Abstract—Chinese remainder theorem (CRT), an old and famous theorem, is widely used in many modern computer applications. The computation of CRT contains many similar operations which can be implemented concurrently. Here, a parallel algorithm implemented on the ring topology is proposed to parallelize almost all the computations in CRT and J-conditions in this paper. Some recently proposed modular arithmetic operations are stated and employed to accelerate the computation.

I. INTRODUCTION

Chinese remainder theorem (CRT), invented by an ancient Chinese mathematician, Sun-Tse, is an important theorem in number theory and is widely used for computer applications. J-conditions [9], which is similar to CRT, was introduced in 1981 and also adopted in some applications. For example, Chang proposed a key-lock-pair file management system based on CRT in 1986 [2]. In 1985, Chen and Chiou presented a method based on CRT to broadcast message securely [7]. Later, Chang and Lin proposed another secure broadcasting system based on generalized CRT in 1988 [5]. The J-conditions property is useful for developing the information protection mechanism [3] and the minimal perfect hashing scheme [9]. Although more and more emphasis has been put upon information security, CRT is never too old to be employed in modern computer cryptographics. CRT is used for accelerating the computation of RSA [14][15].

However, the computation of the applications of CRT is complex containing many arithmetic operations. Modular multiplication, modular inverse, multi-multiplication ($\prod_{i=1}^{m} n_i$) are mainly required. When the inputted integers are large, these computations become very complicated. Many algorithms were proposed to solve these problems, e.g., [4][12][13]. In this paper, we shall employ some modular arithmetic algorithms to solve this CRT problem.

It is observed that there are many similar computations in CRT and J-conditions as well. Chang and Lin have developed the Reciprocal Confluence Tree unit [6] to implement them. In their algorithm, many arithmetic units are employed for parallel processing, but it does not parallelize the entire computation and no detail computation in each unit is discussed. We introduce a parallel topology, ring, to solve the CRT problem in parallel. The ring parallel system (see Figure 1) is composed by a set of nodes arranged and linked as a ring. Each node has its own router, processor, local memory and arithmetic operations. The message passing in this system is efficient. Since there is only one path between the adjacent nodes, the message passing inhibits any collision, and each node can send and receive one message at one time only. The ring topology can be easily embedded into the famous topology, hypercube [11].

In this paper, we shall review the Montgomery reduction algorithm and some modular arithmetic operations based on it. In the section of preliminaries, a parallel modular exponentiation algorithm [8] is also introduced. Chinese remainder theorem and J-conditions are stated in Section III. Then the proposed parallel algorithm for CRT is stated in Section IV. We shall also show how to adapt the algorithms introduced in Section II in our method. Of course, the computation analysis is presented following the method. In the last section, conclusions will be presented.

Fig. 1. Ring topology.

II. PRELIMINARIES

In this section, we shall introduce some algorithms that will later be used in our parallel scheme, including the modular arithmetic and the parallel modular exponentiation algorithm.

A. Montgomery Reduction and the Fast Modular Arithmetic

The division operation is intuitionally necessary for modular arithmetic. However, a division is much more expansive than a multiplication. A modular reduction algorithm without trial divisions
was proposed by P. L. Montgomery [12] in 1985 where shifting, addition and multiplication operations instead of divisions were used to implement the modular reduction. It computes a multiplication of two integers, \( A \) and \( B \), into a residue class \( ABR^{-1} \mod N \) by an inexpensive algorithm.

Let \( N \) be a large integer that is relatively prime to \( R \), where \( R \) is an integer greater than \( N \) and the computation modulo \( R \) is easy to implement. \( N \)-residue is defined to be a residue class modulo \( N \). Computing \( ABR^{-1} \mod N \), where \( A \) and \( B \) are two inputs and \( R \) is the number of \( n \)-th power of 2, \( 2^n \), \( N < R < 2N \), by using Algorithm 1 is very simple.

