ELSEVIER

Information Processing Letters 61 (1997) 31-36

Information Processing Letters

Improvements on the accelerated integer GCD algorithm

Mohamed S. Sedjelmaci [a], Christian Lavault [b,*]

[a] Computer Science Institute, University of Oran, Es-Senia, Algeria

[b] LIPN, CNRS URA 1507, Université Paris-Nord, Av. J.-B. Clément, 93430 Villetaneuse, France

Received 4 February 1996; revised 14 September 1996

Communicated by M.T. Goodrich

Abstract

The present paper analyses and presents several improvements to the algorithm for finding the (a,b)-pairs used in the k-ary reduction of the right-shift k-ary integer GCD algorithm. While the worst-case complexity of the "Accelerated integer GCD algorithm" is $O((\log_2(k))^2)$, we show that the worst-case number of iterations of the while loop is exactly $t(k) = \frac{1}{2}\lfloor \log_\phi(k) \rfloor$ (where $\phi = \frac{1}{2}(1+\sqrt{5})$). We suggest improvements on the latter algorithm and present two new faster residual algorithms: the sequential and the parallel one. A lower bound on the probability of avoiding the while loop in our parallel residual algorithm is also given.

Keywords: Parallel algorithms; Integer greatest common divisor (GCD); Parallel arithmetic computing; Number theory

1. Introduction

Given two integers a and b, the *greatest common divisor* of a and b, or $\gcd(a,b)$, is the largest integer which divides both a and b. Applications for integer GCD algorithms include computer arithmetic, integer factoring, cryptology and symbolic computation. Since Euclid's algorithm, several GCD algorithms have been proposed. Among others, the binary algorithm of Stein and the algorithm of Schönhage must be mentioned. With the recent emphasis on parallel algorithms, a large number of new integer GCD algorithms have been proposed. (See [5] for a brief overview.) Among these is the "right-shift k-ary" algorithm of Sorenson, which generalizes the binary algorithm. It is based on the following reduction. Given two positive integers $u > v$ relatively prime to k (i.e., (u,k) and (v,k) are coprime), pairs of integers (a,b) can be found that satisfy

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

If we perform the transformation (also called "k-ary reduction"):

$$(u,v) \longmapsto (u',v') = (|au+bv|/k, \min(u,v)),$$

which replaces u with $u' = |au+bv|/k$, the size of u is reduced by roughly $\frac{1}{2}\log_2(k)$ bits.

Note also that the product uv is similarly reduced by a factor of $\Omega(\sqrt{k})$, viz. uv is reduced by a factor $\geqslant k/(2\lceil\sqrt{k}\rceil) > \frac{1}{2}\sqrt{k} - \frac{1}{2}$ (see [5]). Another advantage is that the operation $au+bv$ allows quite a high degree of parallelization. The only drawback is that $\gcd(u',v)$ may not equal $\gcd(u,v)$ (we can only say that $\gcd(u,v) \mid \gcd(u',v)$), whence spurious fac-

* Corresponding author. Email: lavault@ura1507.univ-paris13.fr.

PII S0020-0190(96)00185-8

tors which must be eliminated. In the k-ary reduction, Sorenson suggests table lookup to find sufficiently small a and b such that $au + bv \equiv 0 \pmod{k}$. By contrast, Jebelean [2] and Weber [6,7] both propose an easy algorithm, which finds small a and b satisfying (1). This latter algorithm we call the "Jebelean–Weber Algorithm", or JWA for short, in the paper.

The worst-case time complexity of the JWA is $O((\log_2(k))^2)$. In this paper, we first show (Section 2) that the number of iterations of the while loop in JWA is exactly $t(k) = \frac{1}{2}\lfloor\log_\phi(k)\rfloor$ in the worst case, where ϕ is the golden ratio, $\frac{1}{2}(1+\sqrt{5})$, which makes Weber's result [7] more precise. In Section 3, we suggest improvements on JWA and present two new faster algorithms: the sequential and the parallel residual algorithm. Both run faster than the JWA (at least on the average), and their time complexity is discussed. In Section 4, a lower bound on the probability of avoiding the while loop of our parallel residual algorithm is given.

2. Worst case analysis of JWA

Let us first state the JWA, as presented in [7]: "the Accelerated GCD Algorithm", or "the General ReducedRatMod algorithm".

