Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT®

Journal of Computational and Applied Mathematics 162 (2004) 17–31

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.com/locate/cam

A modular reduction for GCD computation

Sidi Mohammed Sedjelmaci

LIPN CNRS UMR 7030, Université Paris-Nord, 99 Avenue J.B. Clément, 93430, Villetaneuse, France

Received 30 November 2001; received in revised form 5 November 2002

Abstract

Most of integer GCD algorithms use one or several basic transformations which reduce at each step the size of the inputs integers u and v. These transformations called *reductions* are studied in a general framework. Our investigations lead to many applications such as a new integer division and a new reduction called *Modular Reduction* or MR for short. This reduction is, at least theoretically, optimal on some subset of reductions, if we consider the number of bits chopped by each reductions. Although its computation is rather difficult, we suggest, as a first attempt, a weaker version which is more efficient in time. Sequential and parallel integer GCD algorithms are designed based on this new reduction and our experiments show that it performs as well as the Weber's version of the Sorenson's k-ary reduction.

1. Introduction

Given two integers u and v, the *greatest common divisor*, or GCD, of u and 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 algorithms for finding the GCD of two integers is Euclid's algorithm, which uses the GCD preserving transformations $(u, v) \mapsto (v, u \bmod v)$.

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. Although it is an improvement on the best known

E-mail address: sms@lipn.univ-paris13.fr (S.M. Sedjelmaci).

0377-0427/$ - see front matter
doi:10.1016/j.cam.2003.08.014

serial integer GCD algorithm $O(n(\log n)^2 \log\log n)$ of Schönhage [1,10], their method still requires n iterations; the parallelism only reduces the bit operations per iteration.

In 1987, Kannan, Miller and Rudolph (KMR) [6] 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. Since 1990, Chor and 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 $O_\varepsilon(n/\log n)$ using only $n^{1+\varepsilon}$ 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 [13] take $O_\varepsilon(n/\log n)$ time using at most $n^{1+\varepsilon}$ 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 the usual sublinear time parallel algorithms which usually "compress" multiple iterations (a "phase" [6]) of simpler algorithms into one iteration ("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 of nonnegative integers. Let u and v be two such (nonnegative) integers and a function $N^* \times N^* \to N^* \times N^*$, such that $(u, v) \mapsto (v, R(u, v))$, where R is a GCD "*reducing transformation*" defined within a domain $D \subseteq N^* \times N^*$.

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 [4,11,13,16] are practical and efficient; they use the k-ary reduction technique. Given an integer parameter $k > 0$ and 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

$$au + bv \equiv 0 \pmod{k} \quad \text{with } 0 < |a|,\ |b| < \sqrt{k}. \tag{1}$$

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

$$R(u, v) = |au + bv|/k < 2u/\sqrt{k} \tag{2}$$

the size of u is also reduced by roughly $\frac{1}{2}\log(k)$ 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 [16] and Jebelean [4] 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 transforma-

tion "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 [16].

The aim of this paper is to try to show what can we expect of such reducing transformations R in order to improve integer GCD algorithms. In the new reduction presented in the paper the size of v is reduced by $\log(k)$ bits. This reduction we call the *Modular Reduction* (MR).

In Section 2, definitions and basic properties of the fast reduction MR are given, as well as the main results of the paper: Theorems 6 and 10. Section 3 is devoted to an efficient way to compute MR. A parallel integer GCD based on our reduction is designed and analyzed in Section 4.

A comparison with other fundamental reductions with numerical experiments is presented in Section 5. Finally, concluding remarks are given in Section 6.

2. The modular reduction MR

2.1. Notation

Throughout, the following notation is used. Let $u \geqslant v > 2$ be positive integers. Let $k > 0$ be an integer parameter. We assume that k is a power of 2, i.e., $k = 2^m$. Given a nonnegative integer $x \in N$, $\ell_2(x)$ represents the number of its significant bits, not counting leading zeros:

$$\ell_2(x) = \begin{cases} \lfloor \log_2(x) \rfloor + 1 & \text{if } x \geqslant 1, \\ 1 & \text{if } x = 0. \end{cases}$$

Usually we denote by $\rho = \rho(u, v)$ the difference of significant bits between u and v plus one, namely $\rho = \rho(u, v) = \ell_2(u) - \ell_2(v) + 1$. Thus, we have $2^{\rho-2} < u/v < 2^{\rho}$. We let $\ell_2(u) = n$, so $2^{n-1} \leqslant u < 2^n$. Let a, b and c be positive integers.

The integer $x = a \bmod b$ is the unique nonnegative integer x such that

$$0 \leqslant x \leqslant b - 1 \quad \text{and} \quad a - x \equiv 0 \pmod{b}.$$

Note that this notation still holds when $a < 0$.

