Computers Math. Applic. Vol. 22, No. 8, pp. 63–71, 1991
Printed in Great Britain.

0097-4943/91 $3.00 + 0.00

AN IMPROVED ALGORITHM FOR MIXED-RADIX CONVERSION OF RESIDUE NUMBERS*

ÇETIN K. KOÇ
Department of Electrical Engineering
University of Houston, Houston, TX 77204

(Received February 1991)

Abstract — The mixed-radix representation of a residue number with respect to n moduli can be computed in $O(n^2)$ arithmetic operations. We present a new algorithm based on the partitioning of the moduli set into q disjoint subsets, such that the product of the moduli in each subset is less than the largest positive integer representable by the computer. We show that the new algorithm requires less than $O(n^2)$ arithmetic steps whenever the number of subsets is strictly less than n. In case the partitioning produces equal-size subsets and $q = \sqrt{n}$, only $O(n^{1.5})$ arithmetic steps are required. Experiments performed on a scientific workstation indicate significant speedup even for small values of n.

1. INTRODUCTION

Since addition, subtraction and multiplication can be performed without carry propagation, residue number systems (RNS) have advantages over weighted number systems for doing arithmetic on large integers and for implementing digital signal processing algorithms [1]. Residue arithmetic is also used to find exact solutions of linear equations with integer and rational coefficients. The reader is referred to the papers [2–6] and the books [7–10] for details on this technique which is especially useful when the matrix of the coefficients is ill-conditioned.

A drawback of RNS, however, is that a number represented in residue notation does not have magnitude information and, thus, should be converted to a weighted number system to extract this information. The conversion techniques from a residue notation to a weighted notation, and vice versa, are needed in almost all applications employing residue arithmetic. For example, in hardware implementations of signal processing algorithms, the input and output signals are usually in analog form; thus the numbers have to be represented in a weighted notation before converting to analog. For software implementation of RNS algorithms, the sign detection and magnitude comparison operations require that the numbers be converted to a weighted notation.

The conversion from a weighted representation to a residue representation is achieved by concurrent application of integer division operations, a very simple and efficient process. The conversion from a residue notation to a weighted notation is, however, not that simple. The methods for this conversion are based on two different constructive proofs of the *Chinese Remainder Theorem*. In the first case, the number is converted to a *single-radix* weighted number system; in the second case, it is converted to a *mixed-radix* weighted number system [7]. The mixed-radix conversion algorithm given in the book by Szabo and Tanaka [11] is attributed to Garner [12] (see [7, 8]). Henceforth, we refer to this algorithm as the *Garner Algorithm*.

We define W as the largest positive integer which can be operated on by the arithmetic unit of the computer. When each of the moduli is less than W, Garner's algorithm requires $O(n^2)$ arithmetic steps [8]. However, with the emergence of 32-bit and 64-bit microprocessors and numeric coprocessors, which are capable of operating on integers as large as 2^{64}, it is more likely

*An earlier version of this article was presented in the *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Cambridge, Massachusetts, October 2–4, 1989.

that W is significantly larger than each of the moduli; therefore, Garner's algorithm does not fully utilize the arithmetic unit of the computer. Here we present a new algorithm based on the partitioning of the coefficient matrix when the mixed-radix conversion problem is cast as a set of linear congruent equations. Equivalently, the new algorithm makes use of the partitioning of the moduli set into disjoint subsets, such that the product of the moduli in each subset is less than W. We show that, when there is a nontrivial partitioning (i.e., the number of disjoint subsets is strictly less than n), the algorithm requires less than $O(n^2)$ arithmetic operations.

This paper is organized as follows: in Section 2, we give the Garner Algorithm and show that it requires $O(n^2)$ arithmetic operations. In Section 3, we present the *improved* Garner Algorithm, which converts a large mixed-radix conversion problem into several mixed-radix problems of smaller size. We prove that the partitioning approach yields an algorithm which requires $O(n^{1.5})$ arithmetic operations when the moduli set partitions into equal-size subsets and $q = \sqrt{n}$. In Section 4, we give some examples of the partitioning process and briefly summarize experiments performed on a scientific workstation which uses 32-bit binary integer representation. Finally, in Section 5, we discuss the issues regarding a practical implementation of this algorithm and present a simple partitioning strategy.