The Jebelean–Weber Algorithm (JWA)
Input: $x, y > 0$, $k > 1$, $\gcd(k,x) = \gcd(k,y) = 1$
Output: (n, d) such that $0 < n$, $|d| < \sqrt{k}$, and $ny \equiv dx \pmod{k}$
$c := x/y \bmod k$;
$f_1 = (n_1, d_1) := (k, 0)$;
$f_2 = (n_2, d_2) := (c, 1)$
while $n_2 \geqslant \sqrt{k}$ **do**
  $f_1 := f_1 - \lfloor n_1/n_2 \rfloor f_2$
  swap (f_1, f_2)
return f_2

The above version of the JWA finds small n and d such that $ny \equiv dx \pmod{k}$; $a = -d$ and $b = n$ meet the requirements of the k-ary reduction, so the need for large auxiliary tables is eliminated. Besides, the output from the JWA satisfies property (1). (The proof is due to Weber in [6,7].)

To prove Theorem 2 which gives the number of iterations of the while loop in JWA, we need the following technical Lemma 1 about Fibonacci numbers.

Lemma 1. *Let $(F_n)_{n\in\mathbb{N}}$ be the Fibonacci sequence defined by the recursion*

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \ (n \geqslant 2).$$

The following two properties hold:
(i) $n = \lceil\log_\phi(F_{n+1})\rceil$, *for* $n \geqslant 0$.
(ii) $(F_{\lceil n/2\rceil})^2 < F_n < (F_{\lceil n/2\rceil+1})^2$, *for* $n \geqslant 3$.

Proof. For $n \geqslant 2$, it is straightforward that $\phi^{n-1} < F_{n+1} < \phi^n$, where ϕ is the golden ratio. Hence (i) clearly holds. Now considering both cases when n is either even or odd yields the two inequalities in (ii). □

Theorem 2. *The number of iterations of the while loop of JWA is $t(k) = \frac{1}{2}\lfloor\log_\phi(k)\rfloor$ in the worst case.*

Proof. It is well known that, for $0 < u, v \leqslant N$, the worst-case number of iterations of the "Extended Euclidean Algorithm", or EEA for short, is $O(\log_\phi(N))$ (see [3,4]). Moreover, the worst case occurs whenever $u = F_{p+1}$ and $v = F_p$. The situation is very similar in JWA's worst case which also occurs whenever $u = F_{p+1}$ and $v = F_p$. However, the (slight) difference between the EEA and the JWA lies in the exit test. EEA's exit test is $n_{i+1} = 0$, where $n_i = \gcd(u, v)$, whereas JWA's exit test is

$$n_i < \sqrt{k} \leqslant n_{i-1}. \tag{2}$$

Let then $j = p - i$, the worst case occurs when $k = F_{p+1}$; and $c = F_p = n_0$. The number of iterations of the while loop is $t = i = p - j$, when i satisfies inequalities (2). In that case, $n_i = F_{p-i} = F_j$, and the exit test (2) may be written $(F_j)^2 < F_{p+1} < (F_{j+1})^2$. Thus, by Lemma 1, we have $j = \lceil\frac{1}{2}(p+1)\rceil$ and $t = p - \lceil\frac{1}{2}(p+1)\rceil$, which yields

$$t = \begin{cases} \frac{1}{2}p - 1 & \text{if } p \text{ is even, and} \\ \frac{1}{2}(p-1) & \text{if } p \text{ is odd.} \end{cases}$$

Hence, the worst case is when p is odd, and $t(k) = \frac{1}{2}\lfloor\log_\phi(k)\rfloor$. □

In JWA, Euclid's algorithm is stopped earlier. Yet, as shown in the above proof, the worst-case inputs remain the same as far when the algorithm runs to completion: $u = F_{p+1}$ and $v = F_p$, i.e. $k = F_{p+1}$ and $c = F_p$.

Example. Let $(k,c) = (F_{12}, F_{11}) = (144, 89)$. We get $t = 5$ by JWA as expected; and $t(144) = \frac{1}{2}\lfloor \log_\phi(144) \rfloor = 5$.

Notice that if $k = 2^{2l}$, the worst case never occurs since k cannot be a Fibonacci number. However, the case when $k = 2^{2l}$ corresponds to how the algorithm is usually used in practice; the "actual worst running time" is still less than the theoretical worst-case number of iterations of the algorithm. More precisely, when $k = 2^{2l}$ $t(k) = O(\log_2(k))$ only.