If b is relatively prime to k, then $c = a/b \bmod k$ is the unique nonnegative integer c such that

$$0 \leqslant c \leqslant k - 1 \quad \text{and} \quad cb \equiv a \pmod{k}.$$

Moreover if $c = u/v \bmod 2^{\rho}$ then $|u - cv|/2^{\rho}$ is a positive integer called the bmod, i.e.:

$$\operatorname{bmod}(u, v) = |u - cv|/2^{\rho}$$

2.2. Definition and basic properties

Definition 1. Let $u \geqslant v > 0$ be positive integers. We call a *reduction* any transformation R: $N^* \times N^* \to N^*$ satisfying the following two properties:

(P_1) $0 \leqslant R(u, v) < v$.
(P_2) $\gcd(v, R(u, v)) = \lambda \gcd(u, v)$, 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 λ which can easily be removed afterwards ([4,16]).

Definition 2. Let a and b be two rational numbers s.t. $ab \neq 0$. We call a *linear reduction* any reduction of the form $R(u,v) \stackrel{\text{def}}{=} |au + bv|$.

If a and b are integers then R is called integer linear reduction.

Example 3. The Euclid's reduction: $R(u,v) = u \bmod v = u - qv$, with $q = \lfloor u/v \rfloor$.

2.3. Integer reductions

Integer linear reductions are closely connected to the *extended Euclidean algorithm* (EEA). Proposition 5 emphasizes the fact that $\gcd(u,v)$ expresses as a minimum of linear reductions of u and v. We first give a technical lemma:

Lemma 4. *Let $u \geqslant v > 0$ and $a, b > 0$ be four integers such that $|au - bv| < v$ then*

$$b = \lfloor au/v \rfloor \quad \text{or} \quad b = \lceil au/v \rceil.$$

Proof. From $|au - bv| < v$ we obtain $-v < bv - au < v$ or $au/v - 1 < b < au/v + 1$ hence the result. □

Let R be an integer linear reduction. Since $R(u,v) < v$ then we have the following:

(1) There exists $a, b > 0$, s.t. R writes $R(u,v) = |au - bv|$.
(2) $R(u,v) = |au - bv| = (au) \bmod v$ or $v - (au) \bmod v$.
(3) $\gcd(v, |au - bv|) = \lambda \gcd(u,v)$, where $0 < \lambda$ divides a. □

Proposition 5. *For all integers $u \geqslant v > 0$,*

$$\gcd(u,v) = \min\{|au - bv| > 0 \text{ with } a, b \text{ integers } \geqslant 0\}$$
$$= \min\{au \bmod v > 0,\ v - (au) \bmod v > 0 \text{ with } 1 \leqslant a < v\}.$$

Proof. Let $m = \min\{|au - bv| > 0, \text{ with } a, b \text{ integers } \geqslant 0\}$ and $d = \gcd(u,v)$. If a and b are two integers such that $|au - bv| > 0$, then there exists an integer $\lambda \geqslant 1$ such that $|au - bv| = \lambda d$ hence $m \geqslant d$.

On the other hand the EEA computes, at each step i, integer coefficients a_i and b_i such that: $a_i b_i < 0$ and $a_i u + b_i v = r_i$, where $(r_i)_i$ is the sequence of the remainders in EEA. At the end of the algorithm we obtain a' and b' such that $d = \gcd(u,v) = a'u + b'v = |\alpha u - \beta v|$, with $a'b' < 0$, and $\alpha = |a'|$, $\beta = |b'|$. Hence $m \leqslant d$ and finally $m = d$.

Moreover if $d < v$ then $m = d < v$ and by Lemma 4

$$m = \min\{0 < |au - bv| < v\} = \min\{0 < au \bmod v,\ v - au \bmod v\}.$$

The case $d = v$ is obvious since for all $a > 0$, $au \bmod v = 0$ and $v - (au) \bmod v = v$ so $m = v$. □

Table 1
Examples of rational reductions

Name	Reduction	Property
Binary	$(u-v)/2$	u and v odds
bmod	$\lvert u-xv\rvert/2^{\rho}$	$x=(u/v) \pmod{2^{\rho}}$
Sorenson	$\lvert au+bv\rvert/k$	$au+bv\equiv 0 \pmod{k}$

However, for multiprecision inputs, EEA is not the best choice to compute the coefficients a and b in practice. In order to avoid expensive long divisions, Lehmer suggested [9] to extract the leading digits u_1 and v_1 of u and v, and run EEA on these single precision approximations of the inputs. KMR proposed [6] to compute the integer linear reduction $R(u,v)=au \bmod v$, in parallel for $a=1,2,\ldots,n$ with $n=\ell_2(u)$. Some improvements of Lehmer's approach [14,12] have been proposed later to compute more efficient integer reductions.

2.4. Rational reductions

It is easy to see that all the rational linear reductions are based on a modular relation

$$au+bu=0 \pmod{k}$$

and the associated reduction are then defined by $R(u,v)=|au+bv|/k$ (see Table 1). Let c denotes the modular quotient of u by v modulo k, i.e.: $c=(u/v) \bmod k$. There are several couples (a,b) satisfying the previous modular relation, namely those for which $b/a \equiv -c \pmod{k}$, or $b \equiv -ac \pmod{k}$. However, it is interesting to choose a couple (a,b) which minimizes the value of $|au+bv|$. We investigate this point in this section and define a new rational reduction for which the couple (a,b) not only satisfies $au+bv\equiv 0 \pmod{k}$ but also $|au+bv|<v$, so that $|au+bv|$ is itself a reduction (integer reduction). Thus, unlike the other reductions, the value of b satisfies $b=-\lfloor au/v\rfloor$ or $b=-\lceil au/v\rceil$ and therefore, we take also into account the order of magnitude of u/v. This reduction will be, in the remainder of the paper, called the *Modular Reduction* MR.

