Fast $k$-ary Reduction
and Integer GCD Algorithms

M.S. Sedjelmaci        C. Lavault

LIPN CNRS UPRES-A 7030,
Université Paris-Nord 93430 Villetaneuse, France.
E-mail: {sms,lavault}@lipn.univ-paris13.fr

Abstract

The paper presents a new fast $k$-ary reduction for integer GCD. It enjoys powerful properties and improves on the running time of the quite similar integer GCD algorithm of Kannan et al. Our $k$-ary reduction also improves on Sorenson’s $k$-ary reduction [14] and thus favorably matches Weber’s algorithm [15]. More generally, the fast $k$-ary reduction also provides a basic tool for almost all the best existing integer GCD algorithms.

1 Introduction

Given two integers $u$ and $v$, the greatest common divisor (GCD) of $u$ and $v$, \( \gcd(u, v) \), is the largest integer that divides both $u$ and $v$.

1.1 Related Works and Results

The advent of practical parallel computers has caused the re-examination of many existing algorithms with the hope of discovering a parallel implementation. One of the oldest and best known algorithm for finding the GCD of two integers is Euclid’s algorithm, which uses the GCD preserving transformations $u \mod v$.

Although there have been results in the computation of the GCD of polynomials, the integer case still appeared to be inherently serial. Indeed, in 1983 Brent and Kung [2] achieved a running time of $O(n)$ with $n$ processors arranged in a systolic array, where $n$ is the number of bits required to represent the larger of the two input numbers. Though it is an improvement on the best known serial integer GCD algorithm $O(n \log n)^2 \log \log n)$ of Schönhage [10], their method still requires $n$ iterations; the parallelism only reduces the bit operations per iteration.

In 1987, Kannan, Miller and Rudolph (KMR) [7] gave the first sublinear time parallel integer GCD algorithm on a common CRCW PRAM model. Their time bound is $O(n \log \log n / \log n)$ assuming there are $n^2 (\log n)^2$ processors working in
parallel. (Throughout, \( \lg(n) \) denotes \( \log_2(n) \), the logarithm to the base 2.) Since 1990, Chor et Goldreich [3] currently have the fastest parallel GCD algorithm; it is based on the systolic array GCD algorithm of Brent and Kung. The time complexity of their algorithm achieves \( O(n/\log n) \) using only \( n^{1+\epsilon} \) processors on a CRCW PRAM. By varying the main parameter to the algorithm, they also obtain a polylog time, subexponential processor algorithm. More recently (1994), Sorenson’s right- and left-shift \( k \)-ary algorithms [14] take \( O(n/\log n) \) time using at most \( n^{1+\epsilon} \) processors on a CRCW PRAM, matching Chor and Goldreich’s. Although the \( k \)-ary algorithms seem more involved than, say, the Euclidean and binary algorithms, a straightforward parallelization is sufficient to rival the best previous parallel integer GCD algorithms. Actually, the \( k \)-ary algorithms are simpler than Chor and Goldreich’s algorithm and than the usual sublinear time parallel algorithms. The latter usually “compress” multiple iterations (a ) of simpler algorithms into one iteration, or “phase” [7] (“Packing method” [3]).

1.2 Reduction Techniques

Given two integers \( u \) and \( v \), most serial integer GCD algorithms use one or several transformations which reduce the size of current pairs \((u, v)\), till a pair \((u, 0)\) is eventually reached. The last value \( u = \gcd(u, v) \) is then the result we want to find.

Throughout, we restrict ourselves to the set \( \mathbb{N} \) of non-negative integers. Let \( u \) and \( v \) be two such (non-negative) integers and a function \( \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N} \), such that \((u, v) \mapsto (v, R(u, v))\), where \( R \) is a GCD “reducing transformation” defined within a domain \( D \subseteq \mathbb{N}^* \times \mathbb{N}^* \).

The simplest and most popular transformation for integer GCD algorithms is the linear combination of \( u \) and \( v \); i.e. \( R(u, v) = au + bv \), where \( a \) and \( b \) are assumed to be rational numbers.