3. Two residual algorithms

In the sequel, we make use of the following notation: for $k \geqslant 4$, A_k, B_k and U_k are sets of positive integers defined by

$$A_k =]0, \sqrt{k}[, \quad B_k =]k - \sqrt{k}, k[, \quad U_k = A_k \cup B_k.$$

Definition 3. Let $(x,y) \in U_k \times U_k$. The *T-transformation* is defined as follows:
- if $x, y \in A_k$, then $T(x,y) = (x,y)$,
- if $x \in A_k$, $y \in B_k$, then $T(x,y) = (x, y-k)$,
- if $x \in B_k$, $y \in A_k$, then $T(x,y) = (k-x, -y)$,
- if $x, y \in B_k$, then $T(x,y) = (k-x, k-y)$.

Remark. The equivalent analytic definition of the T-transformation is

$$T(x,y) = ((1-2\chi(x))x + \chi(x)k,$$
$$(1-2\chi(x))(y - \chi(y)k)),$$

where χ is the characteristic function of the set B_k.

Proposition 4. *For every $(x,y) \in U_k \times U_k$, the pair $(x',y') = T(x,y)$ satisfies*
(i) $0 < x', |y'| < \sqrt{k}$.
(ii) $x'y \equiv xy' \pmod{k}$.

Proof. (i) If $k - \sqrt{k} < x < k$, then $0 < k - x < \sqrt{k}$ and $0 < |k - x| < \sqrt{k}$.

(ii) is easily derived from the definition of T. □

3.1. The residual algorithm

The Residual Algorithm *Res*
Input: $x, y > 0$, $k > 1$, $\gcd(k,x) = \gcd(k,y) = 1$
Output: (n,d) such that $0 < n$, $|d| < \sqrt{k}$, and $ny \equiv dx \pmod{k}$
$a := x \bmod k$;
$b := y \bmod k$
if $(a,b) \in U_k \times U_k$ **then** $f_2 := T(a,b)$
else $c := a/b \bmod k$
  if $c \in U_k$ **then** $f_2 := T(c,1)$
  else
    $f_1 = (n_1, d_1) := (k, 0)$;
    $f_2 = (n_2, d_2) := (c, 1)$
    while $n_2 \geqslant \sqrt{k}$ **do**
      $f_1 := f_1 - \lfloor n_1/n_2 \rfloor f_2$
      swap (f_1, f_2)
return f_2 /* $f_2 = Res(x,y)$ */

The worst-case complexity of the residual algorithm remains in the same order of magnitude as the JWA, viz. $O((\log_2(k))^2)$. However, the above algorithm runs faster *on the average*. The use of transformation T makes it possible to avoid the while loop quite often indeed. (See the related probability analysis in Section 4.)

For example, the residual algorithm provides an immediate result in the cases when $(a,b) \in U_k \times U_k$, or $c > k - \sqrt{k}$.

Note that the computational instruction $c := a/b \bmod k$ may be performed either by the euclidean algorithm [1], or by a routine proposed by Weber when k is an even power of two [6,7].

Since x and y are symmetrical variables, the same algorithm can also be designed with the instruction $s := b/a \bmod k$ instead of $c := a/b \bmod k$, and then by swapping n and d at the end of the algorithm. This remark leads to an obvious improvement about the residual algorithm: why not compute in parallel both c and s? The following parallel algorithm is based on such an idea.

3.2. The parallel residual algorithm

The Parallel Residual Algorithm *Pares*
Input: $x, y > 0$, $k > 1$, $\gcd(k,x) = \gcd(k,y) = 1$
Output: (n,d) such that $0 < n$, $|d| < \sqrt{k}$, and $ny \equiv dx \pmod{k}$
$a := x \bmod k$;
$b := y \bmod k$
  if $(a,b) \in U_k \times U_k$ **then** $f_2 := T(a,b)$
  else pardo

$v_1 := Res(a,b)$; $v_2 := Res(b,a)$
return f_2

v_1 and v_2 are two variables whose values are the result returned in the parallel computation performed by $Res(a,b)$ and $Res(b,a)$, respectively. The algorithm *Pares* ends when either of these two algorithms terminates.

$Res(a,b)$ is the residual algorithm described in Section 3.1 and $Res(b,a)$ is the following (very slightly) modified version of *Res*.