### Algorithm 1. Montgomery reduction algorithm

**function** REDC\( (A, B, N) \) // \( A = (a_0, a_1, \ldots, a_{n-1}) \)

begin
\[ T = 0; \]
begin
for \( i = 0; i < n; i \) begin
\[ p_i = (B + a_i) \mod 2; \]
\[ T = (T + 2p_i) \mod 2; \]
e nd
\[ \text{if} \ (T \geq N) \text{ then return } T - N; \]
\[ \text{else return } T; \]
e nd.
\]

**B. Parallel Exponentiation**

The binary method is a well-known algorithm for fast exponentiation. In 1993, Chiou [8] proposed a fast exponentiation algorithm using parallel processing. His parallel architecture contains two processors concurrently implementing squaring and multiplication respectively. Figure 2 shows the topology connecting two nodes by a path. This two-node system can easily implement Chiou’s parallel exponentiation algorithm (expressed as Algorithm 4).

![Fig. 2. The two-node topology for parallel exponentiation](image)

The final result of exponentiation will be held in processor \( P \) by using the following algorithm.

### III. Computations Of Chinese Remainder Theorem And The J-condition

Chinese remainder theorem and the J-conditions are the problems we want to solve in this paper. We shall introduce the definition of these two problems and followed by their computations. The mathematical equations in the part of computations are all correctly proven.

#### Algorithm 2. Parallel Exponentiation Algorithm

**function** ParExp\( (X, Z, N) \)

begin
\[ C = 1; \]
\[ S = X; \]
\[ Temp = X; \]
for \( j = 0 \) to \( k - 1 \) do
begin
for processors \( P \) and \( P' \) do in parallel
begin
\[ \text{processor } P: \text{if } (a_j = 1) \text{ then } C = C \times \text{Temp mod } N; \]
\[ \text{processor } P': S = S \times \text{Temp mod } N; \]
end;
\[ \text{end}; \]
\]
end.

#### Theorem 1. Chinese Remainder Theorem [10]

Let \( n_1, n_2, \ldots, n_m \) be \( m \) pairwise co-prime numbers i.e., \( \gcd(n_i, n_j) = 1 \) when \( i \neq j \), and \( N = n_1n_2\ldots n_m \). The following residue system,

\[
\begin{align*}
&x \equiv a_1 \mod n_1, \\
&x \equiv a_2 \mod n_2, \\
&\vdots \\
&x \equiv a_m \mod n_m
\end{align*}
\]

will have only one solution under \([0, N-1]\).

#### Computations of CRT [10]

The solution of \( x \) can be computed by the following equation.

\[
x = \left( \sum_{i=1}^{m} \frac{n}{n_i} a_i \right) \mod N,
\]

where

\[
N_i \equiv 1 \mod n_i.
\]

#### Definition 1. J-conditions [9]

Let \( n_1, n_2, \ldots, n_m \) be \( m \) co-prime numbers i.e., \( \gcd(n_i, n_j) = 1 \) when \( i \neq j \). Let \( a_1, a_2, \ldots, a_m \) be \( m \) integers. Assume that \( \max\{a_i\} < m < \min\{n_i\} \), then the constant \( x \) in the residue class \( N = \prod_{i=1}^{m} a_i \) satisfies J-conditions if

\[
a_1 = \left\lfloor \frac{x}{n_1} \mod m \right\rfloor, \quad a_2 = \left\lfloor \frac{x}{n_2} \mod m \right\rfloor, \quad \ldots, \quad a_m = \left\lfloor \frac{x}{n_m} \mod m \right\rfloor.
\]

#### Computations of J-conditions [9]

\[
x = \sum_{j=1}^{m} n_i y_j M_i \mod N \quad \text{satisfies} \quad J\text{-conditions},
\]

where

\[
N_i = \prod_{j=1}^{m} a_j, \quad y_j \quad \text{satisfies}
\]
\[ N_{ij} y_i = 1 \pmod{q_j}, \quad M_i = \left\lfloor \frac{a_i n_i}{m} \right\rfloor \quad \text{and} \quad N = m \prod_{i=1}^{m} n_i \]
for \( i = 1, 2, \cdots, m \).

Here \( \left\lfloor \frac{a_i n_i}{m} \right\rfloor \) and \( \left\lceil \frac{a_i n_i}{m} \right\rceil \) denote the floor and ceiling operations, respectively.