2. THE GARNER ALGORITHM

We are given:

- the moduli set $\{m_1, m_2, \ldots, m_n\}$, consisting of n pairwise relatively prime numbers, and
- the residues of a weighted number u with respect to each modulus

$$u_i = u \pmod{m_i}, \quad \text{for} \quad 1 \leq i \leq n.$$

The number u then can be computed using Garner's algorithm as follows:

Step 1. Compute the constants c_{ij} for $1 \leq i < j \leq n$, where

$$c_{ij}\, m_i = 1 \pmod{m_j}.$$

Step 2. Compute

$$\begin{aligned}
v_1 &= u_1 \pmod{m_1}, \\
v_2 &= (u_2 - v_1)\, c_{12} \pmod{m_2}, \\
v_3 &= ((u_3 - v_1)\, c_{13} - v_2)\, c_{23} \pmod{m_3}, \\
&\vdots \\
v_n &= (\cdots((u_n - v_1)\, c_{1n} - v_2)\, c_{2n} - \cdots - v_{n-1})\, c_{n-1,n} \pmod{m_n}.
\end{aligned}$$

Thus, the number u given in residue representation $(u_1, u_2, \ldots, u_n)$ now can be represented in mixed-radix notation as $(v_1, v_2, \ldots, v_n)$. This representation of u has magnitude information, since

$$u = v_1 + v_2\, m_1 + v_3\, m_1\, m_2 + \cdots + v_n\, m_1\, m_2 \cdots m_{n-1}. \tag{1}$$

The constants c_{ij} are the multiplicative inverses of m_i modulo m_j, for all $1 \leq i < j \leq n$, and can be computed using Euclid's algorithm (see [7, 8]). We denote the operation to find the multiplicative inverse of a modulo b by $\texttt{INVERSE}(a, b)$.

For the timing analysis of the algorithms on a single processor machine, we assume the following:

- an arithmetic operation ($\in \{+, -, \times\}$) on numbers less than W taking 1 unit time, defined as 1 *arithmetic step*,
- for operations on numbers larger than W, we implement a multi-precision arithmetic.

Thus, we see that according to this model:

1. If $a, b < W$, then Euclid's algorithm requires $O(\log W)$ arithmetic operations to compute the multiplicative inverse of a modulo b. Since W is independent of n, we assume the computation of **INVERSE**(a, b) on single-precision numbers a and b requires $O(1)$ arithmetic operations.

2. If $a, b < W$, then $a \bmod b$ can be computed in $O(1)$ time since it can be achieved by a division operation.

3. If v_i, $m_i < W$, for $1 \leq i \leq n$, then the application of Horner's algorithm to compute single-radix representation of u requires $O(n^2)$ arithmetic steps using multi-precision arithmetic [8].

We define V_{ij} for $0 \leq i < j \leq n$, such that $V_{0j} = u_j$ for $1 \leq j \leq n$, and $V_{i-1,i} = v_i$ for $1 \leq i \leq n$. V_{ij}, for $1 \leq i < j \leq n$, are the temporary values of v_j resulting from the operations in Step 2. This way, we build a lower triangular table of values with diagonal entries $v_i = V_{i-1,i}$, for $2 \leq i \leq n$. The entries of this table are named the *multiplied differences* [8]. For $n = 4$, it can be given as follows:

$$V_{12} = (V_{02} - V_{01})\, c_{12} \pmod{m_2}$$
$$V_{13} = (V_{03} - V_{01})\, c_{13} \pmod{m_3} \quad V_{23} = (V_{13} - V_{12})\, c_{23} \pmod{m_3}$$
$$V_{14} = (V_{04} - V_{01})\, c_{14} \pmod{m_4} \quad V_{24} = (V_{14} - V_{12})\, c_{24} \pmod{m_4} \quad V_{34} = (V_{24} - V_{23})\, c_{34} \pmod{m_4}$$