Theorem 6. *For all positive integers $u \geqslant v > k \geqslant 2$ s.t. $\gcd(v,k)=1$, there exists a couple (a,b) s.t. $1 \leqslant a \leqslant k-1$ and $b=\lfloor au/v\rfloor$ or $b=\lceil au/v\rceil$, which satisfies*

(a) $au-bv\equiv 0 \pmod{k}$.
(b) $|au-bv|<v$.

Proof. Consider the two sequences (r_i) and (d_i),

$$r_i \stackrel{\text{def}}{=} (iu) \bmod v, \ (0 \leqslant i < k) \quad \text{and} \quad r_k = v;$$

$$d_i \stackrel{\text{def}}{=} r_i \bmod k, \quad \text{for } i=0,1,\ldots,k.$$

For each $i=0,\ 1,\ldots,k$, (d_i) takes only k possible distinct values, whereas there is a total of $k+1$ d_is. Therefore, there obviously exist two distinct values i and j s.t. $0 \leqslant j < i \leqslant k$ and $d_i=d_j$. Set $\delta=r_i-r_j$. Then, $\delta \equiv 0 \pmod{k}$ and (a) holds.

Table 2
Computation of the couple (a,b)

	$\lvert\delta\rvert = \lvert r_i - r_j\rvert$	(a,b)
$0 \leqslant j < i \neq k$	r_{i-j}	$(i-j, \lfloor (i-j)u/v \rfloor)$
	or $v - r_{i-j}$	$(i-j, \lfloor (i-j)u/v \rfloor + 1)$
$1 \leqslant j < i = k$	$v - r_j$	$(j, \lfloor ju/v \rfloor + 1)$

It is easy to prove that $|\delta| = |r_i - r_j| = r_{i-j}$ or $v - r_{i-j}$, hence the result (b). Note that the case $i=k$ and $j=0$ is impossible since it implies that $\delta = v \equiv 0 \pmod{k}$ which contradicts the assumption $\gcd(v,k) = 1$. □

Table 2 summarizes this discussion.

Definition 7. Let (a,b) be one of the couples found in Theorem 6. The associated MR transformation is defined by $\mathrm{MR}(u,v) \stackrel{\text{def}}{=} |au - bv|/k$.

Note that, for a given $k > 2$, many such couples (a,b) could be found with $1 \leqslant a < k$ and for any one of them, we have (see Section 2.1)

$$\mathrm{MR}(u,v) = |au - bv|/k < v/k < \frac{u}{k2^{\rho-2}}.$$

Corollary 8. *For all positive integers $u \geqslant v > k > 2$ s.t. $\gcd(v,k) = 1$, there exists a parameter a s.t. $1 \leqslant a \leqslant k-1$ which satisfies*

$$au \bmod v \equiv 0 \pmod{k} \quad \textit{or}$$

$$au \bmod v \equiv v \pmod{k}.$$

We show in the following proposition that the smallest rational reduction is an MR reduction.

Proposition 9. *For all positive integers $u \geqslant v > k > 2$ s.t. $\gcd(v,k) = 1$, there exists a couple (a,b), $1 \leqslant a \leqslant k-1$ and $b = \lfloor au/v \rfloor$ or $b = \lceil au/v \rceil$, and an associated reduction* MR *such that*

$$\mathrm{MR}(u,v) = |au - bv|/k = \min_{1 \leqslant p < k} \{|pu + qv|/k,\ s.t : pu + qv \equiv 0 \pmod{k}\}.$$

Proof. As noticed before, for a given $k > 2$, many MR reductions can be found corresponding to several couples (a,b), $1 \leqslant a < k$. Let MR^* denotes the smallest of them, namely:

$$\mathrm{MR}^*(u,v) = |\alpha u - \beta v|/k = \min_{1 \leqslant a < k} \{\mathrm{MR}(u,v)\},$$

and let

$$m = \min_{1 \leqslant p < k} \{|pu + qv|/k,\ \text{s.t} : pu + qv \equiv 0 \pmod{k}\}.$$

Then from the previous theorem, we obtain:

$$m = \min_{1 \leqslant p < k} \{|pu + qv|/k, \text{ s.t : } pu + qv \equiv 0 \pmod{k} \text{ and } |pu + qv| < v\}$$

$$= \min_{1 \leqslant p < k} \{\mathrm{MR}(u, v)\} = \mathrm{MR}^*(u, v) = |\alpha u - \beta v|/k. \qquad \square$$

Theorem 10. *For all positive integers $u \geqslant v > k > 2$ s.t.* $\gcd(v, k) = 1$.

(i) *Any* MR *transformation is a reduction* (*in the sense of Definition* 1).

(ii) $\mathrm{MR}(u, v) = 0 \Leftrightarrow \gcd(u, v) = (v/a)\gcd(a, b) > v/k$.

