. 30-32),

$$|r_i|_k = |r_j|_k \tag{1}$$

for all i and j and where $k = gcd(m_i, m_j)$. $|r_i|_k$ represents the residue of r_i with respect to k and "gcd(m_i, m_j)" is the greatest common divisor of m_i and m_j . r_i and r_j are residues with respect to some moduli m_i and m_j . In the above example, $gcd(6, 8) = 2$, and the residue representation (3, 4) does not correspond to any integer as $|3|_2 \neq |4|_2$. Therefore, erroneous residue representations can be detected simply by residue calculation.

We now consider another example that demonstrates how to check for consistency of a set of residues. Given the moduli, $m_1 = 4$, $m_2 = 15$, $m_3 = 36$, and $m_4 = 48$, is the residue representation (2, 6, 30, 42) consistent? The greatest common divisors of a pair of any two moduli are, $gcd(4, 15) = 1$, $gcd(4, 36) = 4$, $gcd(4, 48) = 4$, $gcd(15, 36) = 3$, $gcd(15, 48) = 3$, and $gcd(36, 48) = 12$. Using (1) to check for consistency, we get, $|2|_4 = |30|_4$, $|2|_4 = |42|_4$, $|6|_3 = |30|_3$, $|6|_3 = |42|_3$, and $|30|_{12} = |42|_{12}$. Therefore, the residue representation (2, 6, 30, 42) is consistent. Before we deal with error detection in address translation, we introduce some definitions and previously known results [9].

4.1 Definitions and Previous Results

If a set of moduli (m_1, m_2, \dots, m_n) have common factors, then every residue representation (r_1, r_2, \dots, r_n) need not correspond to an integer. Every residue representation that corresponds to an integer is referred to as a **codeword**.

DEFINITION 1. The **distance** between two residue representations (codewords) equals the number of positions in which the two codewords differ. For example, the distance between (1, 2, 5, 9) and (1, 6, 0, 9) is 2.

DEFINITION 2. The **code-distance** is the minimum of the distances between all pairs of codewords.

It is necessary and sufficient that the code-distance be at least d , in order to detect $(d - 1)$ or less residue errors [10]. t or fewer residue errors can be detected and corrected if and only if the code-distance is greater than $(2t + 1)$ [10]. If the code-distance is greater than or equal to $(t + d + 1)$, then any combination of t errors can be corrected and up to d ($d \geq t$) errors can be detected [10]. For a set of n moduli, the range of integers that can be represented is $\{0, \dots, M - 1\}$ where $M = lcm(m_1, m_2, \dots, m_n)$.

DEFINITION 3. The **cycle number**, c_i , of each modulus m_i is given by $c_i = M/m_i$.

4.2 Moduli with Relatively Prime Cycle Numbers

We now show how to construct a moduli set with pairwise relatively prime cycle numbers. This construction will assist us in obtaining efficient error detection and correction algorithms in the residue representation. Note that the moduli correspond to the number of memory modules and the number of locations in each module and the residue representation corresponds to a physical address. The logical address corresponds to the integer x which has a certain residue representation (its physical address).

Construction 1:

- 1) Choose n relatively prime numbers, none of which equals 1, that represent the cycle numbers, c_1, c_2, \dots, c_n of the moduli m_1, m_2, \dots, m_n .
- 2) Then,

$$m_i = \prod_{j=1}^{i-1} c_j \prod_{j=i+1}^n c_j$$

The moduli are the n distinct products of $(n - 1)$ cycle numbers.

The following two properties and theorem describe moduli that are obtained using Construction 1 above [9].

PROPERTY 1. The lcm of every pair of moduli are equal.

PROPERTY 2. The gcd of every pair of moduli are distinct.

THEOREM 3. Construction 1 results in a code-distance of $(n - 1)$, where n is the number of moduli.