Sorenson’s \( k \)-ary GCD algorithms and the like [5, 11, 14, 15] are practical and efficient; they use the \( k \)-ary reduction technique. Given two integers \( u > v > 0 \) relatively prime to \( k \) (i.e., \((u, k)\) and \((v, k)\) are coprime), pairs of integers \((a, b)\) can be found that satisfy

\[
a u + bv \equiv 0 \pmod{k}, \quad \text{with} \quad 0 < |a|, |b| < \sqrt{k}.
\]

The \( k \)-ary reduction performed by the transformation \( R(u, v) = |au + bv|/k \) ensures the following inequality

\[
R(u, v) = |au + bv|/k < 2u/\sqrt{k};
\]

the size of \( u \) is also reduced by roughly \( \frac{1}{2} \lg(k) \) bits. Such algorithms run in \( O(n/\log k) \) iterations. (See [11].)

The above reduction is useful whenever the bit-size difference between \( u \) and \( v \) is small enough. Unfortunately, since the inequality \( R(u, v) < v \) does not surely
hold, the reduction proves inappropriate in the case when their difference is too large. In order to cut off this drawback, Weber [15] and Jebelean [5] choose another preserving transformation called the “dmod reduction” (“digit modulus”) which is very efficient for large integers because it costs much less than the usual binary transformation “modulo”. Thus, according to the size of \( u \) and \( v \), Weber’s algorithm works by simultaneously combining Sorenson’s \( k \)-ary reduction with the dmod reduction [15].

In the new reduction presented in the paper the size of \( v \) is reduced by \( \lg (k) \) bits. In terms of reduced bits, such a reduction is equivalent to two \( k \)-ary and one dmod reductions performed simultaneously. This reduction we call the fast \( k \)-ary reduction.

In Section 2, definitions and basic properties of the fast \( k \)-ary reduction are given, as well as the main results of the paper: Theorems 1 and 2. Section 3 is devoted to the comparison with fundamental \( k \)-ary reductions. A parallel integer GCD routine is designed and analyzed in Section 4, which improves on KMR’s algorithm. Finally, concluding remarks are given in Section 5.

2 The Fast \( k \)-ary Reduction

Throughout, the following notation is used. Given a non-negative integer \( n \in \mathbb{N} \), \( \ell_{\beta}(n) \) represents the number of its significant \( \beta \)-digits, not counting leading zeros, in any base \( \beta \geq 2 \):

\[
\ell_{\beta}(n) = \begin{cases} 
\lfloor \log_{\beta}(n) \rfloor + 1 & \text{if } n \geq 1, \\
1 & \text{if } n = 0.
\end{cases}
\]

Given two integers \( u > v > 0 \), we let \( \rho = \rho(u,v) = \ell_{\beta}(u) - \ell_{\beta}(v) + 1 \). Thus, for all positive integers \( u > v > 0 \), \( \beta^{\rho-1} \leq u < \beta^{\rho} \) and \( \beta^{\rho-2} < u/v < \beta^{\rho} \), where \( s = \ell_{\beta}(u) \).

2.1 Definition and Basic Properties

Definition 1 Let \( u > v > 0 \) be positive integers. We call a reduction any transformation \( R: \mathbb{N}^* \times \mathbb{N}^* \rightarrow \mathbb{N}^* \) satisfying the following two properties:

\[
(P_1) \quad 0 \leq R(u,v) < v.
\]

\[
(P_2) \quad \gcd(v, R(u,v)) = \lambda \gcd(u,v), \text{ with } \lambda > 0.
\]

With \((P_1)\) and \((P_2)\), we are guaranteed that algorithms terminate and return the correct value \( \gcd(u,v) \), up to a constant factor \( \lambda \) which can easily be removed afterwards.
Definition 2 Let $a$ and $b$ be two rational numbers such that $ab \neq 0$. We call a linear reduction any reduction of the form $R(u, v) \overset{def}{=} |au + bv|$.

Example. The following integer GCD algorithms use linear reductions.

1. Euclid’s algorithm: $R(u, v) = u \mod v = u - qv$, with $q = \lceil u/v \rceil$.
2. binary algorithm: $R(u, v) = (u-v)/2$, where $u > v$ are positive odd integers.
3. dmod$\beta$ algorithm: $R(u, v) = |u - xv|/\beta^p$, where $x, \beta$ and $\rho$ are positive integers.
4. Sorenson’s algorithm: $R(u, v) = |au + bv|/k$, where $k > 2^{2\rho+2}$ ($\rho$ positive integer).

The basic properties of linear reductions are first characterized with integer coefficients.