Proof. (i) $\mathrm{MR}(u, v) < v/k < v$. Now, since $\gcd(v, k) = 1$, $\gcd(v, \mathrm{MR}(u, v)) = \gcd(v, k\mathrm{MR}(u, v))$ and hence,

$$\gcd(v, \mathrm{MR}(u, v)) = \gcd(v, |au - bv|) = \gcd(v, au) = \lambda \gcd(u, v) \text{ with } 1 \leqslant \lambda \mid a.$$

(ii) ($\Rightarrow$) If $\mathrm{MR}(u, v) = 0$, then $au = bv$. Now, let $d' = \gcd(a, b)$ and $d = \gcd(u, v)$. Then we have $a = d'a_1$, $b = d'b_1$, $u = du_1$ and $v = dv_1$ with $\gcd(a_1, b_1) = \gcd(u_1, v_1) = 1$.

Then, $a_1u_1 = b_1v_1$ and thus, $v_1 = a_1$ and $u_1 = b_1$. By substituting the values,

$$d = v/v_1 = \frac{v}{a/d'} \geqslant v/a > v/k;$$

which yields the sufficient condition.

($\Leftarrow$) If $d > v/k$, then from (i) we have $\gcd(u, v) = d \mid \mathrm{MR}(u, v)$. Let then $\mathrm{MR}(u, v) = td$, with $t \geqslant 0$. If $t \geqslant 1$, $v/k < d \leqslant \mathrm{MR}(u, v) < v/k$, which is impossible. Hence, $t = 0$ and so is $\mathrm{MR}(u, v)$. $\square$

Property (ii) checks a situation when the reduction is too big. The condition $\mathrm{MR}(u, v) = 0$ may be used as a stopping test for GCD algorithms since $\gcd(a, b) \leqslant a < k$. The division $v/(a/d')$ is then easily performed because it is exact (it is known in advance that the remainder is zero [5]).

3. The Computation of MR

Now we have to specify how to compute the couple (a, b) in MR. For this purpose, two approximations of the quotient $q_i = \lfloor iu/v \rfloor$ are proposed. We first define a new integer reduction which is similar to Euclid's reduction. A parallelized version is proposed in order to perform an efficient computation of MR. We let $\ell_2(u) = n$ and $\ell_2(v) = p$, $n \geqslant p$.

Proposition 11. *Let $u \geqslant v > 0$ and $q = \lfloor u/v \rfloor$. We consider a parameter λ s.t. $0 \leqslant \lambda \leqslant p$. We define u_1 and v_1 by: $u_1 = \lfloor u/2^{p-\lambda} \rfloor$ and $v_1 = \lfloor v/2^{p-\lambda} \rfloor$. We let $q' = \lfloor u_1/v_1 \rfloor$, then:*

$$(\lambda \geqslant n - p + 2) \Rightarrow (q' = q \text{ or } q + 1).$$

Proof. Let $u_2 = u \bmod 2^{p-\lambda}$ and $v_2 = v \bmod 2^{p-\lambda}$, so that

$$u = 2^{p-\lambda}u_1 + u_2 \quad \text{and} \quad v = 2^{p-\lambda}v_1 + v_2.$$

We have $q'v_1 2^{p-\lambda} + u_2 \leqslant u_1 2^{p-\lambda} + u_2 \leqslant ((q'+1)v_1 - 1)2^{p-\lambda} + u_2$. So $q'(v - v_2) \leqslant u < (q'+1)v$ since $u_2 < 2^{p-\lambda}$. Let $A = q'v_2/v$, then $A < 2^{n-p-\lambda+2}$ and

$$q' - 2^{n-p-\lambda+2} < q' - A \leqslant u/v < q' + 1,$$

hence the result for $\lambda \geqslant \rho + 1 \ = n - p + 2$. □

Remark 12. Note that it is very easy to compute u_1 and v_1: u_1 is the number obtained by the $(n - p + \lambda)$ first significant leading bits of u while v_1 is the number obtained by the λ first leading bits of v.

This result generalizes and improves a previous lemma of KMR in [6, p. 9]. They took $\lambda = n - p$ and obtain $|q'_i - q_i| \leqslant 3$.

3.1. The ρ-Euclid reduction

Applying the previous result to the smallest λ, i.e.: $\lambda = \rho + 1 = n - p + 2$, we obtain a new linear integer reduction.

Definition 13. With the notation described in Proposition 11, if u and v are such that $2p \geqslant n + 2$ and $\lambda = n - p + 2$, the ρ-Euclid transformation is defined by

$$R_\rho(u, v) \stackrel{\text{def}}{=} |u - q'v|.$$

Proposition 14. *The transformation ρ-Euclid is a reduction.*

Proof. First we note that $R_\rho = u \ mod \ v$ or $v - (u \bmod v)$. So we have just to prove that the case $R_\rho = v$ never occurs. This is derived by contradiction. Suppose $R_\rho = v$, then $q' = q + 1$ and $u \bmod v = 0$. By definitions of u_1 and v_1 we have

$$u_1/v_1 - 1 \geqslant \lfloor u_1/v_1 \rfloor - 1 = q' - 1 = q = u/v > \frac{2^{p-\lambda}u_1}{2^{p-\lambda}(v_1 + 1)} = \frac{u_1}{v_1 + 1}.$$

