Output-sensitive decoding for redundant residue systems

Majid Khonji majid.khonji@imag.fr
Clément Pernet clement.pernet@imag.fr
Jean-Louis Roch jean-louis.roch@imag.fr
Thomas Roche thomas.roche@imag.fr
Thomas Stalinski thomas.stalinski@imag.fr
Grenoble Univ. INRIA MOAIS, LIG, 51, avenue J. Kuntzmann, F38330 Montbonnot Saint Martin, France

ABSTRACT

We study algorithm based fault tolerance techniques for supporting malicious errors in distributed computations based on Chinese remainder theorem. The description holds for both computations with integers or with polynomials over a field. It unifies the approaches of redundant residue number systems and redundant polynomial systems through the Reed Solomon decoding algorithm proposed by Gao. We propose several variations on the application of the extended Euclid algorithm, where the error correction rate is adaptive. Several improvements are studied, including the use of various criterions for the termination of the Euclidean Algorithm, and an acceleration using the Half-GCD techniques. When there is some redundancy in the input, a gap in the quotient sequence is stated at the step matching the error correction, which enables early termination parallel computations. Experiments are shown to compare these approaches.

Categories and Subject Descriptors
G.4 [Mathematics and Computing]: Mathematical Software—Algorithm Design and Analysis, Reliability and robustness; E.4 [Coding and Information Theory]: Error Control Codes; I.1.2 [Computing Methodologies]: Symbolic and Algebraic Manipulation

General Terms

Keywords
Algorithm Based Fault Tolerance, Redundant Residue Number System, Early termination, Adaptive algorithms, Fast extended Euclidean algorithm

1. INTRODUCTION

In the context of distributed computations, grid computing or more recently cloud computing, the computation has to be secured against failures of several kinds: crash fault, where an expected data is never received, or malicious errors where a data is still being transmitted but is corrupted (e.g. by a malicious code). The model of Byzantine errors [14, 18] illustrates the case where the computation is not always corrupted, and therefore, one might still want to use the results provided by this computing node. These errors can be managed, at least heuristically, using error detection: for instance blacklisting of corrupted machines, or checking post conditions on the results. Yet, errors can be corrected using fault tolerant techniques based on redundancy: either by duplicating the same computation on several machines (e.g. using replication codes); or, to further improve efficiency, by introducing redundancy at the algorithm level in the framework of Algorithm Based Fault Tolerance (ABFT) [10]. For instance, to manage some crash faults, ABFT matrix-vector computation is achieved in [2]: redundancy is added in the matrix-vector product by slightly increasing the dimension of the matrix based one an error correcting code.

Evaluation/interpolation techniques, such as Chinese renumbering algorithm, are very well suited for parallel computations, as they allow to split a computation into independent tasks. For example, the computation of the determinant of an integer matrix, can be parallelized with a residue number system (RNS), where each parallel task is the computation of a determinant modulo a prime number.

When the result to be reconstructed by a RNS is expected to be small, but no bound is known, the early termination approach [4] allows to limit the number of modular computations to the appropriate amount. This adaptive approach lead to a complexity that is output-sensitive. It is of great interest for e.g. computations with sparse or structured matrices.

Residue systems also allow to easily introduce redundancy, by doing additional modular computations, in order to form a redundant residue number system (RRNS) [21, 15]. But they heavily rely on an a priori knowledge of a bound on the output, thus preventing the use of early termination.

In this paper, we propose a unified point of view on redundant residue systems for both polynomial and integer computations, gathering together Gao’s algorithm [5] for the polynomials and Mandelbaum’s algorithm [16] for the integers. We then relax the prerequisites for these algorithms to make them output sensitive. The advantage is twofold: the computation is adaptive in both the output size and the error rate induced by the varying reliability of a global computing environment. We also describe how to incorporate a fast Euclidean algorithm [20] in this framework.
2. REDUNDANT RESIDUE SYSTEMS

2.1 Notations and state of the art

Let \( R \) be an Euclidean ring equipped with an Euclidean function \( \nu : R \to \mathbb{Z}_+ \). We will assume that \( \nu \) is multiplicative and sub-additive, i.e. for all \( a, b \in R \):

\[
\nu(ab) = \nu(a)\nu(b),
\nu(a + b) \leq \nu(a) + \nu(b).
\]

For \( R = \mathbb{Z} \), the usual Euclidean function \( \nu(x) = |x| \) can be used. For \( R = K[X] \), where \( K \) is a field, one can use the function \( \nu(a) = 2^{|\log_2(a)|} \) (see [6, Note 3.1]). For an element \( x \in R \), we define its amplitude as the valuation \( \nu(x) \) and its size as the logarithm of its amplitude \( \log_2 \nu(x) \). Note that the size is a real number.