In the following, we consider two integers $u > v > 0$.

Proposition 1 Let $a$ and $b$ be integers. Any linear reduction $R$ writes $R(u, v) = |au - bv|$, where $a, b > 0$.

Proof: Obvious; by contradiction from $(P_1)$ in Definition 1.

Integer linear reductions are closely connected to the extended Euclidean algorithm (EEA). Lemma 1 emphasizes the fact that $\text{gcd}(u, v)$ expresses as linear reductions of $u$ and $v$ [1, 8].

Lemma 1 Let $a, b > 0$ be integers. For any linear reduction of the form $R(u, v) = |au - bv| < v$,

1. $R(u, v) = |au - bv|$ equals either $(au) \mod v$, or $v - (au) \mod v$ (according to whether $au - bv \geq 0$ or not). Therefore, $b = \lceil au/v \rceil$ or $b = \lfloor au/v \rfloor$.

2. $\text{gcd}(v, |au - bv|) = \lambda \text{gcd}(u, v)$, where $\lambda > 0$ divides $a$.

Proof: Since $|au - bv| < v$, $(au/v) - 1 < b < (au/v) + 1$. When $b = \lceil au/v \rceil$, $au - bv = (au) \mod v$, and when $b = \lfloor au/v \rfloor$, $bv - au = v - (au) \mod v$. Besides, $\text{gcd}(v, |au - bv|) = \text{gcd}(v, au) = \lambda \text{gcd}(u, v)$, where $\lambda \mid a (\lambda > 0)$.

Remarks 1.

- By Lemma 1, $(P_1) \implies (P_2)$ and any integer linear reduction is thus uniquely defined with property $(P_1)$.
- In the case when $v \mid au$, $R(u, v) = (au) \mod v = 0$ and $b = au/v \in \mathbb{N}^*$. 

4
Proposition 2 For all integers $u > v > 0$ and $p$ such that $1 \leq p < v$,
\[
d = \gcd(u, v) = \min((pu) \mod v, v - (pu) \mod v),
\]
where $0 < (pu) \mod v < v$.

Proof: The EEA computes the integer coefficients $p$ and $q$ along with
the GCD: $p$ and $q$ are such that $d = \gcd(u, v) = \min(pu + qv > 0)$ (see for example $[1, 8]$). From the EEA we are also guaranteed that there exists a pair
of positive integers $(p, q)$, such that $|pu - qv| = d \leq v$. When $d < v$, the result
follows from Lemma 1; and the case when $d = v$ is obvious: $(pu) \mod v = 0$ and
$v - (pu) \mod v = v$. \hfill \qed

2.2 Main Results

Theorem 1 For all non-negative integers $u > v > k > 0$ such that
$\gcd(v, k) = 1$, there exists a pair $(p, q)$ such that $1 \leq p \leq k - 1$ and $q = \lfloor pu/v \rfloor$
or $q = \lfloor pu/v \rfloor$, which satisfies
(a) $pu - qv \equiv 0 \pmod{k}$.
(b) $|pu - qv| < v$.

Proof:
(a) Consider the two sequences $(r_i)$ and $(d_i)$,
\[
\begin{align*}
  r_i &\overset{\text{def}}{=} (iu) \mod v & \text{and} & & r_k = v; \\
  d_i &\overset{\text{def}}{=} r_i \mod k, & \text{for} & & i = 0, 1, \ldots, k.
\end{align*}
\]
For each $i = 0, 1, \ldots, k$, sequence $(d_i)$ takes only $k$ possible distinct values,
whereas there is a total of $k + 1$ $d_i$s. Therefore, there exist two distinct values $i$
and $j$ not both zero such that $0 \leq j < i \leq k$ and $d_i \neq d_j$.

Set $\delta = r_i - r_j$. Then, $\delta \equiv 0 \pmod{k}$ and (a) holds with $1 \leq p \leq k - 1$ and
$q = \lfloor pu/v \rfloor$ or $q = \lceil pu/v \rceil$, according to whether $\delta \geq 0$ or $\delta < 0$, respectively.

