On a Parallel Lehmer-Euclid GCD Algorithm

Sidi Mohammed Sedjelmaci
LIPN CNRS UPRES-A 7030,
Université Paris-Nord 93430 Villetaneuse,France.
sms@lipn.univ-paris13.fr

ABSTRACT

A new version of Euclid's GCD algorithm is proposed. It matches the best existing parallel integer GCD algorithms since it can be achieved in $O_\epsilon(n/\log n)$ time using at most $n^{1+\epsilon}$ processors on CRCW PRAM.

1. INTRODUCTION

The problem of the *Greatest Common Divisor* (or GCD) of two integers is important for two major reasons. First because it is widely included as a low operation in several arithmetic packages. On the other hand, despite its amazing simplicity, the complexity of the GCD problem in parallel is still unknown. We do not know whether it belongs to the NC class or if it is a P-complete problem.

The advent of practical parallel computers has caused the re-examination of many existing algorithms with the hope of discovering a parallel implementation. 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 was $O(n \log\log n/\log n)$ assuming there are $n^2(\log n)^2$ processors working in parallel, where n is the bit-length of the larger input. Since 1990, Chor and Goldreich [3] 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 is $O_\epsilon(n/\log n)$ using only $n^{1+\epsilon}$ processors on a CRCW PRAM. More recently (1994), Sorenson's right- and left-shift k-ary algorithms [13] match Chor and Goldreich's performance.

Euclid's algorithm is one of the simplest and most popular integer GCD algorithm. Its extended version called *Extended Euclidean Algorithm* or *EEA* for short [9] is tightly linked with the continued fractions [4, 9] and is important for its multiple applications (cryptology, modular inversion, etc..). In [7], Kannan, Miller and Rudolph proposed a first parallelization of EEA. Their algorithm was based on a reduction step which uses a non trivial couple (a, b) of integers $|a| \le kv/u$, $|b| \le 2k$, s.t. $0 \le au - bv \le u/k$ for a given parameter $k > 0$; therefore, at each step, the larger input u is reduced by $O(\log k)$ bits.

However, one of the major drawbacks of their algorithm is the expensive cost of the computation (a, b). As a matter of fact, in order to reach an $O(1)$ time computation for their reduction step, more than $O(n^2 \log^2 n)$ processors are needed to compare in pairs the $O(n)$ numbers $au - bv$ of $O(\log^2 n)$ bits (see [7] for more details). The main results of the paper are summarized below:

- We propose a new reduction step which is easily obtained from the $O(\log n)$ first significant leading bits of the inputs.
- Based on this reduction step, new sequential and parallel GCD algorithms are designed. The parallel algorithm matches the best known GCD algorithms: its time complexity is $O_\epsilon(n/\log n)$ using only $n^{1+\epsilon}$ processors on a CRCW PRAM, for any constant $\epsilon > 0$.
- Moreover, a compression method may be considered to improve the complexity.

In Section 2, we recall the basic reduction step used in Kannan, Miller and Rudolph's algorithm [7]. In Section 3, we define a new reduction which uses the same Lehmer idea [10]. Basically, with the $O(\log n)$ most significant bits of u and v, we can easily find a couple (a, b) such that the associated reduction satisfies $|au-bv| < 2v/k$, with $1 \le a \le k$, for a given parameter $k = O(n)$. A new parallel integer GCD algorithm based on our reduction is designed in Section 4. Section 5 is devoted to the complexity analysis. Comparisons with other reductions as well as preliminary experiments are given in Section 6. We conclude with some remarks in Section 7.

ISSAC 2001, 7/01, Ontario, Canada
©2001 ACM 1-58113-417-7/ 01/ 0007 $5.00

2. BASIC REDUCTION STEPS

2.1 Notation

Throughout this paper, we restrict ourselves to the set of non-negative integers. Let u and v be two such (non-negative) integers, u and v are respectively $n - bits$ and $p - bits$ numbers with $u \ge v$. Let k be an integer parameter s.t. $k = 2^m$

with $m \geq 2$ and $m = O(\log n)$.