The following procedure computes all V_{ij}, for $1 \leq i < j \leq n$.

```
PROCEDURE GARNER(u_i, m_i; 1 ≤ i ≤ n)
FOR j = 1 TO n DO
    BEGIN
    V_0j = u_j
    END FOR
FOR i = 1 TO n − 1 DO
    BEGIN
    FOR j = i + 1 TO n DO
        BEGIN
        c_ij = INVERSE(m_i, m_j)
        V_ij = (V_{i−1,j} − V_{i−1,i}) c_ij   (mod m_j)
        END FOR
    END FOR
FOR i = 1 TO n DO
    BEGIN
    v_i = V_{i−1,i}
    END FOR
END PROCEDURE
```

THEOREM 1. *Given the moduli $m_1, m_2, \ldots, m_n$ and the remainders $u_1, u_2, \ldots, u_n$, such that $m_i < W$, for $1 \leq i \leq n$, the mixed-radix number representation $(v_1, v_2, \ldots, v_n)$ of u can be computed in $O(n^2)$ arithmetic steps with Garner's algorithm.*

PROOF. Garner's algorithm computes the terms V_{ij}, for $1 \leq i < j \leq n$, by performing the following operations on single-precision operands:

$$\begin{aligned} c_{ij} &= \textbf{INVERSE}(m_i, m_j), \\ V_{ij} &= (V_{i-1,j} - V_{i-1,i})\, c_{ij} \pmod{m_j}. \end{aligned}$$

There are exactly $n(n-1)/2$ entries in the table. Each entry is computed using $O(1)$ arithmetic operations: **INVERSE**, subtraction and multiplication on single-precision numbers. Thus, the total number of arithmetic operations required for the computation of multiplied differences by Garner's algorithm is $O(n^2)$. ∎

3. THE IMPROVED GARNER ALGORITHM

In this section, we formulate the mixed-radix conversion problem as a system of linear congruence equations and show that it can be partitioned into smaller size mixed-radix conversion problems and some additional arithmetic operations.

Recall equation (1)

$$u = v_1 + v_2 m_1 + v_3 m_1 m_2 + \cdots + v_n m_1 m_2 \cdots m_{n-1}.$$

Coefficients v_i, for $1 \leq i \leq n$, can be obtained by solving

$$\begin{array}{lllllllll}
v_1 & & & & & & & = & u_1 \pmod{m_1} \\
v_1 & + & v_2 m_1 & & & & & = & u_2 \pmod{m_2} \\
v_1 & + & v_2 m_1 & + & v_3 m_1 m_2 & & & = & u_3 \pmod{m_3} \\
\vdots & & & & & & & \vdots & \\
v_1 & + & v_2 m_1 & + & v_3 m_1 m_2 & + \cdots + & v_n m_1 m_2 \cdots m_{n-1} & = & u_n \pmod{m_n}.
\end{array}$$

We represent the above linear system of equations in matrix notation as

$$\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
1 & m_1 & 0 & 0 & \cdots & 0 \\
1 & m_1 & m_1 m_2 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & m_1 & m_1 m_2 & m_1 m_2 m_3 & \cdots & m_1 m_2 m_3 \cdots m_{n-1}
\end{bmatrix}
\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix}
=
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix}
\pmod{\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_n \end{bmatrix}},$$

or more compactly,

$$\mathbf{M}\mathbf{v} = \mathbf{u} \pmod{\mathbf{m}}. \tag{2}$$

This system is solved by applying Garner's algorithm to the integers $u_1, u_2, \ldots, u_n$ using the moduli set $m_1, m_2, \ldots, m_n$. Taking

$$n = k_1 + k_2 + \cdots + k_q,$$

we define the quantities l_i as

$$l_i = k_1 + k_2 + \cdots + k_i, \quad \text{for} \quad 1 \leq i \leq q \quad \text{and} \quad l_0 = 0.$$