Thus $u_1/v_1 - 1 > u_1/(v_1 + 1)$ or $u_1/v_1 > v_1 + 1$. But $u_1/v_1 < 2^{n-p+1}$ and $v_1 \geqslant 2^{n-p+1}$ so we obtain $2^{n-p+1} + 1 \leqslant v_1 + 1 < u_1/v_1 < 2^{n-p+1}$, which is impossible. □

Remark 15. A new integer GCD algorithm similar to Euclid's one can be designed with R_ρ. However this algorithm avoids many long divisions. Moreover, we obtain another extended GCD algorithm, i.e.: we find a couple of integers (a, b) s.t.: $au + bv = \gcd(u, v)$ and this couple may differ from that of EEA. Note that a similar algorithm is proposed in [8, p. 376, exercise 30].

Example 16. Let $u = 26,977$ and $v = 8737$, we have

1101 00101100001 = 26,977

100 01000100001 = 8737.

We obtain $\lambda = n - p + 2 = 3$, $u_1 = 13$ and $v_1 = 4$ (the bits of u_1 and v_1 are written in bold). Thus $q' = \lfloor u_1/v_1 \rfloor = 3$ and $R_\rho = |u - 3v| = 766$.

3.2. A first approximation of q_i

For the sake of simplicity (shifting, computing "modulo", maintaining $\gcd(v,k)=1$, etc.), it is usually easier to let $k=2^m$, where the parameter m will be such that $m=\mathrm{O}(\log n)$.

In order to compute MR we shall apply the previous proposition to the couple (iu,v) for each i, $0<i<k=2^m$. Computing in parallel $R_\rho(iu,v)$, for each i with $\rho=\rho(iu,v)=\ell_2(iu)-\ell_2(v)+1$, implies different sizes for u_1 depending on the different values of $i=1,2,\ldots,k-1$. However, it is easier to choose a common value λ such that $\lambda\geqslant\rho_i+1$, for all i, namely $\lambda=2m$. On the other hand, this choice of λ allows a uniform framework needed in parallel computation processing. More precisely we prove the following.

Proposition 17. *Let $u\geqslant v>k=2^m>2$ such that $n-p+2\leqslant m\leqslant p/2$. Let $1\leqslant i\leqslant 2^m-1$ and $q_i=\lfloor iu/v\rfloor$. We define $(iu)_1=\lfloor(iu)/2^{p-2m}\rfloor$, $v_1=\lfloor(v)/2^{p-2m}\rfloor$ and $q'_i=\lfloor(iu)_1/v_1\rfloor$. If $\rho<m$ and $\lambda=2m$ then*:

$$q'_i=q_i \quad or \quad q_i+1.$$

Proof. Just use Proposition 11 with $u'=iu$ instead of u and $n'=\ell_2(iu)$ instead of $n=\ell_2(u)$. We obtain $\ell_2(iu)\leq\ell_2(i)+n\leqslant m+n$ and since $\rho<m$:

$$2m\geqslant m+n-p+2\geqslant\ell_2(iu)-p+2. \qquad \square$$

Recall that there exists (see Section 2, Corollary 8) an index i, $0<i<k=2^m$ such that

$$iu-q_iv\equiv 0 \quad \text{or} \quad v \pmod{2^m} \quad \text{(Test 1)}$$

and

$$\mathrm{MR}=|iu-q_iv|/2^m \quad \text{or} \quad |iu-(q_i+1)v|/2^m, \quad \text{where } q_i=\lfloor iu/v\rfloor.$$

It is worth noticing that we do not need to know exactly the value of q_i for computing MR. As a matter of fact, we can substitute q_i by $q'_i=\lfloor(iu)_1/v_1\rfloor$ and use the following test:

$$|iu-q'_iv|\equiv 0 \quad \text{or} \quad v \pmod{2^m} \quad \text{(Test 2)},$$

because

$$iu-q_iv\equiv 0 \quad \text{or} \quad v \pmod{2^m} \;\Leftrightarrow\; |iu-q'_iv|\equiv 0 \quad \text{or} \quad v \pmod{2^m}.$$

Now the computation MR is performed as follows:

In order to obtain the actual index i, more information is considered. The index i has to be such that: q'_i-1 or q'_i or $q'_i+1\equiv iu/v \pmod{2^m}$. When this latter relation holds, it causes to proceed to the computation of a new reduction MR_1 as follows:

$$\mathrm{MR}_1(u,v)=\begin{cases}|iu-(q'_i-1)v|/2^m & \text{if } q'_i-1\equiv iu/v \pmod{2^m},\\ |iu-q'_iv|/2^m & \text{if } q'_i\equiv iu/v \pmod{2^m},\\ |iu-(q'_i+1)v|/2^m & \text{if } q'_i+1\equiv iu/v \pmod{2^m}.\end{cases}$$

where i is one of the indices satisfying

$$q'_i - 1 \text{ or } q'_i \text{ or } q'_i + 1 \equiv iu/v \pmod{2^m} \quad \text{(Test 3)}.$$