Let \( p_1, \ldots, p_m \) be \( m \) elements in \( R \) pairwise relatively prime. Define \( \Pi = \prod_{i=1}^{m} p_i \).

The usual construction of redundant residue system (or ideal error correcting codes) proceeds as follows: consider \( \mathcal{M} = \{X \in R \mid \nu(X) < \kappa \} \) with \( 0 \leq \kappa \leq \nu(\Pi) \). The code \( \mathcal{C} \) is a subset of \( \Pi(R/p_1) \times \cdots \times \Pi(R/p_m) \) defined by \( \mathcal{C} = \{(p \mod p_1, \ldots, p \mod p_m) \mid p \in \mathcal{M} \} \). This allows to easily define the notion of symbol of the code (each residue of \( p \mod p_i \) defines a symbol of the code). In the literature \([16, 7, 13]\) the decoding algorithm (algorithm 1) based on extended Euclidean algorithm: it allows to describe the decoding algorithm in an unified manner over the Euclidean rings of polynomials and integers, and without conditions on the size and order of the modulus. After the description of a first bounded error decoding algorithm (algorithm 1), we propose an improvement making it adaptive in the error correction capacity in algorithms 2 and 3.

The following notations will be used throughout the sequel. \( X \) is a codeword and \( Y \) is the erroneous word received. Then \( E = Y - X \) is the error. We define \( I = \{i \in 1 \cdots m \mid X \notin Y \mod P_i \} \) as the set of indices of the erroneous residues. The impact \( \Pi_I \) of the error is defined by \( \Pi_I = \Pi/\Pi_{\mathcal{F}_I} = \prod_{i \notin I} P_i \).

We will define an amplitude-code as a subset \( \mathcal{C} = \{x \in R \mid \nu(x) < \kappa \} \) of the set \( \mathcal{A} = \{x \in R \mid \nu(x) < \nu(\Pi) \} \). In order to form an error correcting code, there remain to define a distance in \( \mathcal{A} \).

**Lemma 2.1.** The function

\[
\Delta : \mathcal{A} \times \mathcal{A} \to \mathbb{R}_+ \quad (x, y) \mapsto \sum_{i \neq j} \log_2 \nu(P_i)
\]

is a metric over \( \mathcal{A} \).

**Proof.** \( \Delta \) is positive, symmetric, satisfies the triangular inequality, and \( \Delta(x, y) = 0 \) if and only if \( x = y \). \( \square \)

The distance \( \Delta \) is a real number; it leads to the definition of an amplitude-code in the ring \( R \) which characteristics \((n, k, d)\) are real numbers. Informally, \( \Delta \) corresponds to the maximal amount of information that differs between the two words.

**Definition 2.2.** Let \( \Pi = \prod_{i=1}^{m} p_i \), where \( p_1, \ldots, p_m \) are pairwise relatively prime. Let \( n = \log_2 \nu(\Pi), k = \log_2 \kappa \). A \((n, k, d)\)-amplitude code is a subset \( \mathcal{C} = \{x \in R \mid \nu(x) < \kappa \} \) of the set \( \mathcal{A} = \{x \in R \mid \nu(x) < \nu(\Pi) \} \) such that the minimal distance between two elements of \( \mathcal{C} \) is \( d \).

**Proposition 2.3.** For any \((n, k, d)\)-amplitude code,

\[
d > n - k - 1
\]

**Proof.** Let \( X, Y \in \mathcal{C} \) and \( X \neq Y \). Let \( E = Y - X \) and \( I \) be the subset of \( \{1, \cdots, m\} \) such that \( X \neq Y \mod P_i \) if and only if \( i \in I \). Let \( \Pi_I = \prod_{i \in I} P_i \) and \( \Pi_{\mathcal{F}_I} = \Pi/\Pi_I \).

\[
\Delta(X, Y) = \log_2 \nu(\Pi_{\mathcal{F}_I})
\]

Now \( E \) is a multiple of \( \Pi_I \), as \( E = 0 \mod P_i \) for any \( i \notin I \). Therefore

\[
\nu(E) \geq \nu(\Pi_I)/\nu(\Pi_{\mathcal{F}_I})
\]

On the other hand \( \nu(\Pi_I) \leq \nu(X) + \nu(Y) < 2\kappa \). Consequently \( \nu(\Pi_I)/\nu(\Pi_{\mathcal{F}_I}) < 2\kappa \) and then \( d > n - k - 1 \). \( \square \)
Corollary 2.4. When the Euclidean function $\nu$ satisfies the following identity: $\nu(a + b) = \max(\nu(a), \nu(b))$ for all $a, b \in R$, then a tighter bound on the minimal distance can be achieved:

$$d > n - k.$$  
Moreover, in the case of the ring $R = K[x]$, where $K$ is a field, choosing $\nu(a) = 2^\deg(a)$ leads to the equivalent of the Singleton bound $d \geq n - \kappa + 1$.