We shall restrict ourselves to single error detection only. Therefore, the code-distance must be 2. From Theorem 3, this implies that the number of moduli must be at least three. Our previous examples of Tables 1 and 2 had only two moduli, p and m . We now present two approaches to map logical addresses to physical addresses that results in a simple error detection algorithm for the address translation. The main advantage of our approach is that error detection involves checking for the consistency of a residue representation (physical address). The methods presented do not involve the conversion of the residue representation (physical address) to an integer, using the CRT. The methods presented are therefore very fast and work well in real time.

4.3 Approach 1

This approach is similar to the approach taken in Section 3. Let p denote the number of memory modules and let m denote the number of memory locations in each memory module. Let $G = gcd(p, m)$ and $L = lcm(p, m)$. The logical address (d, w) ($0 \leq w < G$, $0 \leq d < L$) is mapped to a physical address (r_1, r_2, r_3) , as follows:

$$r_1 = d \bmod p, r_2 = (d + w) \bmod m, r_3 = d \bmod q.$$

$(p, m, q) = (m_1, m_2, m_3)$ form a moduli set obtained using Construction 1. r_1 denotes the module number, r_2 denotes the memory location number in module r_1 , and r_3 is the redundant part of the physical address. We now present an error detection algorithm for address translation.

Error Detection Algorithm:

- 1) Check if any of the residues r_i in a physical address (r_1, r_2, r_3) are out-of-range. If so, r_i is erroneous. A residue r_i is out-of-range if $r_i \geq m_i$ for $i = 1, 3$. r_2 is out-of-range if $(r_2 - w) \geq m_2$ or if $(r_2 - w) \leq -m_2$.
- 2) Check the consistency of the physical address. If it is not consistent, an error exists. The consistency of a physical address (r_1, r_2, r_3) can be checked by showing that the following equalities hold,

$$|r_1|_{k_1} = \left| \left((r_2 - w) \right) \right|_{m_2|_{k_1}}, \text{ where } k_1 = \gcd(m_1, m_2) \quad (2)$$

$$|r_1|_{k_2} = |r_3|_{k_2}, \text{ where } k_2 = \gcd(m_1, m_3) \quad (3)$$

$$\left| \left((r_2 - w) \right) \right|_{m_2|_{k_3}} = |r_3|_{k_3}, \text{ where } k_3 = \gcd(m_2, m_3) \quad (4)$$

EXAMPLE. Let the moduli set using Construction 1 be $(p, m, q) = (m_1, m_2, m_3) = (6, 10, 15)$. This system has six memory modules and each memory module has 10 memory locations. Consider a logical address $(d, w) = (17, 1)$. Note that $0 \leq d < 30$ and $0 \leq w < 2$. This logical address is mapped to the physical address $(r_1, r_2, r_3) = (5, 8, 2)$. Let us assume that the physical address computed during address translation is $(5, 7, 2)$. The physical address therefore has a single error. This error can be detected by checking for the consistency of the physical address. The first and the third equalities below are not satisfied and hence an error is detected.

$$|5|_2 = | |7-1|_{10} |_2$$

$$|5|_3 = |2|_3$$

$$| |7-1|_{10} |_5 = |2|_5$$

It should be noted that at least one of the equalities will not be satisfied in the case of a single error.

4.4 Approach 2

Approach 1 has the problem that the physical address (r_1, r_2, r_3) is too long as r_3 is redundant. We remedy this situation in this approach. Let (m_1, m_2, m_3) be the moduli set. Here m_1 is the number of memory modules, and $(m_2 \times m_3)$ is the number of memory locations in each memory module. The moduli set is obtained using Construction 1. The logical address is now given by (d_1, d_2, d_3) , where $0 \leq d_1 < \text{lcm}(m_1, m_2, m_3)$, $0 \leq d_2 < \gcd(m_1, m_2)$, and $0 \leq d_3 < \gcd(m_1, m_3) \times \gcd(m_2, m_3)$. Every logical address (d_1, d_2, d_3) , corresponds to a physical address (u_1, u_2, u_3) given by,