(b) One of the two cases may arise:

- If $i < k$, then $1 \leq i \leq k - 1$. We have $\delta \equiv r_i - j \pmod{v}$. Thus, when
  $\delta \geq 0$, $\delta = (r_i - j) \mod v$ and when $\delta < 0, -\delta = (v - r_i - j) \mod v$. Therefore,
  $|\delta| < v$ and, by Lemma 1, (b) holds with $p = i - j$ and $q = \lfloor pu/v \rfloor$ or
  $q = \lceil pu/v \rceil$ according to whether $\delta \geq 0$ or $\delta < 0$, respectively.

- If $i = k$, then $j \geq 1$; otherwise, $\delta = v \equiv 0 \pmod{k}$, by contradiction with
  the assumption that $\gcd(v, k) = 1$. Thus, $0 < \delta = v - ju \mod v < v$. By
  Lemma 1, choosing $p = j$ and $q = \lceil ju/v \rceil$ yields the result. \hfill \qed

5
Definition 3 Let \((p, q)\) be a pair defined as in Theorem 1. For all integers \(u > v > k > 0\) such that \(\gcd(v, k) = 1\), the fast \(k\)-ary transformation is defined by \(R(u, v) \triangleq \left| pu - qv \right|/k\).

Notice that \(R(u, v) = \left| pu - qv \right|/k < v/k < \frac{u}{k^{\beta^2}}\).

Corollary 1 For all integers \(u > v > k > 0\) such that \(\gcd(v, k) = 1\), there exists a parameter \(p\) such that \(1 \leq p \leq k - 1\) which satisfies
\[
pu \mod v \equiv 0 \pmod{k} \quad \text{or} \quad pu \mod v \equiv v \pmod{k}.
\]

Proof: Immediate by Theorem 1 and the equalities
\[
(\forall i, j \neq k) \quad |r_i - r_j| = \begin{cases} 
  r_{i-j} & \text{if } (r_i - r_j)/(i - j) > 0 \text{ and } i, j < k, \\
  v - r_{i-j} & \text{if } (r_i - r_j)/(i - j) < 0 \text{ and } i, j < k, \\
  v - r_j & \text{if } i = k \text{ and } j > 0.
\end{cases}
\]

Theorem 2 For all integers \(u > v > k > 0\) such that \(\gcd(v, k) = 1\),

(i) The fast \(k\)-ary transformation is a reduction (in the sense of Definition 1).

(ii) \(R(u, v) = 0 \iff \gcd(u, v) = \frac{v}{p} \gcd(p, q) > v/k\).

Proof:

(i) \(R(u, v) < v/k < v\). Now, since \(\gcd(v, k) = 1\),
\[
\gcd(v, R(u, v)) = \gcd(v, kR(u, v)) \text{ and hence},
\]
\[
\gcd(v, R(u, v)) = \gcd(v, |pu - qv|) = \gcd(v, pu) = \lambda \gcd(u, v), \quad \text{with } 1 \leq \lambda | p.
\]

(ii) \((\implies)\) If \(R(u, v) = 0\), then \(pu = qv\). Now, let \(d' = \gcd(p, q)\) and \(d = \gcd(u, v)\). Then we have \(p = d' p_1, q = d' q_1, u = d u_1\) and \(v = d v_1\) with \(\gcd(p_1, q_1) = \gcd(u_1, v_1) = 1\). Then, \(p_1 u_1 = q_1 v_1\) and thus, \(v_1 = p_1\) and \(u_1 = q_1\). By substituting the values,
\[
d = v/v_1 = \frac{v}{p/d'} \geq v/p > v/k,
\]
which yields the sufficient condition.

\((\impliedby)\) If \(d > v/k\), then \(d \mid R(u, v)\). Let then \(R(u, v) = td\), with \(t \geq 0\). If \(t \geq 1\), \(v/k < d \leq R(u, v) < v/k\), which contradicts the assumption. Hence, \(t = 0\) and so is \(R(u, v)\).

Property (ii) checks a situation when the reduction is too big. The condition \(R(u, v) = 0\) is a stopping test for the GCD algorithm since \(\gcd(p, q) \leq p < k\).

The division \(\frac{v}{p/d'}\) is then easily performed because it is exact (it is known in advance that the remainder is zero [6, 12]),
3 Comparison to other Reductions

3.1 Sorenson’s Reduction

Lemma 2 Sorenson’s k-ary transformation is a reduction if \( k > 2^{2^\varphi+2} \).