Proof. In that case $\nu(E) = \max(\nu(X), \nu(Y)) < \kappa$. It follows $\nu(\Pi)/\nu(\Pi_F) < \kappa$ and then $d > n - k$. When $R = K[x]$ and $\nu(a) = 2^\deg(a)$, the characteristics $n, k, d$ are integers, therefore $d \geq n - k + 1$. $\square$

In the setting of coding theory, the minimal distance gives a bound on the maximal number of errors that can be corrected: $t = \left\lfloor \frac{d - 1}{2} \right\rfloor$. Here, it corresponds to the maximal amplitude that can be corrected. More precisely, let $\Pi_F = \prod_{i \in I} P_i$ be the impact, and $\nu(\Pi_F)$ its amplitude. Then a code will have amplitude-\tau corrector if it can correct errors of impact $\Pi_F$, such that $\nu(\Pi_F) \leq \tau$.

Hence, we could aim at correcting impact of amplitude $\tau$ such that $\log_2 \tau = \frac{d - 1}{2} = \frac{n - k - 2}{2}$. This approach prevents us from reaching the optimal correction rate of [9] but remains close to it when the valuations $\nu(P_1), \ldots, \nu(P_n)$ remain of the same order of magnitude.

2.2 The bounded impact amplitude algorithm

The Euclidean function and metric defined in the previous section allow to unify both algorithms from [16] for integers and [19, 5] for polynomials. In the following, the Euclidean ring $R$ can be either $\mathbb{Z}$ or $K[X]$, where $K$ is a field. We present the decoding algorithm based on the usual extended Euclidean algorithm. It allows to correct up to the largest impact amplitude with respect to the bound on the minimal distance (property 2.3).

Algorithm 1: Amplitude based decoder over $R$

Data: $\Pi \in R$: the product of the moduli
Data: $Y \in R$: the possibly erroneous message
Data: $\tau \in \mathbb{R}_+$, $\tau < \frac{\nu(\Pi)}{2\tau}$: a bound on the amplitude of the maximal impact of an error to correct
Result: $X \in R$: the corrected message satisfying $\nu(X)4\tau^2 \leq \nu(\Pi)$

begin
  $a_0 = 1, b_0 = 0, r_0 = \Pi$;
  $a_1 = 0, b_1 = 1, r_1 = Y$;
  $i = 1$;
  while ($\nu(r_i) > \nu(\Pi)/2\tau$) do
    Let $r_{i+1} = q_i r_i + r_{i-1}$ be the Euclidean division of $r_{i-1}$ by $r_i$;
    $a_{i+1} = a_{i-1} - q_i a_i$;
    $b_{i+1} = b_{i-1} - q_i b_i$;
    $i = i + 1$;
  return $X = \frac{r_i}{\lambda_i}$

end

Theorem 2.5. Algorithm 1 decodes any corrupted message $Y$ originating from a codeword $X$ affected by an error impact $\Pi_F$ such that $4\nu(X)\nu(\Pi_F)^2 \leq \nu(\Pi)$ and $\nu(\Pi_F) \leq \tau$.

Proof. In the last iteration of the loop, the invariant relation in the extended Euclidean algorithm is written

$$a_{i+1} \Pi - b_{i+1} Y = r_{i+1}$$

Developing the noisy message $Y = E + X$ gives:

$$a_{i+1} \Pi - b_{i+1} E = r_{i+1} + b_{i+1} X.$$  
Now $\gcd(\Pi, E) = \Pi_T$, therefore there is a $Z \in R$ such that $Z \Pi_T = r_{i+1} + b_{i+1} X$.

We now will prove that $Z = 0$ which implies that $X = -\frac{r_{i+1}}{b_{i+1}}$ upon termination of the algorithm.