### IV. THE PROPOSED ALGORITHM

#### A. The Parallel Topology

Since we want to parallelize most of the computations of CRT and J-conditions, the computation of a unit inverse is computed in parallel. We also adapt the two-node topology used in parallel exponentiation in each node and connect these nodes as a ring as shown in Figure 3.

Actually, we can reduce this topology into the simple ring topology with \( 2m \) processors, where \( m \) is the number of congruencies in the given CRT or J-conditions problem. The algorithms proposed will still work.

![Fig. 3. The parallel topology for CRT.](image)

#### B. Compute \( N_i \) and \( N \)

Prefix computation [11] that computes a set of integers \( a_1, a_2, \cdots, a_m \) is thoroughly discussed in parallel processing. It can be defined as follows.

Given \( a_i \), prefix computation is to compute \( R_i \) for \( i = 1, 2, \cdots, m \).

\[
R_i = \begin{cases} a_i & \text{if } i = 1, \\ R_{i-1} \otimes a_i & \text{if } i = 2, 3, \cdots, m. \end{cases}
\]

Here \( \otimes \) is an associative binary operation.

There is an efficient parallel algorithm that performs prefix computation in \( \log_2 m \) time units. Adopting the topology mentioned in Section IV.A, in the following, we propose a parallel algorithm to compute \( N \). For each processor to hold the correct \( N \), we assign \( N = n \), \( n \pmod{m} \), and \( n = n_i \) in \( P_i \) and \( N' = 1 \) in \( P_i \) for the initial state. The assignments are for the later computations.

After running Algorithm 5, each \( P_i \) holds \( N \) as well as \( n \), and each \( P'_i \) holds \( N \). Figure 4 illustrates the three-round computation and message passing for our algorithm to compute eight input items.

![Fig. 4. 8 nodes example for computing \( N \).](image)

#### C. Fast Modular Inverse Method

There are two famous algorithms, Euclidean algorithm and Euler algorithm, to compute the modular multiplicative inverse. In case that the modulus \( N \) is a prime number, Euler algorithm is more efficient and commonly used because \( \phi(N) \) is easily obtained. If the modulus is a composite number, Euclidean algorithm is more suitable since the Euler quotient cannot be easily computed under such circumstances. For common
applications of CRT, each \( N_i \) should be carefully chosen to ensure that they are all relative primes.

**Euler’s Theorem** [10]

If \( X \) and \( N \) are co-primes i.e., \( \text{gcd}(X, N) = 1 \), then \( X^{\phi(N)} \mod N = 1 \). The Euler quotient function of \( N \), \( \phi(N) \), is the number of integers not exceeding \( N \) which are co-primes of \( N \).

By Euler’s theorem, \( X^{\phi(N)} \mod N = 1 \) implies \( X \times X^{\phi(N)-1} \mod N = 1 \). It is obvious that the inverse of \( X \) under modulo \( N \) is \( X^{\phi(N)-1} \). The function \( \phi(N) \) can be easily derived if the factorization of \( N \) is known. Let \( N = N_1^{P_1} \times N_2^{P_2} \times \ldots \times N_k^{P_k} \), where \( N_i \) is a prime and \( P_i \) is a positive integer for \( 1 \leq i \leq k \), then \( \phi(N) = \prod_{i=1}^{k} (N_i^{P_i} - N_i^{P_i-1}) \). The proof can be found in most number theory textbooks.

Employing the parallel exponentiation algorithm, the modular inverse can be readily fast computed using the following pseudo-code. Notice that this algorithm is implemented on the two-node topology.

### Algorithm 4. Modular Inverse for Known Factorization Module

```plaintext
function PointedInverse\((X, N)\)
begin
    for processors \( P \) and \( P' \) do in parallel
        \( \phi(N) = 1 \);
    for \( i = 1 \) to \( \frac{1}{2} \log_2 m \) do in parallel
        processor \( P \) performs \( \phi(N) = \phi(N) \times (X^{P_i} \mod N) \);
        processor \( P' \) performs \( \phi(N) = \phi(N) \times (X^{P'_i} \mod N) \);
    end;
    processor \( P \) performs \( \text{SendMsg}(P', \text{tmp}) \);
    processor \( P' \) performs \( \text{SendMsg}(P, \text{tmp}) \);
    processor \( P \) performs \( Y = X^{\phi(N)} \mod N \);
    return \( Y \);
end.
```

This algorithm saves 50% of the time units in the computation of \( \phi(N) \) and 30% in the computation of the later exponentiation.