Proof: It is easily seen that Sorenson’s transformation satisfies \( R(u, v) < 2u/\sqrt{k} < 2^{\varphi+1}v/\sqrt{k} \). A sufficient condition for \( R(u, v) \) to be a reduction is then \( 2^{\varphi+1}v/\sqrt{k} < v \), that is \( k > 2^{2^\varphi+1} \). \( \square \)

Note that Weber [15] and Jebelean [5] use Sorenson’s reduction only when \( k > 2^{4^\varphi-2} \).

3.2 The Reduction of Kannan, Miller and Rudolph

The reduction of Kannan et al. (KMR) [7] is quite similar to ours. A constant factor is however omitted in [7] (though it is present in their previous paper in 1984). This makes the proof of their Lemma 2 false. In Lemma 3, we give a counter-example to the proof and use it to provide a correct statement of our resulting Lemma 4, which bounds the constant factor.

The proof of KMR’s Lemma 2 [7] is based on the following (incorrect) claim.

Claim 1 Given \( N+1 \) non-negative integers such that \( 0 \leq x_i \leq b \) \( (0 \leq i \leq N) \), then there exists at least one pair \( (x_i, x_j) \) \( (x_i \neq x_j) \) with their first \( m \) leading bits equal \( (m = \lfloor \log N \rfloor) \), which satisfies \( |x_i - x_j| \leq b/N \).

The above inequality is false (up to a real constant factor \( C \)); the following Lemma 3 provides a counter-example to Claim 1.

Lemma 3 Let \( N = 2^m \) \( (m \geq 3) \) and \( b = 2^{s-1} \) \( (s \geq 5) \) be integers. There exists a sequence \( (x_i) \) of \( N+1 \) non-negative integers such that \( 0 \leq x_i \leq b \) \( (0 \leq i \leq N) \), which satisfies \( x_{2i+1} - x_{2i} > b/N \).

Proof: Define the sequence \( (x_i) \) of \( N+1 \) distinct positive integers as follows:

\[
x_i \overset{def}{=} [i/2]2^{s-m} + \frac{1}{2}(1 + (-1)^{i+1})(2^{s-m} - 1), \quad \text{for } i = 0, 1, \ldots, N-1,
x_N = b = 2^{s-1}.
\]

For all \( i \) such that \( 2i + 1 \leq N - 1 \), \( x_{2i+1} \) and \( x_{2i} \) have their first \( m \) leading bits equal. However, since \( s - m > 1 \), for all \( i \) \( (0 \leq i \leq N) \),

\[
x_{2i+1} - x_{2i} = 2^{s-m} - 1, \quad \text{whereas} \quad b/N = 2^{s-1}/2^m = 2^{s-m-1}.
\]

Therefore, \( x_{2i+1} - x_{2i} > b/N \) and the claim is false. \( \square \)
Remarks 2.

- There are several other counter-examples to Claim 1. Pairs in the sequence \((x_i)\) other than those mentioned in Lemma 3 (or not making up counter-examples) fail to enjoy the requisite property relating their first \(m\) leading bits.

- Choosing the value \(\Delta_i = x_{2i+1} - x_{2i}\) defined in Lemma 3 yields the largest possible difference \(x_i - x_j\) for any pair \((x_i, x_j)\), \(0 \leq i < j \leq N\), with their first \(m\) leading bits equal:

\[
0 \leq b/N = 2^{s-m-1} < \Delta_i = 2^{s-m} - 1 < 2^{s-m} \leq 2 \cdot \frac{2^{n-1}}{2^m} = 2b/N.
\]

Lemma 4 If \(a\), \(b\) and \(n\) are positive integers and \(a \leq bn\), then there exist integers \(p\) and \(q\) not both zero, such that \(|p| \leq nb/a\) and \(|q| \leq n + 1\), which satisfy

\[
0 \leq pa - qb \leq C a/n, \quad \text{where} \quad 1 < C < 4 \quad (C \in \mathbb{R}).
\]

Proof: Let \(N = \lfloor nb/a \rfloor + 1\), \(s = \ell_2(b)\) and \(m = \lfloor \log N \rfloor\). Consider the sequence \((r_p)\) of remainders \(r_p = pa - qb = (pa) \mod b\), \(p = 0, 1, \ldots, N - 1\). There exist \(N\) distinct such pairs \((p, q)\). Adding the pair \((0, -1)\) such that \(0 < pa - qb = b\), we obtain \(N+1\) pairs \((p, q)\) satisfying \(0 \leq p \leq N - 1\), \(-1 \leq q \leq n\) and \(0 \leq pa - qb \leq b\).