It easy to check that $\mathrm{MR}_1(u,v) < 2v/k$. Moreover, we suggest to use *Priority* CRCW PRAM model in order to solve the write concurrency for such index i. In this CRCW PRAM model, the processor with the smallest index is allowed to write (see [7]). Otherwise, if Test 3 is false the calculation is stopped. Moreover, only the $\rho_i - 1 + \lambda$ first leading bits of iu and the λ first leading bits of v are needed for computing the value of q'_i. Taking $\rho < m$ yields $\rho_i = \rho(iu, v) \leqslant m + \rho \leqslant 2m - 1$. Thus, for computing MR_1 with Test 3, we need for each processor i, $(1 \leqslant i \leqslant 2^m - 1)$:

- Only the $4m - 2$ first bits of iu and the $2m$ first bits of v to compute q'_i.
- The m least significant bits of u and v to compute $iu/v \pmod{2^m}$.

3.3. A second approximation of q_i

In this section we suggest a way to avoid the multiprecision calculation of the product iu, $1 \leqslant i < 2^m$ and substitute $(iu)_1$ by iu_1. Only the $n - p + 2m$ first leading bits u are considered to obtain another approximation of q_i.

Proposition 18. *With the notations of Proposition* 17 $(\lambda = 2m)$, *we let* $q''_i = \lfloor iu_1/v_1 \rfloor$, *then there exists an index* i, $1 \leqslant i \leqslant 2^m - 1$ *such that*

$$q''_i = q_i - 1 \text{ or } q_i \text{ or } q_i + 1, \quad \text{and}$$

$$q''_i - 1 \text{ or } q''_i \text{ or } q''_i + 1 \text{ or } q''_i + 2 \equiv iu/v \pmod{2^m} \quad \text{(Test 4)}.$$

Proof. The proof proceeds from the same arguments as for Proposition 11 and the previous Section 3.2. □

3.4. Discussion and algorithms

We have proposed two approximations q'_i and q''_i of the actual quotient $q_i = \lfloor iu/v \rfloor$. Although less accurate than q'_i, the approximation q''_i seems to be more appropriate for computing MR because of its simplicity. We suggest another version of MR with q''_i called MR_2 defined by

$$\mathrm{MR}_2(u,v) = \begin{cases} |iu - (q''_i - 1)v|/2^m & \text{if } q''_i - 1 \equiv iu/v \pmod{2^m}, \\ |iu - q''_i v|/2^m & \text{if } q''_i \equiv iu/v \pmod{2^m}, \\ |iu - (q''_i + 1)v|/2^m & \text{if } q''_i + 1 \equiv iu/v \pmod{2^m}, \\ |iu - (q''_i + 2)v|/2^m & \text{if } q''_i + 2 \equiv iu/v \pmod{2^m}. \end{cases}$$

We can easily prove that MR_2 is also a reduction that satisfies: $\mathrm{MR}_2(u,v) < 3v/k \ = 3v/2^m$. We assume $u \geqslant v > k^2/2 = 2^{2m-1}$ with $\gcd(v,k) = 1$ and $n - p + 2 \leqslant m \leqslant p/2$. We give below a sequential and a parallel algorithm for computing MR_2 (Fig. 1).

$i := 0;$
repeat
$i := i + 1;$
$c := iu/v \pmod{2^m}$;
$q := \lfloor iu_1/v_1 \rfloor - 1;\ t := (c - q)\ mod\ 2^m$
until $t \leq 3$
$MR_2 := |iu - (q + t)v|/2^m$;
return MR_2.

Fig. 1. The sequential algorithm for computing MR_2.

Remark 19. The computation of i, q, and t are done in single precision. c and q can be performed simultaneously in parallel within the loop because c deals with the m least significant bits of u and v, while q deals with the $n - p + 2m$ first significant leading bits of u and the $2m$ first significant leading bits of v. Thus, no read concurrency could occur.

3.5. An example

Let $u = 56,149$ and $v = 34,195$, we obtain $\rho = 1$. We take $m = 3$, thus $\lambda = 2m = 6$, $u_1 = 54$ and $v_1 = 33$. Applying the previous sequential algorithm yields:

i	c	q	t
1	7	0	7
2	6	2	4
3	5	3	2

Finally for $i = 3$, we obtain

$$\mathrm{MR}_2(u, v) = |3u - 5v|/8 = 316.$$

Note that the computation of the Sorenson's reduction R_S yields for $k = 2^6$:

$$R_S = |7u - v|/64 = 5607.$$

4. The MR-GCD algorithm

Given integers $u \geqslant v > k > 2$ s.t. $\gcd(v, k) = 1$, we assume that when the algorithm starts, u is n bits large. We find it easier to take m as a "threshold" (the borderline choice between MR and the dmod reductions); but likewise, we might choose a varying threshold, depending upon v and experimental data [4,16].

4.1. High level description of a GCD algorithm

Step 1: Find d_1, s.t. d_1 equals the product of all common divisors to u and v which are less than k.

Step 2: Perform reductions until $v < k^2/2 = 2^{2m-1}$: if $\rho < m$, then perform MRs; else, perform the bmod reduction.

Compute $d = \gcd(u, v)$ with Euclid's algorithm, where (u, v) is the last pair satisfying $v < k^2/2$.