$s := b/a \bmod k$
if $s \in U_k$ **then** $f_2 := T(1,s)$
else
  $f_1 = (n_1, d_1) := (k, 0)$
  $f_2 = (n_2, d_2) := (s, 1)$
    while $n_2 \geqslant \sqrt{k}$ **do**
      $f_1 := f_1 - \lfloor n_1/n_2 \rfloor f_2$
      swap (f_1, f_2)
    endwhile
  swap (n_2, d_2)
return f_2

Remark. $Res(b,a)$ and $Res(b,a)$ are the only parallel routines performed in the algorithm *Pares*, and they are to both quit if either one finishes. Such a (quasi-) serial computation certainly induces an overhead on most parallel computers. Overhead costs may yet be reduced to a minimum thanks to a careful scheduling and synchronization of tasks and processors.

Note also that s may belong to U_k while c does not. This may be seen in the following example.

Example. Let $k = 1024$, $(a,b) = (263, 151)$, and $\sqrt{k} = 32$. Then, $c = a/b \bmod k = 273$, and $c \notin U_k$. But $s = b/a \bmod k = 1009 \in U_k$. So the while loop is simply avoided by performing $f_2 := T(1, 1009)$. Such an example shows that the parallel residual algorithm is very likely to run faster than its sequential variant – at least on the average.

4. Probability analysis

We first need a technical result to tackle the evaluation of the probability that the while loop is avoided in the parallel residual algorithm.

Lemma 5. *Let k be a square such that $k \geqslant 9$, and let $E_k = \{x \in \mathbb{N} \mid 1 \leqslant x \leqslant k$ and $\gcd(x,k) = 1\}$. Then, for all $x \in E_k$:*

$$(1 < x < \sqrt{k}) \implies (\sqrt{k} < 1/x \bmod k < k - \sqrt{k}). \tag{3}$$

Proof. Notice first that, obviously, there cannot exist any integer $1 < x < \sqrt{k}$ for $k = 1$ and $k = 4$; whence the statement of the lemma: $k \geqslant 9$.

Let $x \in E_k$ such that $1 < x < \sqrt{k}$, and let $y = 1/x \bmod k \in E_k$. The whole proof is by contradiction. First, on the assumption that $1 < x < \sqrt{k}$, suppose that $y \leqslant \sqrt{k}$. Hence, $xy < k$, and since $xy \equiv 1 \pmod{k}$ and $x > 1$, the contradiction is obvious. Thus $y = 1/x \bmod k > \sqrt{k}$.

Now, let us prove that $y < k - \sqrt{k}$ in (3). On the assumption that $1 < x < \sqrt{k}$, suppose also by contradiction that $y \geqslant k - \sqrt{k}$, with $\gcd(y,k) = 1$ and $y \leqslant k$. Let m, n be two non-negative integers, and let

$$x = \sqrt{k} - m, \qquad \text{where } 1 \leqslant m \leqslant \sqrt{k} - 2,$$
$$y = k - \sqrt{k} + n, \qquad \text{where } 0 \leqslant n \leqslant \sqrt{k}.$$

The upper bound on n may be reduced as follows: $n \neq \sqrt{k}$, since if $y = k$, $\gcd(y,k) \neq 1$, and $y \notin E_k$. So n is actually such that $0 \leqslant n \leqslant \sqrt{k} - 1$.

The product xy writes

$$\begin{aligned} xy &= (\sqrt{k} - m)(k - \sqrt{k} + n) \\ &= k(\sqrt{k} - m) + P(m,n) + 1 - k, \end{aligned}$$

where $P(m,n) = k - 1 - (\sqrt{k} - m)(\sqrt{k} - n)$.

Now we have that $xy \equiv 1 \pmod{k}$, and therefore, $P(m,n)$ must satisfy

$$P(m,n) \equiv 0 \pmod{k}. \tag{4}$$

From the bounds on m and n we can derive bounds on $P(m,n)$, which yields

$$\begin{aligned} k - 1 - (\sqrt{k} - 1)\sqrt{k} &\leqslant P(m,n) \\ &\leqslant k - 1 - [\sqrt{k} - (\sqrt{k} - 2)][\sqrt{k} - (\sqrt{k} - 1)], \end{aligned}$$

$$\sqrt{k} - 1 \leqslant P(m,n) \leqslant k - 3,$$

and, since $k \geqslant 9$,

$$1 < P(m,n) < k.$$

A contradiction with Eq. (4).