The first leading bits of the \(N + 1\) distinct \(r_p = pa - qb\) are compared here. By the pigeon-hole principle, there exist two values in the sequence \((r_p)\) having their first \(m\) leading bits equal. In other words, there are two pairs \((p_1, q_1)\) and \((p_2, q_2)\) such that

\[
0 \leq (p_1 a - q_1 b) - (p_2 a - q_2 b) \leq 2^{s-m} - 1 < 4 \cdot \frac{2^{s-1}}{2^{m+1}} = \frac{4b}{N} = \frac{4b}{\lfloor nb/a \rfloor + 1} \leq 4a/n,
\]

where \(s = \ell_2(b) = \lfloor \log(b) \rfloor + 1\).

If we let \(p = p_1 - p_2\) and \(q = q_1 - q_2\), the combination \(pa - qb\) satisfies \(|p| \leq nb/a\) and \(|q| \leq n + 1\), and

\[
0 \leq pa - qb \leq C a/n.
\]

Now, from Lemma 3, we also know that the real constant \(C > 1\).

Hence, \(1 < C < 4 \quad (C \in \mathbb{R})\) and the combination \(pa - qb\) is the desired result. \(\square\)

In addition to the above bounds on \(C\), the upper bound on \(q\) is also improved by Lemma 4. Indeed, Lemma 4 assumes that \(|q| \leq n + 1\), whereas KMR’s lemma only assumes that \(|q| \leq 2n\).

Note however that the bounds \(1 < C < 4\) are neither the greatest lower bound, nor the least upper bound: the g.l.b. is very likely to be 2, and the l.u.b. is close to 4.
Remark. In [4, Theorem 36, page 30], Hardy and Wright use the Farey dissection to find a better bound on the approximation of real numbers by rational numbers.

If $\xi$ is any real number, and $n$ a positive integer, then there exists an irreducible fraction $p/q$ such that

$$0 < q \leq n, \quad \left| \xi - \frac{p}{q} \right| \leq \frac{1}{q(n+1)}.$$ 

This theorem actually provides a number theoretic non-constructive proof of KMR's Lemma 2. However, assumptions are more general: in [4], $\xi$ is any real number, while KMR's result and our Lemma 4 both assume $\xi = u/v$ to be rational with the first $m$ leading bits of $u$ and $v$ equal. This of course explains why such combinatorial proof cannot meet the result in [4], but only up to a constant factor. In contrast, these are constructive combinatorial proof, and thus really suited to the design of KMR's integer GCD algorithm.

3.3 Comparison between Fundamental Reductions

For all integers $u > v > k > 0$ such that $\gcd(v,k) = 1$, comparing Sorenson's and KMR's reductions and algorithms with the fast $k$-ary (denoted FR) results in several fundamental comments.

- Substituting $a$ for $u$, $v$ for $b$ and $k = [nv/u]$ for $n$ in Definition 3 yields a similar result to Lemma 4, up to a constant factor $1 < C < 4$.

- In Section 4, the FR GCD algorithm needs to perform only one comparison per processor. By Corollary 1, the last $m$ bits of $iu \mod v$ are compared with 0 when $iu \mod v$ is even, while the last $m$ bits are compared with $v \mod 2^m$ when $iu \mod v$ is odd. By contrast, $O(m)$ comparisons are necessary in KMR's algorithm. (The above improvement is not valid in KMR's reduction.)

- By Theorem 1, an explicit expression of the quotient is established. According to the parity, we perform only one comparison per processor, whereas $O(k)$ comparisons needed in KMR's approach.

- On the same lines, performing computations modulo $k$ is much easier, especially when $k = 2^m$. Moreover, taking $u$ and $v$ odd, we are always guaranteed that property $\gcd(v,k) = 1$ is maintained all along the algorithm.

- KMR's reduction fits to algorithms running in MST ("Most Significant digit First"), whereas our reduction is more suited to LSF ("Least Significant digit First") and systolic applications.

On the same general assumptions and using the same notation, Table 1 is summing up the definitions and properties of various reductions.
Table 1: Basic reductions.