The moduli set $\mathcal{M}$ is partitioned into q disjoint subsets $\mathcal{M}_i$, where the cardinality of $\mathcal{M}_i$ is equal to k_i, such that the product of the moduli in each subset $\mathcal{M}_i$ is less than W, for $1 \leq i \leq q$, i.e.,

$$\mathcal{M}_i = \{m_{l_{i-1}+1}, m_{l_{i-1}+2}, \ldots, m_{l_i}\},$$

with

$$\mu_i = m_{l_{i-1}+1} \cdot m_{l_{i-1}+2} \cdots m_{l_i} < W,$$

for $1 \leq i \leq q$. Equivalently, the matrix $\mathbf{M}$ is partitioned into $k_i \times k_j$-dimension matrices $\mathbf{M}_{ij}$ for $1 \leq j \leq i \leq q$, and vectors $\mathbf{v}$, $\mathbf{u}$ and $\mathbf{m}$ into k_i-dimension vectors $\mathbf{v}_i, \mathbf{u}_i, \mathbf{m}_i$, for $1 \leq i \leq q$. Thus, we write (2) as

$$\begin{bmatrix}
\mathbf{M}_{11} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\
\mathbf{M}_{21} & \mathbf{M}_{22} & \mathbf{0} & \cdots & \mathbf{0} \\
\mathbf{M}_{31} & \mathbf{M}_{32} & \mathbf{M}_{33} & \cdots & \mathbf{0} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\mathbf{M}_{q1} & \mathbf{M}_{q2} & \mathbf{M}_{q3} & \cdots & \mathbf{M}_{qq}
\end{bmatrix}
\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \vdots \\ \mathbf{v}_q \end{bmatrix}
=
\begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_q \end{bmatrix}
\pmod{\begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \\ \vdots \\ \mathbf{m}_q \end{bmatrix}}, \tag{3}$$

where the matrices $\mathbf{M}_{ii}$ are lower-triangular and the zero matrices are of the same dimension as the $\mathbf{M}_{ij}$ matrices. This partitioning of the problem yields an algorithm which solves (2) by applying forward block-substitution to the linear congruent system (3), i.e., we first solve

$$\mathbf{M}_{11}\,\mathbf{v}_1 = \mathbf{u}_1 \pmod{\mathbf{m}_1}$$

using the procedure **GARNER**, and then update the values of $\mathbf{u}_i$, for $2 \le i \le q$, using

$$\mathbf{u}_i' = \mathbf{u}_i - \mathbf{M}_{21}\,\mathbf{v}_1 \pmod{\mathbf{m}_i}.$$

The matrix $\mathbf{M}_{21}$ consists of k_2 copies of the row vector

$$\left[\ 1 \quad m_1 \quad m_1 m_2 \quad \cdots \quad m_1 m_2 \cdots m_{k_1-1}\ \right],$$

thus, the matrix operation $\mathbf{M}_{21}\,\mathbf{v}_1$ requires the computation of

$$s_1 = v_1 + v_2 m_1 + \cdots + v_{k_1} m_1 m_2 \cdots m_{k_1-1}.$$

Since all $v_i, m_i < W$, for $1 \le i \le k_1$, and the maximum value s_1 can attain is

$$(m_1 - 1) + m_1 (m_2 - 1) + \cdots + m_1 m_2 \cdots m_{k_1-1} (m_{k_1} - 1) = m_1 m_2 m_3 \cdots m_{k_1} - 1 < W,$$

the computation of s_1 requires single-precision arithmetic. We then compute the new values of $\mathbf{u}_i$ for $2 \le i \le q$, or equivalently, compute u_i' for $k_1 + 1 \le i \le n$, using