Remark. Lemma 5 is false when k is not a square: e.g., let $k = 17$. Then $x = 4$ and $y = 1/x \bmod k = 13$, while $k - \sqrt{k} < 17 - 4 = 13$.

Proposition 6. *Let k be a square such that $k \geqslant 9$. Let λ be a one-one mapping, $\lambda : E_k \longrightarrow E_k$, defined by $\lambda(x) = 1/x \bmod k$, and let φ denote Euler's totient function $\varphi(k) = \sum_{\gcd(j,k)=1, 0 \leqslant j < k} 1$. Then we have*
(i) $U_k \cap \lambda(U_k) = \{1, k-1\}$.
(ii) $|U_k \cup \lambda(U_k)| = 4\varphi(\sqrt{k}) - 2$.

Proof. Recall that

$$E_k = \{x \in \mathbb{N} \mid 1 \leqslant x \leqslant k \text{ and } \gcd(x,k) = 1\},$$
$$U_k = \{x \in E_k \mid 0 < x < \sqrt{k} \text{ or } k\sqrt{k} < x < k\},$$

and

$$\lambda(U_k) = \{y \in E_k \mid y = 1/x \bmod k\} \subset E_k.$$

(i) Obviously, 1 and $k-1$ belong to U_k. Let $x \in U_k$ such that $x \neq 1$ and $x \neq k-1$. By definition, x may belong to either distinct subset of U_k:

Case 1: $1 < x < \sqrt{k}$. By Lemma 5, $\lambda(x) \notin U_k$ and $\lambda(\lambda(x)) \notin U_k$.

Case 2: $k - \sqrt{k} < x < k - 1$. Let $x' = k - x$. The integers x and x' play a symmetrical role, which brings us back to Case 1, and $\lambda(x') \notin U_k$.

Hence, $\lambda(x') = \lambda(k-x) = k - \lambda(x) \notin U_k$. It follows that $\lambda(x) \notin U_k$, and $x = \lambda(\lambda(x)) \notin U_k$. Therefore, every integer $x \in U_k$ distinct from 1 and $k-1$ does not belong to $\lambda(U_k)$; whence equality (i).

(ii) The function λ being one-to-one, $|\lambda(U_k)| = |U_k|$. It yields

$$\begin{aligned}|U_k \cup \lambda(U_k)| &= |U_k| + |\lambda(U_k)| - |U_k \cap \lambda(U_k)| \\ &= 2|U_k| - |U_k \cap \lambda(U_k)|.\end{aligned}$$

Now $x < \sqrt{k}$ and $\gcd(x,k) = 1$, so $\gcd(x, \sqrt{k}) = 1$. Thus,

$$|\{x \in E_k \mid \gcd(x,k) = 1,\ x < \sqrt{k}\}| = \varphi(\sqrt{k}),$$

and $|U_k| = 2\varphi(\sqrt{k})$.

By equality (i), $|U_k \cap \lambda(U_k)| = |\{1, k-1\}| = 2$, and (ii) holds:

$$|U_k \cup \lambda(U_k)| = 4\varphi(\sqrt{k}) - 2. \qquad \square$$

From the previous results we can estimate the probability p_1 that $x \in U_k$ or $1/x \bmod k \in U_k$ when k is a square ($k \geqslant 9$). In particular we have the following theorem.

Theorem 7. *Let k be a square such that $k \geqslant 9$, and $p_1 = \Pr\{x \in U_k \vee 1/x \bmod k \in U_k\}$. Then*

$$p_1 = \frac{2}{\sqrt{k}}\left(2 - \frac{1}{\varphi(\sqrt{k})}\right).$$

Proof. $E_k = \{x \in \mathbb{N} \mid 1 \leqslant x \leqslant k \text{ and } \gcd(x,k) = 1\}$, and $|E_k| = \varphi(k)$. Let $x \in E_k$. If $x \notin U_k$ and $\lambda(x) = 1/x \bmod k \in U_k$, $x = \lambda(\lambda(x)) \in \lambda(U_k)$. Now, $|\lambda(U_k)| = |U_k| = 2\varphi(\sqrt{k})$. Let r be the number of integers $x \in E_k$ such that $x \in U_k$ or $1/x \bmod k \in U_k$. By Proposition 6, $r = |U_k \cup \lambda(U_k)| = 4\varphi(\sqrt{k}) - 2$, and $p_1 = r/\varphi(k)$. Since $\varphi(k) = \sqrt{k}\varphi(\sqrt{k})$, the result follows. $\square$