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:

repeat $R := |au - bv|/k$; /* in parallel by either MR_1 or MR_2 */
$R := R/2^t$; $(u, v) := (v, R)$ / * t is s.t. R is made odd */
until $v < k$

if $v = 0$ **then** $d' := \gcd(a, b)$; $d := d'v/a$
else $d := \gcd(u, v)$ /* perform bmods $(v < k^2/2)$ */

return d

Remark 20. It may be the case that computing the reduction MR 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 Kannan's algorithm and is also much alike Brent and Kung's binary algorithm.

4.2. Complexity analysis of the algorithm

With MR_1:

MR_1 is performed by using $k - 1 = O(2^m)$ processors working in parallel, where each processor computes the expression $iu - q'_i v$, for $i = 1, \ldots, k - 1$, as described in Section 3.2. Each iu can be achieved in constant time with $O(n2^{2m}) + O(n \log\log n)$ processors. Indeed, precomputed table lookup can be used for multiplying two m-bit numbers and computation modulo 2^m in constant time with $O(n2^{2m})$ processors in CRCW PRAM model, providing that $m = O(\log n)$ (see [13]).

Precomputed table lookup of size $O(m2^{2m})$ can be achieved in $O(\log m)$ time with $O(M(m)2^{2m})$, where $M(m) = m \log m \log\log m$ (see [13] or [3] for more details).

For each i, only the first $4m - 2$ and the last m bits are needed to determine $q'_i = \lfloor (iu)_1/v_1 \rfloor$ in $O(1)$ time within the same number of processors. Test 3 is applied to determine an index i and MR_1, and, finally, the computation of MR_1 takes $O(1)$ time with: $(\rho < m$ and $k = 2^m)$

$$O(n2^{3m}) + O(k\ n \log\log n) = O(n2^{3m}) \text{ processors.}$$

With MR_2:

The computation of MR_2 is easier because we do not need to compute all the products $iu,$: $i = 1, \ldots, k - 1$. Only the first $O(m)$ leading and the $O(m)$ least significants bits of u and v are required for computing $q''_i = \lfloor (iu_1/v_1 \rfloor$ in $O(1)$ time within the same number of processors. Test 4 is applied to determine an index i, t and MR_2. The computation of $MR_2 = |iu - (q + t)v|/2^m$ requires (see Fig. 2) only one product iu with the selected index i. Thus MR_2 can be achieved in parallel in $O(1)$ time with: $(\rho < m)$

$$O(n2^{2m}) + O(n \log\log n) = O(n2^{2m}) \text{ processors.}$$

For $i = 1, 2, .., 2^m - 1$ **Do in parallel**
$c := iu/v \pmod{2^m}$;
$q := \lfloor iu_1/v_1 \rfloor - 1$; $t := (c - q)\ mod\ 2^m$;
if $t \leq 3$ **then** $MR_2 := |iu - (q + t)v|/2^m$;
End Do
return MR_2.

Fig. 2. The parallel algorithm for computing MR_2.

MR (*either* MR_1 *or* MR_2) reduces the size of the largest input u by at least $m - 2$ bits. Hence the MR-GCD algorithm runs in $O(n/m)$ iterations. For $m = 1/2\varepsilon \log n$, $(\varepsilon > 0)$ the parallel MR-GCD algorithm matches the best previous GCD algorithms with $O_\varepsilon(n/\log n)$ time using only $n^{1+\varepsilon}$ processors on a CRCW PRAM.

5. Comparison to other reductions

5.1. Sorenson's reduction

As previously noticed, both MR and Sorenson's reduction R_S are based on the same modular relation $au + bv \equiv 0 \pmod{k}$, and use the same modular quotient $c = u/v \pmod{k}$. However, with R_S, we are not always sure that the inequality $|au + bv|/k < v$ holds and this restrict somehow the domain of use of R_S. We prove the following.

Lemma 21. *Sorenson's k-ary transformation is a reduction if* $k > 2^{2\rho+2}$.

Proof. It is easily seen that Sorenson's transformation satisfies $R(u, v) < 2u/\sqrt{k} < 2^{\rho+1}v/\sqrt{k}$. A sufficient condition for $R(u, v)$ to be a reduction is then $2^{\rho+1}v/\sqrt{k} < v$, that is $k > 2^{2(\rho+1)}$ or $\rho < m/2 - 1/2$. □

Indeed Weber [16] and Jebelean [4] use Sorenson's reduction only when $k > 2^{4\rho-2}$ or $\rho < m/4 + 1/2$, while MR can be used for $\rho < m$.

Actually one of the major drawbacks of R_S is that the value of $|au + bv|$ may be large especially when a and b are both positive (see the example in Section 3.5). Note that this case $(ab > 0)$ never occurs with MR.

5.2. Experiments

The implementation is written in C with GNU C Compiler *gcc*, version 2.7 (Stallman, 1991 [15]) on a Pentium III 667 MHz PC running Unix System 5. We have compared sequential algorithms computing the Weber's reduction WEB in version [16], MR_2 version of MR, bmod and Rho. The variable $\Delta = l(v) - l(R)$ represents the average number of bits chopped after each reduction. The average times are in milliseconds. The experiments were done on $N = 10,000$ random numbers u and v of size 50 and 60 bits respectively.