$$u_i' = u_i - s_1 \pmod{m_i}.$$

The next step is to compute $\mathbf{v}_2$ by solving

$$\mathbf{M}_{22}\,\mathbf{v}_2 = \mathbf{u}_2' \pmod{\mathbf{m}_1}.$$

This requires some initial processing which further updates u_i, for $k_1 + 1 \le i \le n$. We then apply the procedure **GARNER** to solve for $\mathbf{v}_2$. The improved Garner Algorithm given below computes all $\mathbf{v}_i$, for $1 \le i \le q$.

```
        PROCEDURE IMPROVED GARNER (u_i, m_i; 1 ≤ i ≤ n)
        FOR i = 1 TO q DO
            BEGIN
1:          (v_j; l_{i-1} + 1 ≤ j ≤ l_i) = GARNER(u_j, m_j; l_{i-1} + 1 ≤ j ≤ l_i)
2:          s_i = v_{l_{i-1}+1} + v_{l_{i-1}+2} · m_{l_{i-1}+1} + ··· + v_{l_i} · m_{l_{i-1}+1} · m_{l_{i-1}+2} ··· m_{l_i-1}
            FOR j = l_i + 1 TO n DO
                BEGIN
3:              u_j = u_j - s_i  (mod m_j)
                END FOR
4:          t_i = m_{l_{i-1}+1} · m_{l_{i-1}+2} ··· m_{l_i}
            FOR j = l_i + 1 TO n DO
                BEGIN
5.1:            t_ij = INVERSE(t_i, m_j)
5.2:            u_j = t_ij u_j  (mod m_j)
                END FOR
            END FOR
        END PROCEDURE
```

THEOREM 2. *When there is a partitioning of the moduli set $\mathcal{M}$ into disjoint subsets $\mathcal{M}_i$, such that the cardinality of $\mathcal{M}_i$ is k_i and the product of the moduli in each $\mathcal{M}_i$ is less than W, for $1 \le i \le q$, then the improved Garner Algorithm requires*

$$O(k_1^2 + k_2^2 + \cdots + k_q^2 + n\,q)$$

arithmetic steps to compute the multiplied differences v_i, for $1 \le i \le n$. Furthermore, the minimum number of arithmetic operations required by the improved Garner Algorithm is found as $O(n^{1.5})$, which is realized when $k = k_i$, for $1 \le i \le q$ and $k = q = \sqrt{n}$.

PROOF. In Step 1, the call to the procedure **GARNER** occurs q times for $i = 1, 2, \ldots, q$, where at instance i, a problem of size k_i is solved. The variables involved in the computation of $\mathbf{v}_i$ are all single-precision numbers. It follows from Theorem 1, that this step requires $O(k_i^2)$ arithmetic steps for a particular value of i.

In Step 2, we find s_i by applying Horner's algorithm to the multiplied differences computed in Step 1 and the moduli from the set $\mathcal{M}_i$. The maximum value s_i can attain is less than W, i.e., s_i is a single-precision number. Since all input variables and the temporary values resulting from the application of Horner's algorithm are in single-precision, this step requires only $O(k_i)$ arithmetic operations.

In Step 3, the j loop is executed $n - l_i$ times for a particular value of i, where at each step a single-precision subtraction (followed by a correction to bring the result back to the desired range) is performed. Thus in Step 3, $O(1)(n - l_i)$ arithmetic operations are performed. Since we have $n - l_i = k_{i+1} + k_{i+2} + \cdots + k_q$, the number of arithmetic operations becomes

$$O(1) \sum_{j=i+1}^{q} k_j. \tag{4}$$

In Step 4, the product of all moduli in the subset $\mathcal{M}_i$ can be found by performing $k_i - 1$ single-precision multiplications.

In Step 5, similar to Step 3, we execute the **FOR** loop for $n - l_i$ times where at each step $O(1)$ operations are performed, namely an **INVERSE** operation and a multiplication involving single-precision integers. Thus the number of arithmetic steps required by Step 5 is also given by (4).