$$u_1 = d_1 \bmod m_1, u_2 = (d_1 + d_2) \bmod m_2, u_3 = (d_1 + d_3) \bmod m_3.$$

u_1 is the module number. Each module can be thought of as being a two-dimensional array of memory locations with m_2 columns and m_3 rows. In this case, u_2 specifies the column number and u_3 specifies the row number of a memory location in a memory module. One could also think of a system with the same parameters as above but with $(m_1 \times m_2)$ memory modules and m_3 words per module. In this case, u_1 and u_2 would locate a memory module and u_3 would locate a word in memory.

THEOREM 4. For the proposed nonprime memory system,

- 1) Every logical address (d_1, d_2, d_3) , with $0 \leq d_1 < \text{lcm}(d_1, d_2, d_3)$, $0 \leq d_2 < \gcd(m_1, m_2)$, and $0 \leq d_3 < \gcd(m_1, m_3) \times \gcd(m_2, m_3)$, corresponds to one and only one physical address (u_1, u_2, u_3) , $0 \leq u_1 < m_1$, $0 \leq u_2 < m_2$, $0 \leq u_3 < m_3$, and
- 2) Every physical address (u_1, u_2, u_3) corresponds to one and only one logical address (d_1, d_2, d_3) .

PROOF. The theorem follows directly from the mapping scheme and the CRT. \square

The single error detection algorithm is similar to the previously discussed algorithm.

Error Detection Algorithm:

- 1) Check if any of the residues u_i in a physical address (u_1, u_2, u_3) are out-of-range. If so, u_i is erroneous. The residue u_1 is out-of-range if $u_1 \geq m_1$. u_i ($i = 2, 3$) is out-of-range if $(u_i - d_i) \geq m_i$ or if $(u_i - d_i) \leq -m_i$.
- 2) Check the consistency of the physical address. If it is not consistent, an error exists. The consistency of a physical address (u_1, u_2, u_3) can be checked by showing that the following equalities hold,

$$|u_1|_{k_1} = \left| \left((u_2 - d_2) \right) \right|_{m_2|_{k_1}}, \text{ where } k_1 = \gcd(m_1, m_2) \quad (5)$$

$$|u_1|_{k_2} = \left| \left((u_3 - d_3) \right) \right|_{m_3|_{k_2}}, \text{ where } k_2 = \gcd(m_1, m_3) \quad (6)$$

$$\left| \left((u_2 - d_2) \right) \right|_{m_2|_{k_3}} = \left| \left((u_3 - d_3) \right) \right|_{m_3|_{k_3}}, \text{ where } k_3 = \gcd(m_2, m_3) \quad (7)$$

Approach 1 computes the module number as $d \bmod p$ and Approach 2 computes the module number as either $d_1 \bmod m_1$, or $d_1 \bmod m_1$ and $(d_1 + d_2) \bmod m_2$. The way to compute the module number in Approach 2 depends on whether there are m_1 memory modules or $(m_1 \times m_2)$ memory modules in the system being considered. Therefore, Theorem 2 is valid for both the approaches as two consecutive memory locations have different d_1 in Approach 1 and two consecutive memory locations have different d_1 or d_2 in Approach 2. Here we have assumed that there are $(m_1 \times m_2)$ memory modules in Approach 2.

We now show that the nonprime memory system proposed results in more efficient error detection/correction algorithms than the prime memory system. In a prime-memory system, the moduli (m_1, m_2, m_3) would be pairwise relatively prime. For error detection/correction, we would have to use procedures used in the redundant residue number system. We have already shown in [9] that these procedures result in complex time consuming algorithms as compared to the algorithms derived from Construction 1. The Chinese Remainder Theorem has to be used to convert the residue representation to an integer representation, if the error detection/correction process uses the redundant residue number system. This makes the error detection/correction process more complex. The proposed algorithms do not have to perform the above conversion and hence are more efficient. The proposed algorithms only need residue computation which can be performed very efficiently.