<table>
<thead>
<tr>
<th>Reduction Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorenson’s $k$-ary reduction:</td>
<td>$R(u, v) = \left\lfloor au + bv \right\rfloor / k &lt; 2u/\sqrt{k}$</td>
</tr>
<tr>
<td>dmod$_3$ reduction:</td>
<td>$R(u, v) = \left\lfloor u - xv \right\rfloor / \beta^p &lt; (\beta^p - 1/\beta^p)v$</td>
</tr>
<tr>
<td>KMR’s reduction:</td>
<td>$R(u, v) = \left\lfloor pu - qv \right\rfloor / k &lt; Cv/k, \ 1 &lt; C &lt; 4$</td>
</tr>
<tr>
<td>FR:</td>
<td>$R(u, v) = \left\lfloor pu - qv \right\rfloor / k &lt; v/k$</td>
</tr>
</tbody>
</table>

4 A Fast $k$-ary GCD Algorithm

Given integers $u > v > k > 0$ such that $\gcd(v, k) = 1$, we assume that when the algorithm starts, $u$ is $n$ bits large. For the sake of simplicity, it is usually easier to let $k = 2^m$, where $m$ is multiple of a memory word $\omega = 16, 32$ or $64$ bits (for shifting, computing “modulo”, maintaining $\gcd(v, k) = 1$, etc.). However, we let $m = \lfloor \log n \rfloor$ for $FR$; this value yields $O(n / \log n)$ iterations at most. As to the stop test in the routine, we use $u < k$ (instead of $v = 0$), because $FR$ is undefined when $v < k$ and it is also more general. We find it easier to take $m$ as a “threshold” (the borderline choice between $FR$ and the dmod reductions); but likewise, we might choose a varying threshold, depending upon $v$ and experimental data [5, 15].

4.1 High Level Description of a GCD Algorithm

**Step 1:** Let $d = \gcd(u, v)$. Find $d_1$, such that $d_1$ equals the product of all common divisors to $u$ and $v$ which are less than $k$. (In the algorithm, $d := d_1 d$.)

**Step 2:** Perform reductions until $v < k$: if $\rho \leq m$, then perform $FR$s, else perform the bmod reduction.

- Compute $d = \gcd(u, v)$, where $(u, v)$ is the last pair obtained from Euclid’s algorithm (with $v < k$).

**Step 3:** Remove all divisors $< k$ from $d$.

**Step 4:** Perform the product $d \times d_1$.

Step 1, 3 and 4 are similar to the phases in KMR’s algorithm. Step 2 is designed below. (Refer to [15] for the bmod.)

```
repeat
  $R := \left\lfloor pu - qv \right\rfloor / k$ ; /* in parallel by using $FR$ */
  $R := R/2^t$ ; $(u, v) := (v, R)$  /* $t$ is such that $R$ is made odd */
until $v < k$
  if $v = 0$ then $d' := \gcd(p, q)$ ; $d := v/d'$
  else $d := \gcd(u, v)$  /* perform Euclid’s algorithm */
endif
return $d$
```
Remark. It may be the case that computing with Theorem 1 yields 
\( R(u, v) > k \), though successive halvings yield \( 2^{-t} R(u, v) < k \). Thanks to 
the choice of \( k (k = 2^m) \) and the instruction \( R := R/2^t \), our routine reduces the 
size of \( v \) faster than KMR’s algorithm and is also much alike Brent and Kung’s 
binary algorithms.

4.2 Correctess Proof of the Routine

For all integers \( u > v > k > 0 \) such that \( \gcd(v, k) = 1 \) and \( 1 \leq d \leq v \), step 2 
does terminate in at most \( I \) iterations, where \( I \) is a positive integer defined as 
\( I \overset{def}{=} \lceil \log_k(v/d) \rceil \): \( I \) is such that \( v/k^{l+1} < d < v/k^l \).

The sequence of the current values of \( u \) and \( v \) is \( v_0 = u, \ v_1 = v \) and 
\( v_{i+2} = 2^{-t} R(v_i, v_{i+1}) \) with \( t > 0 \) integer, for \( i = 0, 1, \ldots, I' - 1 \), where \( I' \) 
(possibly +\( \infty \)) denotes the number of iterations in step 2. If \( I \geq I' \) the result is 
obvious. Now suppose \( I < I' \), then \( v_i \) and \( v_{i+1} \) exist and \( R(v_i, v_{i+1}) < k \) (i.e., 
the routine terminates). This is derived by contradiction.