For $1 \leq i \leq n$, we sum the number of arithmetic operations at each step and find the total number of arithmetic operations required by the improved Garner Algorithm as

$$\begin{aligned} T &= \sum_{i=1}^{q} \left[O(k_i^2) + O(k_i) + O(1) \sum_{j=i+1}^{q} k_j \right] = O\left(\sum_{i=1}^{q} k_i^2 + \sum_{i=1}^{q} k_i + \sum_{i=1}^{q} \sum_{j=i+1}^{q} k_j \right) \\ &= O\left(\sum_{i=1}^{q} k_i^2 + q \sum_{i=1}^{q} k_i \right) = O(k_1^2 + k_2^2 + \cdots + k_q^2 + n\,q). \end{aligned}$$

Expression $k_1^2 + k_2^2 + \cdots + k_q^2$ takes its minimum value when $k_i = k$, for $1 \leq i \leq q$. Since we have $k_1 + k_2 + \cdots + k_q = n$, it follows that $k\,q = n$. The number of arithmetic operations required by the **IMPROVED GARNER** procedure, then becomes

$$T(q) = O\left(\left(\frac{n}{q} \right)^2 q + n\,q \right) = O\left(\frac{n^2}{q} + n\,q \right).$$

The minimum value of $T(q)$ is found by solving the equation

$$\frac{d}{dq} T(q) = -\frac{n^2}{q^2} + n = 0,$$

which gives the optimum value of $q^* = \sqrt{n}$. Thus, the minimum number of arithmetic operations required by the improved Garner Algorithm is found as $O(\frac{n^2}{\sqrt{n}} + n\sqrt{n}) = O(n^{1.5})$. ∎

The single-radix representation u of the residue number $(u_1, u_2, \ldots, u_n)$ can also be computed if the temporary values s_i, t_i, for $1 \leq i \leq q$, in the procedure **IMPROVED GARNER** are saved. We compute u with

$$u = s_1 + s_2\,t_1 + s_3\,t_1\,t_2 + \cdots + s_q\,t_1\,t_2 \cdots t_{q-1},$$

which requires multiple-precision integer arithmetic. The application of Horner's algorithm using multiple-precision arithmetic requires $O(q^2)$ arithmetic steps to compute the single-radix representation of u [8]. If $q = \sqrt{n}$, then this computation takes only $O(q^2) = O(n)$ arithmetic steps. When the improved Garner Algorithm is used to compute the single-radix representation of u, the number of arithmetic operations required is dominated by $O(n^{1.5})$ under the assumptions of Theorem 2.

4. EXAMPLES AND EXPERIMENTAL RESULTS

In this section, we give some partitioning examples and briefly summarize our experiments performed on a Digital VAX station 3500 running the Ultrix operating system. Our computer is capable of operating on 32-bit signed integers, thus $W = 2^{31} - 1 = 2,147,483,647$, since one bit (the most significant bit) is reserved for the sign. Taking the moduli set as the first 16 prime numbers, i.e., $\mathcal{M} = \{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53\}$, the partitionings for the values of $q = 3,4,5$ are described below. When $\mathcal{M}$ is partitioned into two sets, the product of the moduli in at least one of the subsets exceeds W; thus, the case $q = 2$ is not feasible.

Partition $q = 3$:

$$\begin{aligned}
&\mathcal{M}_1 = \{7,11,29,31,53\}, && k_1 = 5, && \mu_1 = 7 \cdot 11 \cdot 29 \cdot 29 \cdot 31 \cdot 53 = 3,668,819;\\
&\mathcal{M}_2 = \{5,13,23,37,47\}, && k_2 = 5, && \mu_2 = 5 \cdot 13 \cdot 23 \cdot 37 \cdot 47 = 2,599,805;\\
&\mathcal{M}_3 = \{2,3,17,19,41,43\}, && k_3 = 6, && \mu_3 = 2 \cdot 3 \cdot 17 \cdot 19 \cdot 41 \cdot 43 = 3,416,694.
\end{aligned}$$