A usual result [6, Lemma 5.15] in the extended Euclidean algorithm over $R$ states that $\nu(b_{i+1}) \leq \frac{\nu(\Pi)}{2\tau^2}$ (the inequality is strict in $\mathbb{Z}$ and becomes an equality in $K[X]$). The condition of termination of the while loop writes

$$\nu(r_{i+1}) \leq \frac{\nu(\Pi)}{2\tau} < \nu(r_i)$$

Hence, $\nu(b_{i+1}) < 2\tau$, and we have

$$\nu(Z \cdot \Pi_T) \leq \nu(r_{i+1}) + \nu(X)\nu(b_{i+1})$$

$$< \nu(r_{i+1}) + 2\tau \nu(X)$$
$$< \nu(r_{i+1}) + 2\tau \frac{\nu(\Pi)}{4\tau^2}$$
$$< \frac{\nu(\Pi)}{2\tau} + \nu(\Pi)$$
$$< \frac{\nu(\Pi)}{2\tau}$$

Now, by definition $\nu(\Pi_T) \geq \nu(\Pi)/\tau$. Finally the only possible value for $Z$ is 0. $\square$

Corollary 2.6. When the Euclidean function $\nu$ satisfies the identity $\nu(a+b) = \max(\nu(a), \nu(b))$ for all $a, b \in R$, then a tighter bound for the error correction bound can be achieved: $\nu(X)2\tau^2 \leq \nu(\Pi)$. This is the case for the ring $R = K[x]$, where $K$ is a field, and $\nu(a) = 2^\deg(a)$.

In this situation again, the algorithm corrects up to the largest error amplitude with respect to the bound on the minimal distance given in corollary 2.4.

Proof.

$$\nu(Z \Pi_T) \leq \max(\nu(r_{i+1}), \nu(X)\nu(b_{i+1}))$$

Now $\nu(r_{i+1}) \leq \frac{\nu(\Pi)}{2\tau}$ and

$$\nu(X)\nu(b_{i+1}) < 2\tau \frac{\nu(\Pi)}{2\tau^2}$$
$$< \frac{\nu(\Pi)}{\tau}$$

Therefore

$$\nu(Z \Pi_T) < \frac{\nu(\Pi)}{\tau}.$$  
Since $\nu(\Pi_T) \geq \nu(\Pi)/\tau$ we have $Z = 0$. $\square$

The next proposition states the complexity of algorithm 1.

Proposition 2.7. The complexity of algorithm 1 (arithmetic complexity over $K[X]$ or bit complexity over $\mathbb{Z}$) is $O(n \tau)$, where $t \leq \log_2 \tau$ is the size of the error.

Proof. From the analysis of the classical Extended Euclidean algorithm [12, §4.5.3] applied to two integers or polynomials of size $n$ the number of iterations to reach a remainder $r_i$ with size $\lambda$ is less than $2(n - \lambda) = O(t)$. The total complexity is $O(nt)$ using classical arithmetic. $\square$
Fast algorithm using HGCD.

The complexity can be improved based on the use of a fast truncated gcd algorithm. Indeed, the stopping condition \( \nu(r_i) > \nu(\Pi)/2\tau \) is related to the valuation of the remainder \( r_i \). This valuation is derived from the size of \( r_i \) (degree for a polynomial, \( \log_2 \) for an integer) without requiring a computation of the exact value of \( r_i \). Thus, a divide-and-conquer algorithm, such as HGCD [6, §11], [20], can be used to compute the Euclidean sequence of quotients but not the exact remainders, until the modified stopping condition (based on the size) is reached: then only, the corresponding remainder is computed. The complexity of this algorithm becomes \( O(\log t \cdot M(n)) \) where \( M(n) \) is the cost of the multiplication of two elements of size \( n \) in \( R \).

### 2.3 Comparison with previous decoding algorithms

The decoding algorithm 1 allows to correct a bounded amplitude of the error impact in a \((n, k, d)\)-amplitude code defined over \( Z \) or \( K[X] \) equipped with a sub-additive and multiplicative euclidean function. As it has been mentioned in previous sections algorithm 1 can be seen as a generalization of the algorithm of Mandelbaum [16] over the set of integers or a generalisation of the algorithms of Shiozaki [19] and Gao [5] over the set of polynomials.

Furthermore, when \( R = K[X] \) and the \( P_i \)'s all have degree one, then algorithm 1 corresponds exactly to that of Gao [5] presented as an alternative decoder for Reed-Solomon codes. The bounds and error correction capacity are also the same.