Suppose \( R(v_i, v_{i+1}) > k \), then \( v_{i+2} \) exists and 
\[
0 < v_{i+2} \leq R(v_i, v_{i+1}) < v_{i+1}/k < \cdots < v/k^{l+1}.
\]

Now, by Definition 1 (Property \( P_2 \)), \( d \leq \gcd(v_i, v_{i+1}) \leq v_{i+2} < v/k^{l+1} \), which 
contradicts the definition of \( I \). Therefore, \( R(v_i, v_{i+1}) < k \) \( (v_{i+2} \) does not exist \) 
and the loop of the routine (step 2) terminates in at most \( I = \lceil \log_k(v/d) \rceil \) 
itations. \( \square \)

4.3 Computation of \( FR \)

The computation of \( q_k = [iu/v] \) and \( r_i = [iu - q_i v] \) is performed by finding the 
first leading bits of each value. For each processor \( i \) \( (1 \leq i \leq k - 1) \), \( q_k = [iu/v] \) 
is found as follows:

With the first \( 2\ell \) bits of \( iu \), denoted \( (iu)_\ell \), and the first \( \ell \) bits of \( v \), denoted \( v_\ell \) 
(where \( \ell = \ell_2(iu) - \ell_2(v) \)), the quotient \( q_i' = [(iu)_\ell/v_\ell] \) is computed. The latter 
value \( q_i' \) approximatively equals \( q_i \). More precisely, it is shown in [7] that \( q_i - q_i' = \delta_i \) 
with \( |\delta_i| \leq 3 \).

In order to obtain the exact quotient \( q_i \), more information is needed. The 
quotient has to be such that \( iu - q_i v \equiv 0 \pmod{2^m} \) or \( iu - q_i v \equiv v \pmod{2^m} \). 
More precisely, \( q_k \equiv iu/v \pmod{2^m} \) or \( q_k + 1 \equiv iu/v \pmod{2^m} \). When this 
latter relation holds, it causes the computation of \( q_k \) and \( r_i \) to proceed. Otherwise, 
the calculation is stopped. Note that taking \( \rho \leq m \) yields \( \ell \leq m + \rho - 1 \leq 2m - 1 \).
4.4 Complexity of the Algorithm

Finding FR is performed by using \( k - 1 \) processors in parallel, where each one processor computes the expression \((pu) \mod v = pu - qv\), for \( p = 1, \ldots, k - 1 \), as described in Subsection 4.3. For \( iu \), the first \( 4m - 2 \) and the last \( m \) bits are only needed. For \( v \), the first \( 2m - 1 \) and the last \( m \) bits are only needed.

Therefore, the number of iterations is \( O(n/\log n) \), and the time complexity of each iteration is at most \( O(\log m) = O(\log \log n) \) time, using \( O(m^2) = O(\log^2(n)) \) processors. Precomputed table lookup may improve the time complexity down to \( O(1) \) with \( k^2 = O(\log^2(n)) \) processors.

5 Conclusion

The \( k \)-ary reduction presented in this paper improves on the reductions so far defined. FR may be used as a basic transformation for the best existing GCD algorithms, such as in [14, 15], for example. It also opens various prospects for research and applications.

1. Run Weber’s algorithm [15] by changing Sorenson’s \( k \)-ary for FR, and using the \( \text{bmod} \) in the case when \( \rho \) is large enough.

2. Also run the compression algorithm of Chor and Goldreich [3] with FR. Memorizing the triples \((p, e, t)\) in each reduction and performing calculation on \( u \) and \( v \) only after a certain number of reductions (which make a phase) is sufficient. Parameter \( t \) occurs in FR. Parameter \( e \) is one single bit checking whether \( q = \lfloor pu/v \rfloor \) or \( q = \lceil pu/v \rceil \). It is easily defined from the parity of \( pu \mod v \): i.e., \( e = 0 \) if \( pu \mod v \) is even and \( e = 1 \) otherwise.

3. Correctly check whether \( R(u, v) = 0 \) or not, in order to perform a dichotomic method for finding the GCD.

4. Combine FR and KMR’s reductions, by using the minimal reduction in each step. The idea is to try and get as much information as possible from values \( pu \mod v \) (\( p = 0, 1, \ldots, k \)).

References