Table 3
Results for 50-bits random integers, $m = 4$ for MR, $m = 8$ for WEB

Reduction	WEB	MR	bmod	Rho
Average time	72.6	63.7	32.2	55.6
$\Delta = l(v) - l(R)$	4.272471	4.604829	2.814400	2.233300

Table 4
Results for 50-bits random integers, $m = 5$ for MR, $m = 10$ for WEB

Reduction	WEB	MR	bmod	Rho
Average time	74.0	65.6	32.2	55.6
$\Delta = l(v) - l(R)$	5.216611	5.588013	2.814400	2.233300

Table 5
Results for 60-bits random integers, $m = 4$ for MR, $m = 8$ for WEB

Reduction	WEB	MR	bmod	Rho
Average time	73.3	65.0	34.2	64.5
$\Delta = l(v) - l(R)$	4.228190	4.625653	2.766600	2.221800

Table 6
Results for 60-bits random integers, $m = 5$ for MR, $m = 10$ for WEB

Reduction	WEB	MR	bmod	Rho
Average time	75.4	66.0	34.2	64.5
$\Delta = l(v) - l(R)$	5.254600	5.622900	2.766600	2.221800

In order to compare MR_2 and WEB we must ensure the same size for the couples (a, b), s.t.: $|a|, |b| < k$. Thus, the parameters were $m = 4$ and 5 for MR_2 and, $m = 8$ and $m = 10$ for WEB respectively.

The results described in Tables 3–6 show that our reduction MR (MR_2 version) performs at least, as well as Weber's reduction WEB and it could therefore be used successfully in gcd computations.

Moreover, it is worth to note for larger size integers u and v, say 1000 to 10,000 bits (recall that $m = O(\log n)$, so $9 \leqslant m \leqslant 13$), the situation should be the same. As a matter of fact, the single precision computation of a and b deals only with the few most and the few least significant bits of u and v, the other bits are absolutely disregarded. Thus, since we consider random integers, the behaviour of the computation of a and b for each reduction is roughly the same for large integers.

6. Conclusion

Reductions techniques are widely used in most of integer gcd computations. These reductions are studied in a general framework. We have proposed a new reduction called MR which is optimal on a subset of rational reductions. Both sequential and parallel algorithms are proposed to compute this reduction and an integer gcd algorithm is designed. Our parallel integer gcd algorithm matches the best parallel performance of $O_\varepsilon(n/\log n)$ time with $n^{1+\varepsilon}$ processors on a CRCW PRAM.

Although the parallel complexity is still the same, our experiments are encouraging for this first attempt in the sequential version and we think that our algorithm can be improved. For example, it will be very interesting to provide a method with complexity $o(k)$ to find a "good" pair (a, b). This is our next direction of research.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
[2] R.P. Brent, H.T. Kung, Systolic VLSI arrays for linear-time GCD computation, in: F. Anceau, E.J. Aas (Eds.), Proceedings of VLSI'83, 1983, pp. 145–154.
[3] B. Chor, O. Goldreich, An improved parallel algorithm for integer GCD, Algorithmica 5 (1990) 1–10.
[4] T. Jebelean, A generalization of the binary GCD algorithm, in: Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'93, 1993, pp. 111–116.
[5] T. Jebelean, An algorithm for exact division, J. Symbolic Comput. 15 (1993) 169–180.
[6] R. Kannan, G. Miller, L. Rudolph, Sublinear parallel algorithm for computing the greatest common divisor of two integers, SIAM J. Comput. 16 (1) (1987) 7–16.
[7] R. Karp, V. Rammachandran, Parallel algorithms for shared-memory machines in: J. Van Leeuwen (Ed.), Algorithms and Complexity, Handbook of Theoretical Computer Science, Vol. A, MIT Press, Cambridge, MA, Elsevier, Amsterdam, 1990.
[8] D.E. Knuth, The Art of Computer Programming, 3rd Edition, Vol. 2, Addison-Wesley, Reading, MA, 1998.
[9] D.H. Lehmer, Euclid's algorithm for large numbers, Amer. Math. Monthly 45 (1938) 227–233.
[10] A. Schönhage, Schnelle Berechnung von kettenbruchentwicklungen, Acta Inform. 1 (1971) 139–144.
[11] M.S. Sedjelmaci, On a parallel Lehmer-Euclid GCD algorithm, Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'2001, 2001, pp. 303–308.
[12] M.S. Sedjelmaci, C. Lavault, Improvements on the accelerated integer GCD algorithm, Inform. Process. Lett. 61 (1997) 31–36.
[13] J. Sorenson, Two fast GCD algorithms, J. Algorithms 16 (1994) 110–144.
[14] J. Sorenson, An analysis of Lehmer's Euclidean GCD algorithm, Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'95, 1995, pp. 254–258.
[15] R.M. Stallman, Using and porting GCC, Free Software Foundation, Boston, 1991.
[16] K. Weber, Parallel implementation of the accelerated integer GCD algorithm, J. Symbolic Comput. (Special Issue on Parallel Symbolic Comput.) 21 (1996) 457–466.