Apart from the special case of Reed-Solomon codes, our representation makes a significant difference in the correction capacity. Algorithm 1 is only correcting an amplitude of error impact: this does not mean anything on the number of erroneous residues that can actually be corrected. Therefore, two single errors affecting two different moduli \( P_i \) and \( P_j \) such that \( \nu(P_i) < \nu(P_j) \) don’t have the same weight in the error correction. This fact fits naturally in our setting but must be masked in the standard representation (any moduli should have the same weight in the correction rate of the code). Hence the correction algorithms usually have to be “patched” to avoid the problem and then match the optimal correction capacity: Mandelbaum algorithm uses a perturbation technique and requires the moduli to stay relatively close, equivalently the algorithm of Shiozaki require the moduli to have same degree. On the other hand, Goldreich, Ron and Sudan [7] modify the correction capacity in order to make it dependent to the difference of amplitude of the moduli. Finally Guruswami & Al. [9] use weights on the residues to express this distortion.

### 3. OUTPUT SENSITIVE DECODING AND EARLY TERMINATION

The Chinese remainder algorithm requires a bound on the result to be reconstructed which in turn determines how many modular computations need to be done. In practice the bound might be pessimistic, and a smaller number of modular computations are sufficient for the reconstruction.

We show in sections 3.1 and 3.2 how to take advantage of this unnecessary redundancy to increase the error correction capacity, generalizing algorithm 1 into an adaptive algorithm.

Now this approach still assumes that the total number of modular computations is fixed. Section 3.4 proposes an early termination framework, that allows both adaptive correction rate and output sensitive number of modular computations.

#### 3.1 A first adaptive heuristic

We assume here that no information on the bound \( \kappa \) nor any bound on the maximal error amplitude \( \tau \) is known, but only the values of the product \( \Pi \), and the message \( Y \). The general approach is to define a termination criterion based on the current state of the execution of the Extended Euclidean algorithm.

This first criterion is adapted from a step in the algorithm by Mandelbaum [16]. The idea is to check that \( \beta_{i+1} \) divides \( \Pi \). Indeed, since the error \( E = Y - X \) satisfies: \( \alpha_{i+1} \Pi - \beta_{i+1} E = 0 \), we have \( E = \alpha_{i+1} \frac{\Pi}{\beta_{i+1}} \), which implies that \( \beta_{i+1} \) divides \( \Pi \alpha_{i+1} \). More precisely, lemma 3.1 shows that \( \beta_{i+1} \) actually divides \( \Pi \), yet is a multiple of the impact \( \Pi_F \).

**Lemma 3.1.** Suppose that \( \nu(\Pi) \geq \nu(X)4\nu(\Pi_F)^2 \). Let \( i \) be the iteration where the relation

\[ \nu(r_{i+1}) \leq \frac{\nu(\Pi)}{2\nu(\Pi_F)} < \nu(r_i) \]

holds. Then \( \Pi_F \) divides \( \beta_{i+1} \) and \( \nu(\beta_{i+1}) = \nu(\Pi_F) \).

**Proof.** First, \( \nu(\beta_{i+1}) \leq \frac{\nu(\Pi)}{2\nu(\Pi_F)} < 2\nu(\Pi_F) \). Now, \( \alpha_{i+1} \Pi - \beta_{i+1} E = 0 \) (applying the proof of theorem 2.5 with \( \tau = \Pi \)), hence \( \Pi_F \) divides \( E\beta_{i+1} \). Since \( E \) and \( \Pi_F \) are relatively prime, we deduce that \( \Pi_F \) divides \( \beta_{i+1} \); there exist \( \gamma \in R \) such that \( \beta_{i+1} = \gamma\Pi_F \).

Therefore \( \nu(\Pi_F) \leq \nu(\beta_{i+1}) < 2\nu(\Pi_F) \) which implies that \( \nu(\gamma) = 1 \).  

Lemma 3.1 implies two necessary conditions for \( X \) to be a valid codeword: \( \beta_j \) divides \( \Pi \) and

\[ \nu(X) \leq \frac{\nu(\Pi)}{4\nu(\beta_j)^2} \]

at some iteration \( j \) of the Extended Euclidean algorithm applied to \( \Pi \) and \( Y \).

Algorithm 2 summarizes the adaptive algorithm based on divisibility check.

Since no information is known about the maximal size of the codewords \( \kappa \) or the maximal error correction capacity \( \tau \) the algorithm returns a list of candidates that all satisfy a condition of the type \( \nu(X)4\tau^2 \leq \nu(\Pi) \), for varying values of \( \tau \). In order to discriminate the right solution, one needs to check a postcondition on the result, as will be described in section 3.4.