**D. The Proposed Algorithms**

Now, we have enough tools to implement the entire algorithm for the computation of CRT and \( J \)-conditions. The entire algorithm is composed of five parts. Part 1 compute \( N_i \) and \( N'_i \) in parallel. Then two-node topology works to the derivation of the inverses. The computation, \( y_i \mod n_i \), is performed in Part 3. Due to \( y_i < n_i \) and \( a_i < n_i \). In the last part, a parallel prefix algorithm with the addition operation is implemented to summarize all \( N_i a_i / s \).
required while our algorithm requires
\[ k^2 \sum_{i=0}^{k-1} (2^i)^2 = \frac{m^2 - 1}{3} k^2 \]
multiplication time units in Part 1.

Part 1.

Algorithm 6. The entire parallel algorithm of J-conditions computing

\textbf{Input:} \( n \), \( a \), for \( i = 1, \ldots, n \).
\textbf{Output:} \( x \).
\textbf{Begin}
\textbf{Part 1.} Parallel \( N_i \) and \( M \) Computing.
\textbf{Part 2.} Parallel \( i \) and \( j \) do in parallel

\textbf{Part 3.} Parallel \( i \) and \( j \) do in parallel

\textbf{Part 4.} for \( i = 1 \) to \( m \) do in parallel

In Parts 2 and 3, \( m \) times of \text{ParModInverse}(N, n) \) and modular operations are required respectively by a common place method while our algorithm requires one. Notice that in Part 2, a parallel modular inverse algorithm is implemented to save 30% of the time. To compute an inverse, \( 3kw/2 \) (\( w \) is the word length of the programming language) modular multiplications are required for the sequential algorithm in average since the bit length of \( n_i \) is \( kw \). In Part 2, the time for \( 3mw/2 \) modular multiplication is required if our algorithm, which requires only \( kw \) modular multiplication time, is not employed. In the last part, \( m-1 \) additions are required in the sequential algorithm and \( \log m \) additions are required in our algorithm. The computation time required for the sequential algorithm and our algorithm are listed in Table 1.

Table 1.
THE OPERATION TIME REQUIRED IN EACH PART.

<table>
<thead>
<tr>
<th>Part</th>
<th>Sequential Algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>( \frac{1}{2} (m^2 - m)k + 2(m^2 - m)k )</td>
<td>( \frac{m^2 - 1}{3} k^2 )</td>
</tr>
<tr>
<td>Part 2</td>
<td>( m^2 - m )</td>
<td>( \frac{m^2 - 1}{3} k^2 )</td>
</tr>
<tr>
<td>Part 3</td>
<td>( k )</td>
<td>( m )</td>
</tr>
<tr>
<td>Part 4</td>
<td>( m-1 )</td>
<td>( \log m )</td>
</tr>
</tbody>
</table>

It is clear that our algorithm saves much time in each part. Since Algorithm 6 is very similar to Algorithm 5, the comparison between the sequential and the parallel algorithms is very similar to Table 1. The number \( N_i \) is always smaller than \( n_i \).

VI. CONCLUSIONS

For the basic operations like modular reduction, modular inverse, and multi-multiplication for 3 or 4 integers to be multiplied, we employ the recently proposed fastest algorithms. Notice that the multi-multiplication algorithm is very suitable for the computing of CRT and J-conditions. In this point of view, the proposed algorithms are quite efficient.

The computation of CRT and J-conditions are so complex that some processors are useless in some parts of the proposed algorithms. Due to the constraint of the ring topology we use, two different data can't be sent from one same node at the same time. All the data should be transmitted in the same direction and no conflict is allowed since there is only one path between two adjacent nodes. If the constraint is let go, e.g., two paths between two adjacent nodes, the number of processors required may be reduced to half. However, this will make the parallel algorithm become more complicated; moreover, this kind of topology is harder to design. This is a kind of tradeoff. The best way is to find out an algorithm that is able to solve the CRT and J-conditions problems using only half of the processors we used instead of enhancing the hardware topology.

REFERENCES