Remark. Among all possible values of k, the case when $k = 2^{2l}$ is especially interesting since it allows easy hardware routines. If $k = 2^{2l}$, $l \geqslant 2$, k is a square $\geqslant 9$ and Theorem 7 applies; since $\varphi(\sqrt{2^{2l}}) = \varphi(2^l) = 2^{l-1}$,

$$p_1 = 1/2^{l-2} - 1/2^{2l-2}.$$

Examples. (1) Let $k = 16$.

x	1	3	5	7	9	11	13	15
$1/x \bmod k$	1	11	13	7	9	3	5	15

$1, 3, 13, 15 \in U_{16}$, and also $1/5 \bmod 16$, $1/11 \bmod 16 \in \lambda(U_{16})$. So, the while loop is avoided 6 times (at least) among the 8 possible cases: $p_1 = 6/8$. Similarly, by Theorem 7, $r = 4\varphi(4) - 2 = 6$, $p_1 = \frac{1}{2}(2 - \frac{1}{2}) = 1 - \frac{1}{4} = \frac{3}{4}$. In that case, the while loop is avoided 75% of the time.

(2) Let $k = 64$. $U_{64} \cup \lambda(U_{64}) = \{1, 3, 5, 7, 9, 13, 21, 43, 51, 55, 57, 59, 61, 63\}$: $r = 14$, and $p_1 = 14/32 = (2^3 - 1)/2^4 = 7/16$. The while loop is avoided 14 times among the 32 possible cases, which corresponds to 43.75% at least.

It is worthwhile to notice that if $c = 39$, then $c \notin U_{64}$ and $s = 1/c \bmod 64 = 23 \notin U_{64}$. In some particular cases however, the while loop can still be avoided. This happens for example when $(a, b) = (3, 5)$: $c = 39 \notin U_{64}$, $s = 23 \notin U_{64}$, and yet the while loop is avoided since $(3, 5) \in U_{64} \times U_{64}$.

(3) Let $k = 2^{16}$ or $k = 2^{32}$. When dealing with 16-bits words, $p_1 = (2^8 - 1)/2^{14} \cong 1.55\%$. With 32-bits words, $p_1 = (2^{16} - 1)/2^{30} \cong 6 \cdot 10^{-3}\%$.

The latter examples show that p_1 is only a lower bound on the actual probability p of "systematically" avoiding the while loop, at *each iteration* of the parallel residual algorithm.

5. Summary and remarks

We proved that the number of iterations of the while loop in the worst case of the Jebelean–Weber algorithm equals $t(k) = \frac{1}{2}\lfloor \log_\phi(k) \rfloor$. We presented two new algorithms, the sequential and the parallel residual algorithm, which both run faster than the JWA (at least on the average). Preliminary experimentations on these algorithms meet the above results and confirm the actual and potential efficiency of the method. A lower bound on the probability of avoiding the while loop of the parallel residual algorithm was also given.

Our improvements have certainly more effect when k is small, and this is precisely the case when using table-lookup is more efficient than the use of the JWA. But even if such improvements might seem negligible for only few iterations of our algorithm, avoiding the inner loop several times repeatedly makes them significant indeed in the end.

References

[1] R.T. Gregory and E.V. Krishnamurthy, *Methods and Applications of Error-Free Computation* (Springer, New York, 1984).

[2] T. Jebelean, A generalization of the binary GCD algorithm, in: *Proc. Internat. Symp. on Symbolic and Algebraic Computation (ISSAC'93)* (1993) 111-116.

[3] D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms, Vol. 2* (Addison Wesley, Reading, MA, 2nd ed., 1981).

[4] G. Lamé, Note sur la limite des diviseurs dans la recherche du plus grand commun diviseur entre deux nombres entiers, *C.R. Acad. Sci. Paris* 19 (1844) 867-870.

[5] J. Sorenson, Two fast GCD algorithms, *J. Algorithms* 16 (1994) 110-144.

[6] K. Weber, The accelerated integer GCD algorithm, Dept. of Mathematics and Computer Science, Kent State University, 1995.

[7] K. Weber, Parallel implementation of the accelerated integer GCD algorithm, *J. Symbolic Comput.* (Special Issue on Parallel Symbolic Computation) (1996), to appear.