#### 3.2 Detecting a gap

An alternative termination criterion is to consider the size of the quotients in the Euclidean algorithm. We consider a modified bound on the amplitude of \( \Pi \):

\[ \nu(\Pi) \geq 4\nu(X)\nu(\Pi_F)^22^g \]

where \( g \) is an arbitrarily chosen positive integer that we will refer to as the gap. Introducing this gap, somehow limits the maximal size of error \( \nu(\Pi_F) \) that can be corrected, or equivalently, limits the maximal size of \( X \). Thus, the product \( \Pi_F \) of the correct moduli includes a larger amount of redundancy, characterized by \( \nu(\Pi_F) \geq 4\nu(X)\nu(\Pi_F)^22^g \). This will be used to detect the termination of the decoding: lemma 3.2 states that at the last step \( i \) of the Extended Euclidean algorithm, the size of the quotient \( q_i \) is necessarily larger than the gap \( g \).
Lemma 3.2. Suppose that \( \nu(\Pi) = \nu(X)4\nu(1r_i)2^g \) with \( g \in \mathbb{Z}_+ \). Let \( i \) be the iteration where the relation

\[
\nu(r_i+1) \leq \frac{\nu(\Pi)}{2\nu(1r_i)} < \nu(r_i)
\]

holds. Then \( \nu(q_i+1) \geq 2^g \).

Proof. First, \( \nu(\beta_{i+1}) < \nu(1r_i) < 2\nu(1r_i) \). Now, \( \alpha_{i+1}\Pi - \beta_{i+1}E = 0 \) (applying the proof of theorem 2.5 with \( \tau = \Pi_f \)), hence \( r_{i+1} = \beta_{i+1}X \) and then \( \nu(r_{i+1}) = \nu(\beta_{i+1})\nu(X) \). Consequently \( \nu(\beta_{i+1}) < \nu(1r_i) \).

On the other hand, \( \nu(r_i) > \frac{\nu(r_i)}{2\nu(1r_i)} = 2\nu(X)\nu(1r_i)2^g \). Thus \( \frac{\nu(r_i)}{\nu(r_{i+1})} > 2^g \). Now, \( \nu(q_{i+1}) = \nu(r_{i+2}) + \nu(q_{i+1})\nu(r_{i+1}) \) implies \( \nu(r_i) < r_{i+1}(1 + \nu(q_{i+1})) \). Finally

\[
2^g < \frac{\nu(r_i)}{\nu(r_{i+1})} \leq 1 + \nu(q_{i+1})
\]

which completes the proof.

Algorithm 2: Adaptive decoding by divisibility check

Data: \( \Pi \in R \): the product of the moduli
Data: \( Y \in R \): the possibly erroneous message
Result: \( C \): a list of possible candidates \( X_i \) satisfying:

\[
\nu(X_i)4^2 \leq \nu(\Pi) \text{ if an error of amplitude } \tau \text{ occurred}
\]

begin

\( a_0 = 1, \beta_0 = 0, r_0 = \Pi \)
\( a_1 = 0, \beta_1 = 1, r_1 = Y \)
\( i = 1, C = \{ \} \)
while \( \nu(r_i) > 0 \) do

Let \( r_{i-1} = q_ir_i + r_{i+1} \) be the Euclidean division

of \( r_{i-1} \) by \( r_i \)
\( \alpha_{i+1} = \alpha_{i-1} - q_i\alpha_i \)
\( \beta_{i+1} = \beta_{i-1} - q_i\beta_i \)
if \( \beta_{i+1} \text{ divides } \Pi \) then

\[
X = -\frac{r_{i-1}}{\beta_{i+1}} + \frac{1}{2\nu(1r_i)}
\]

Push \( X \) in \( C \)
end

return \( C \)
end

Algorithm 3: Adaptive algorithm, by detection of a gap

Data: \( \Pi \in R \): the product of the moduli
Data: \( g \in \mathbb{Z}_+ \): the size of the gap
Data: \( Y \in R \): the possibly erroneous message
Result: \( C \): a list of possible candidates \( X_i \) satisfying:

\[
\nu(X_i)4^22^g = \nu(\Pi) \text{ if an error of amplitude } \tau \text{ occurred}
\]

begin

\( a_0 = 1, \beta_0 = 0, r_0 = \Pi \)
\( a_1 = 0, \beta_1 = 1, r_1 = Y \)
\( i = 1, C = \{ \} \)
while \( \nu(r_i) > 0 \) do

Let \( r_{i-1} = q_ir_i + r_{i+1} \) be the Euclidean division

of \( r_{i-1} \) by \( r_i \)
if \( \nu(q_i) \geq 2^g \) then

\[
X = -\frac{r_{i-1}}{q_i\beta_i} + \frac{1}{2\nu(1r_i)}
\]

Push \( X \) in \( C \)
end

\( \alpha_{i+1} = \alpha_{i-1} - q_i\alpha_i \)
\( \beta_{i+1} = \beta_{i-1} - q_i\beta_i \)
\( i = i + 1 \)
return \( C \)
end

Lemma 3.2 states a necessary condition for \( X = -\frac{r_i}{\beta_i} \) to be a valid codeword at the iteration \( j \) of the Extended Euclidean algorithm applied to \( \Pi \) and \( Y \): \( \nu(q_j) \geq 2^g \). This property is used in algorithm 3: only steps corresponding to a quotient with valuation larger than the gap are considered for the recovery of \( X \). The main interest is to decrease the complexity, since the average amplitude for the quotient in the Extended Euclidean algorithm is small: less than 3 for the integers in average [12].