Partition $q = 4$:

$$\begin{aligned}
&\mathcal{M}_1 = \{2,19,23,53\}, && k_1 = 4, && \mu_1 = 46,322;\\
&\mathcal{M}_2 = \{3,17,29,47\}, && k_2 = 4, && \mu_2 = 69,513;\\
&\mathcal{M}_3 = \{5,13,31,43\}, && k_3 = 4, && \mu_3 = 86,645;\\
&\mathcal{M}_4 = \{7,11,37,41\}, && k_4 = 4, && \mu_4 = 116,809.
\end{aligned}$$

Partition $q = 5$:

$$\begin{aligned}
&\mathcal{M}_1 = \{13,17,53\}, && k_1 = 3, && \mu_1 = 11,713; && \mathcal{M}_2 = \{11,19,47\}, && k_2 = 3, && \mu_2 = 9,823;\\
&\mathcal{M}_3 = \{7,23,43\}, && k_3 = 3, && \mu_3 = 6,923; && \mathcal{M}_4 = \{5,29,41\}, && k_4 = 3, && \mu_4 = 5,945;\\
&\mathcal{M}_5 = \{2,3,31,37\}, && k_5 = 4, && \mu_5 = 6,882.
\end{aligned}$$

According to Theorem 2, when a nontrivial partitioning exists, the improved Garner Algorithm requires fewer number of arithmetic operations and, thus, less time than Garner Algorithm. Additionally, when the set $\mathcal{M}$ can be partitioned for q very close to $\sqrt{n}$, the improved Garner Algorithm requires the least amount of time. This observation was confirmed using several different values of n and q. Table 1 shows the timing results for $n = 16, 25$ and $q = 3,4,5,6,7,8$. The procedures `GARNER` and `IMPROVED GARNER` were implemented in C language. The programs were executed to compute the mixed-radix coefficients of randomly selected residues with respect to the above moduli, and the total CPU time was measured using the built-in C routine `clock()`. The values shown in Table 1 are the average times (in milliseconds) taken by the Garner and improved Garner procedures.

Table 1. Timing results.

	Improved Garner						
	$q = 3$	$q = 4$	$q = 5$	$q = 6$	$q = 7$	$q = 8$	Garner
$n = 16$	5.04	5.12	5.50	5.66	6.46	7.51	9.64
$n = 25$	11.00	10.02	10.47	11.40	12.79	14.45	25.25

5. CONCLUSIONS

We have shown that when a partitioning of the moduli set $\mathcal{M}$ into disjoint subsets $\mathcal{M}_i$, for $1 \leq i \leq q$, exists, such that

1. the cardinality of each $\mathcal{M}_i$ is $k_i = k$,

2. the product of the moduli in each subset $\mathcal{M}_i$ is less than W, the largest number representable by the computer, and
3. $k = q = \sqrt{n}$,

then the improved Garner Algorithm computes the multiplied differences using $O(n^{1.5})$ arithmetic operations. Furthermore, the single-radix representation of the number u can also be computed by the application of Horner's algorithm in multiple-precision arithmetic using another $O(n)$ arithmetic steps. When the moduli set partitions trivially, i.e., each $\mathcal{M}_i$ contains exactly one modulus because the product of any two moduli is larger than W, then the number of arithmetic operations required by the improved Garner Algorithm becomes $O(n^2)$. Thus, a practical implementation will have the running time as $O(n^\alpha)$, where $1.5 \leq \alpha \leq 2$. Our implementation has indicated that a significant speedup can be obtained even for small values of n. For example, when $n = 25$ and $q = 5$, the improved Garner Algorithm is 2.41 times faster than the Garner Algorithm.

Several implementations of Garner Algorithm are given in the literature (see [1]). These implementations are mostly hardware oriented, and use table lookup techniques and, thus, put *a priori* restrictions on the size and cardinality of the moduli. The new algorithm described in this paper does not have such restrictions; other than the fact that the running time of the algorithm may exceed $O(n^{1.5})$.