5 SINGLE ERROR CORRECTION

We now consider single error correction for address translation. For single error correction, the code-distance must be 3, and therefore four moduli (m_1, m_2, m_3, m_4) , are needed. These moduli are obtained using Construction 1. m_1 corresponds to the number of memory modules, m_2 corresponds to the number of memory locations per module, and m_3 and m_4 are redundant moduli. The logical address (d, w) ($0 \leq w < G$, $0 \leq d < L$) is mapped to a physical address (u_1, u_2, u_3, u_4) , as follows,

$$u_1 = d \bmod m_1, u_2 = (d + w) \bmod m_2, u_3 = d \bmod m_3, u_4 = d \bmod m_4.$$

In the above sentence, $G = \gcd(m_1, m_2)$ and $L = \text{lcm}(m_1, m_2)$. u_1 denotes the module number, u_2 denotes the memory location in a module, and u_3 and u_4 are redundant residues. Convert the physical address into a new set of residues (r_1, r_2, r_3, r_4) given by:

$$r_1 = u_1, \quad r_2 = |u_2 - w|_{m_2}, \quad r_3 = u_3, \quad r_4 = u_4.$$

We now present error detection, location, and correction algorithms for the residue representation (r_1, r_2, r_3, r_4) . The algorithms presented are similar to those developed by us in [9].

Error Detection Algorithm:

- 1) Check if any of the residues r_i in a residue representation (r_1, r_2, r_3, r_4) are out-of-range. If so, r_i is erroneous. A residue r_i is out-of-range if $r_i \geq m_i$.
- 2) Check the consistency of the residue representation. If it is not consistent, an error exists. The consistency of a residue representation (r_1, r_2, r_3, r_4) can be checked by showing that the following equalities hold,

$$|r_1|_{k_1} = |r_2|_{k_1}, \quad \text{where } k_1 = \gcd(m_1, m_2) \quad (8)$$

$$|r_1|_{k_2} = |r_3|_{k_2}, \quad \text{where } k_2 = \gcd(m_1, m_3) \quad (9)$$

$$|r_1|_{k_3} = |r_4|_{k_3}, \quad \text{where } k_3 = \gcd(m_1, m_4) \quad (10)$$

$$|r_2|_{k_4} = |r_3|_{k_4}, \quad \text{where } k_4 = \gcd(m_2, m_3) \quad (11)$$

$$|r_2|_{k_5} = |r_4|_{k_5}, \quad \text{where } k_5 = \gcd(m_2, m_4) \quad (12)$$

$$|r_3|_{k_6} = |r_4|_{k_6}, \quad \text{where } k_6 = \gcd(m_3, m_4) \quad (13)$$

Theorem 5 [9] below will aid in the location of the erroneous residue.

THEOREM 5. *When there are four moduli (m_1, m_2, m_3, m_4) , which are obtained using Construction 1, a single error in a residue representation will result in at least two and at most three of the equalities in (8) through (13) being false. Furthermore, if r_i is in error, then the equalities that are not satisfied are those that involve r_i .*

The above theorem leads us to the following error location algorithm.

Error Location Algorithm:

If r_i is in error, then the equalities that are not satisfied have the following form:

$$|r_i|_{k_a} = |r_j|_{k_a}, \quad \text{where } k_a = \gcd(m_i, m_j) \quad (14)$$

$$|r_i|_{k_b} = |r_k|_{k_b}, \quad \text{where } k_b = \gcd(m_i, m_k) \quad (15)$$

In the above equations, $i \neq j \neq k$. Therefore, the residue that is common to the equalities that are not satisfied is the erroneous residue.

We now describe the error correction algorithm.