Similarly to algorithm 2, the algorithm based on the gap’s criterion allows to keep the valuations of \( X \) and \( \Pi \) unknown (as far as the relation \( \nu(X)4^2\nu(\Pi)2^g = \nu(\Pi) \) is verified). Even though the presence of a gap seems to decrease the overall capacity of the code compared to algorithm 2, it allows to restrain the number of acceptable candidates: we will see in section 3.3 (table 1) that a relatively small size of the gap (i.e. in our experiments in \( \mathbb{Z} \), a small number of extra redundant bits in comparison to the size of one moduli) leads to a list of only one or two valid candidates and reduces the number of divisibility checks as much.

Again here, the removal of the invalid remaining candidates in the list \( C \) needs to be done, for instance using an external certifier (see section 3.4).

Fast algorithm using HGCD.

Interestingly, algorithm 3 allows to use a fast Extended Euclidean algorithm instead of the standard algorithm. Indeed, when the algorithm HGCD [6, 11], [20], improves the complexity by an order of magnitude, it still constructs the whole list of quotients of the classic Extended Euclidean Algorithm. Algorithm 3, with a termination criterion based on the size of the quotient, can then be easily upgraded in a fast version.

On the contrary, algorithm 2 needs to check every remainder of the Extended Euclidean Algorithm and then won’t allow to use a fast version.

3.3 Experimental comparison

We implemented these algorithms for \( R = \mathbb{Z} \) in \( \mathbb{C} \) using the GMP library [8] for the multi-precision integer arithmetic. In each experiment, \( \Pi \) is the product of the first \( m \) prime numbers of 21 bits, and \( \kappa \) is the product of the first \( \ell < m \) of them.

In figure 2 and 1 we compare the computation time of the static algorithm 1 (Threshold), and the adaptive algorithms 2 (Divisibility) and algorithm 3 (Gap detection), for three parameters of the gap: \( g = 2, 5, 10 \).

First the best computation time is always achieved by algorithm 1, where the parameters are known, since it requires no additional computation for the termination. Now if no information is known on \( \kappa \), the first adaptive algorithm, based on divisibility checks, is slowed down by these expensive tests, performed at each iteration of the Euclidean algorithm. The gap detection of algorithm 3 is much cheaper
Figure 1: Comparison of the variants of the decoder for $n \approx 26\,016$ ($m = 1300$ moduli of 20 bits), $\kappa \approx 6001$ (300 moduli) and $\tau \approx 10007$ (about 500 moduli).

Figure 2: Comparison of the variants of the decoder for $n \approx 200\,917$ ($m = 10000$ moduli of 20 bits), $\kappa \approx 17\,0667$ (8500 moduli) and $\tau \approx 10\,498$ (500 moduli).

(a simple evaluation of the size of the current quotient $q_i$). Now the gap can be set arbitrarily: if it is large, it will reduce the maximal error capacity, for a given $n$ and $X$, and if it is too small, it will reveal several false positives, that need to be discarded by a test of divisibility. For the extremely small value of $g = 2$, the computation time is already significantly reduced, compared to algorithm 2. Then changing it to $g = 5$ bits, still drastically reduce the computation time, as it reduces the number of false positives. For $g = 10$ (half the size of a modulo), the computation time almost matches that of the static algorithm 1.

Now, since no information on the size of the message is known, the gap algorithm has to keep going in the Extended Euclidean algorithm down to the end. Thus if the amplitude of $X$ is relatively large compared to $n$, this will make significant useless amount of work. Figure 1 shows this phenomenon: as the gap $g$ increases, the computation time reduces down to a limit, that roughly corresponds to the time of executing the Extended Euclidean algorithm to the end.

The static algorithm, takes advantage of the a priori knowledge of the threshold $\tau$ to terminate earlier. This motivates the introduction of an external certification algorithm, that will test each candidate produced by the decoder on the fly, against a few independent additional modular residues, and stop the decoder whenever a certification succeeded. This will be described in section 3.4.