The partitioning of an arbitrary moduli set for a large value of n may be complicated, since it is related to the *set partitioning problem* which is NP-complete (see Problem SP12 in [13, p. 223]). However, a simple strategy can be given for small values of n and m_i, for $1 \leq i \leq n$. The strategy explained below is used to produce the partitioning examples described in Section 4.

Given $m_1 < m_2 < \cdots < m_n$, we produce a table of dimension $q \times \lceil n/q \rceil$. We place the moduli $m_n, m_{n-1}, \ldots, m_{n-q+1}$ to the first column of the table starting from the first row. The second column is filled with $m_{n-q}, m_{n-q-1}, \ldots, m_{n-2q+1}$ (in that order) starting from the q^{th} row, and so on. In the end, the i^{th} row of the table contains the elements of the set $\mathcal{M}_i$, for $1 \leq i \leq q$. For example, Partition $q = 5$ in Section 4 was produced as follows: Taking $n = 16$ and $q = 5$, we find $\lceil n/q \rceil = 4$, and thus obtain the sets $\mathcal{M}_i$, for $1 \leq i \leq 5$, as

$\mathcal{M}_1$	m_{16}	m_7	m_6	
$\mathcal{M}_2$	m_{15}	m_8	m_5	
$\mathcal{M}_3$	m_{14}	m_9	m_4	
$\mathcal{M}_4$	m_{13}	m_{10}	m_3	
$\mathcal{M}_5$	m_{12}	m_{11}	m_2	m_1

We note that an earlier version of this paper is found in [14]. Also systolic implementations of the Garner Algorithm can be found in [15], where a partitioning strategy, similar to the one presented in this paper, was introduced to solve a mixed-radix conversion problem on an undersized systolic array.

REFERENCES

1. M.A. Soderstrand, W.K. Jenkins, G.A. Jullien and F.J. Taylor, Editors, *Residue Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, (1986).
2. I. Borosh and A.S. Fraenkel, Exact solutions of linear equations with rational coefficients by congruence techniques, *Mathematics of Computation* **20** (93), 107–112 (January 1966).
3. M. Newman, Solving equations exactly, *Journal of Research of the National Bureau of Standards* **71B** (4), 171–179 (October–December 1967).
4. J.A. Howell and R.T. Gregory, An algorithm for solving linear algebraic equations using residue arithmetic I–II, *BIT* **9** (3) 200–224, (4) 324–337 (1969).
5. J.A. Howell and R.T. Gregory, Solving linear equations using residue arithmetic—Algorithm II, *BIT* **10** (1) 23–37 (1970).

6. S. Cabay and T.P.L. Lam, Congruence techniques for the exact solution of integer systems of linear equations, *ACM Transactions on Mathematical Software* **3** (4), 386–397 (December 1977).
7. D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2, 2nd ed., Addison-Wesley (1981).
8. J.D. Lipson, *Elements of Algebra and Algebraic Computing*, Addison-Wesley (1981).
9. G. Mackiw, *Applications of Abstract Algebra*, John Wiley and Sons (1985).
10. D.M. Young and R.T. Gregory, *A Survey of Numerical Mathematics*, Vol. 2, Dover Publications (1988).
11. N.S. Szabo and R.I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill (1967).
12. H.L. Garner, The residue number systems, *IRE Transactions on Electronic Computers* **8** (6), 140–147 (June 1959).
13. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman (1979).
14. Ç.K. Koç, A fast algorithm for mixed-radix conversion in residue arithmetic, In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 18–21, IEEE Computer Society Press, Cambridge, Massachusetts, (October 2–4, 1989).
15. Ç.K. Koç and P.R. Cappello, Systolic arrays for integer Chinese remaindering, In *Proceedings of the 9th Symposium on Computer Arithmetic*, (M.D. Ercegovac and E. Swartzlander, Eds.), pp. 216–223, IEEE Computer Society Press, Santa Monica, California, (September 6–8, 1989).