Error Correction Algorithm:

Let us assume that r_i is in error and the corrected value of r_i is \bar{r}_i . The corrected value of r_i can be computed by solving the following equations,

$$|\bar{r}_i|_{k_a} = |r_j|_{k_a}, \quad \text{where } k_a = \gcd(m_i, m_j) \quad (16)$$

$$|\bar{r}_i|_{k_b} = |r_k|_{k_b}, \quad \text{where } k_b = \gcd(m_i, m_k) \quad (17)$$

The above equations can be solved by forming two sets A and B (defined below). Then $\{\bar{r}_i\} = A \cap B$. The sets A and B are defined below,

$$A = \left\{ f = |r_j|_{k_a} + k_a k_x : f < m_i, k_x \geq 0, k_x \text{ is an integer} \right\}$$

$$B = \left\{ g = |r_k|_{k_b} + k_b k_y : g < m_i, k_y \geq 0, k_y \text{ is an integer} \right\}$$

In a similar manner, one could extend Approach 2 for single error correction. Since the algorithms are similar to the ones presented above, we do not present the details here.

6 CONCLUSION

We have presented a technique that maps logical addresses to physical addresses in a memory system with several memory banks. The mapping scheme presented does not require that the number of memory modules be prime. The scheme also does not waste any memory capacity and avoids Euclidean Division during address translation. Another major advantage of the scheme is that it results in efficient error detection/correction algorithms for address translation. The algorithms do not require the conversion of a residue representation to an integer, which requires extensive amounts of computation. The algorithms instead rely on the computation of residues and hence are good for real-time applications.

REFERENCES

- [1] P. Budnik and D.J. Kuck, "The Organization and Use of Parallel Memories," *IEEE Trans. Computers*, vol. 20, no. 12, pp. 1,566-1,569, Dec. 1971.
- [2] D.J. Kuck and R. Stokes, "The Burroughs Scientific Processor," *IEEE Trans. Computers*, vol. 31, no. 5, pp. 363-376, May 1982.
- [3] D.H. Lawrie and C.R. Vora, "The Prime Memory System for Array Access," *IEEE Trans. Computers*, vol. 31, no. 5, pp. 435-442, May 1982.
- [4] M. Balakrishnan, R. Jain, and C.S. Raghavendra, "On Array Storage for Conflict-Free Access for Parallel Processors," *Proc. 17th Int'l Conf. Parallel Processing*, pp. 103-107, 1988.
- [5] K. Kim and V.K. Kumar, "Perfect Latin Squares and Parallel Array Access," *Proc. 16th Int'l Symp. Computer Architecture*, pp. 372-379, 1989.
- [6] C.J. Colbourn and K. Heirich, "Conflict-Free Access to Parallel Memories," *J. Parallel and Distributed Computing*, vol. 14, pp. 193-200, 1992.
- [7] Q.S. Gao, "The Chinese Remainder Theorem and the Prime Memory System," *Proc. 20th Int'l Symp. Computer Architecture*, pp. 337-340, 1993.
- [8] N.S. Szabo and R.I. Tanaka, *Residue Arithmetic and its Application to Computer Technology*. New York: McGraw-Hill, 1967.
- [9] R.S. Katti, "A New Residue Arithmetic Error Correction Scheme," *IEEE Trans. Computers*, vol. 45, no. 1, pp. 13-19, Jan. 1996.
- [10] T.R.N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*. Prentice Hall, 1989.
- [11] A. Avizienis, G.C. Gilley, F.P. Mathur, D.A. Rennels, J.A. Rohr, and D.K. Rubin, "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," *IEEE Trans. Computers*, vol. 20, no. 11, pp. 1,312-1,321, Nov. 1971.
- [12] G.L. Feng, T.R.N. Rao, and M.S. Kolluru, "Error Correcting Codes Over Z_{2^m} for Algorithm-Based Fault Tolerance," *IEEE Trans. Computers*, vol. 43, no. 3, pp. 370-374, Mar. 1994.
- [13] A. Seznec and J. Lenfant, "Odd Memory Systems May Be Quite Interesting," *Proc. 20th Int'l Symp. Computer Architecture*, pp. 341-350, 1993.