<table>
<thead>
<tr>
<th>Error size</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g = 2$</td>
<td>1/446</td>
<td>1/765</td>
<td>1/1118</td>
<td>2/1183</td>
<td>2/4165</td>
<td>1/7907</td>
</tr>
<tr>
<td>$g = 3$</td>
<td>1/244</td>
<td>1/414</td>
<td>1/576</td>
<td>2/1002</td>
<td>2/2164</td>
<td>1/4117</td>
</tr>
<tr>
<td>$g = 5$</td>
<td>1/53</td>
<td>1/97</td>
<td>1/153</td>
<td>2/262</td>
<td>1/575</td>
<td>1/1106</td>
</tr>
<tr>
<td>$g = 10$</td>
<td>1/1</td>
<td>1/1</td>
<td>1/9</td>
<td>1/14</td>
<td>1/26</td>
<td>1/35</td>
</tr>
<tr>
<td>$g = 20$</td>
<td>1/1</td>
<td>1/1</td>
<td>1/1</td>
<td>1/1</td>
<td>1/1</td>
<td>1/1</td>
</tr>
</tbody>
</table>

Table 1: Number of candidates in the gap algorithm: $\frac{c}{d}$ means that $d$ candidates appeared with a gap larger than $g$, and $c$ out of them passed the divisibility check. $n \approx 6001$ (3000 moduli), $\kappa \approx 201$ (100 moduli).

However, the amount of candidates that remain after the divisibility check is very small, and in most cases there is only one of them: the correct result. Table 1 displays the fraction of candidates passing the divisibility check over the total number of candidates passing the gap condition for different values of $g$ and error size.

3.4 A framework for early termination

Early termination

In the previous section, the number $m$ of modular computations was fixed, and only the correction capacity was made output-sensitive. In order to limit the number of modular computations, early termination Chinese remaindering is commonly used [11, 3]: an increasing number of modular residues are computed until the reconstructed value in $R$ stabilizes. In the context of parallel computing, a chunk of modular computations will be done in parallel at each step and if the stabilization condition is not met, then another chunk will be computed. Following [1] the cardinality $u_i$ of each chunk needs to follow an amortized function $f$ in order to guarantee a minimal work overhead. They are defined as follows $u_1 = C$ and $u_{i+1} = f(u_i)$. The function $f$ could be e.g. $f(x) = x/\log x$ or $f(x) = x^{1-\epsilon}$, with $1 \gg \epsilon > 0$.

Certifier

The decoder (algorithm 3) only returns a list of candidates $< X^{(1)}, X^{(2)}, \cdots >$, since the parameters of the code are unknown. Therefore a certifier is needed, in order to withdraw the invalid candidates. This can be done by a simple algorithm, maintaining a list of results $r_i \mod P_i$ of modular computations, independent from the one used in the decoder. The certification consists in testing if $X^{(i)} = r_i \mod P_i$ and return the residue that succeeds every tests. By choosing an appropriate number of $r_i$‘s, any probability of success can be guaranteed (refer to [11] for a detailed analysis).

Note that this certifier naturally plays the role of checking the stabilization, as needed for the early termination.
Chinese remaindering lifter

The residues computed on each distributed node must be lifted using the Chinese remainder theorem to form the message $Y$. This operation could either be done on a safe resource running the Decoder, but this can be a significant work load. Another strategy is to have every worker do the reconstruction, and then select the appropriate one by a majority vote, thus preventing a possible corruption by a corrupted node.

A framework for global computing

We now describe a framework for an output-sensitive distributed computation using the components described previously. Following [17], the computing environment is partitioned in two parts: $U$ and $R$. $R$ is a closed set of reliable interconnected computing resources; $R$ is assumed reliable and trustfully, thus it provides a limited computation power $\pi_R$. On the other hand, $U$ includes a large number of resources that operate in an unbounded environment, such as a peer to peer computation environment, and provides a very large computation power $\pi_U$. Such a large scale parallel system is well suited to perform independent computations, such as modular computations. However, since $U$ is open, computations performed on $U$ cannot be considered as fully reliable which motivates the introduction of a fault tolerant reconstruction system.

Figure 3 summarizes the framework of such a system. A program called Master forks the parallel computations following the amortized function $f$. Then the value $R$ is reconstructed by the Lifter and the Decoder (algorithm 3) lists all codewords $X$ that are candidates. The Certifier then discards the invalid candidates. If no candidate is left, then Master forks the next chunk of modular computations. Otherwise, the probabilistically certified result is returned.

5. REFERENCES