EEA denotes the Extended Euclidean Algorithm. If many processors are in write concurrency then the Concurrent Read and Concurrent Write model "$CRCW$" of PRAM is considered. There are many submodels of CRCW PRAM for solving the write concurrency, however in order to allow the priority to the processor with the larger index, we choose the *Priority* sub-model [8].

Most of serial integer GCD algorithms use one or several transformations $(u, v) \longmapsto (v, R(u, v))$ which reduce the size of current pairs (u, v), until a pair $(u', 0)$ is eventually reached. The last value $u' = \gcd(u, v)$ is the result we want to find. These transformations will be called *Reductions* if they satisfy the following two properties:

(P_1) $0 \leq R(u, v) < v$.
(P_2) $\gcd(v, R(u, v)) = \alpha \gcd(u, v)$, with $\alpha > 0$.

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 α which can easily be removed afterwards [5, 16]. Examples of such basic reductions are given in Table 1.

Given a non-negative integer $x \in N$, $\ell_2(x)$ represents the number of significant bits of an non-negative integer x, not counting leading zeros:

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

So $n = \ell_2(u)$, $p = \ell_2(v)$ and p satisfies $2^{p-1} \leq v < 2^p$. We let $\rho = \rho(u, v) = \ell_2(u) - \ell_2(v) + 1$. Thus, we obtain $2^{\rho-2} < u/v < 2^\rho$. We assume that $p > 2m + 3$.

As noticed by many authors the main difficulty in GCD algorithms happens when the input data u and v are roughly of the same size [7, 5, 16]. So we shall assume that when we apply our reduction: $n - p < m - 1$ (or $\rho < m$). Otherwise, we apply a more efficient reduction: the *bmod*, defined as:

$$bmod(u, v) = |u - (u/v \mod 2^\rho)v|/2^\rho.$$

For any parameter λ s.t. $0 \leq \lambda \leq p$ we define u_1 and v_1 by:

$$u_1 = \lfloor u/2^{p-\lambda} \rfloor \text{ and } v_1 = \lfloor v/2^{p-\lambda} \rfloor.$$

Let $u_2 = u \bmod 2^{p-\lambda}$ and $v_2 = v \bmod 2^{p-\lambda}$, then $u_2, v_2 < 2^\lambda$ and

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

The numbers u_1 and v_1 are obtained with, respectively, the $\lambda + n - p$ and the λ most significant leading bits of u and v.

2.2 The Kannan, Miller and Rudolph Reduction

The Kannan, Miller and Rudolph (KMR for short) integer GCD algorithm is based on the following lemma ([7], page 9):

LEMMA 2.1. *(KMR lemma)*
For all positive integers u, v and k with $u \leq kv$, there exists a couple $(a, b) \neq (0, 0)$ s.t. $|a| \leq kv/u$ and $|b| \leq 2k$ which satisfies $0 \leq au - bv \leq u/k$.

Remark: Since $|au - bv| \leq u/k \leq v$, then $|au - bv| = (au) \pmod k$ or $v - au \pmod k$.

Thus any couple (a, b) found provides a reduction step called a KMR's reduction. Kannan, Miller and Rudolph proposed to compute in parallel all the $O(k^2)$ numbers $au - bv$, and select those for which $0 \leq au - bv \leq u/k$. But this latter relation implies $|au - bv| \leq u/k$, thus the couple (a, b) must be chosen from a set of $O(k)$ numbers satisfying this relation. However, the pair (a, b) in [7] is not easily obtained. Although $O(\log^2 n)$-bit numbers are considered at each step, $O(n)$ numbers $au - bv$ must be compared in pairs. Therefore, more than $O(n^2 \log^2 n)$ processors are needed in order to compute their reduction in constant time.

3. THE IMPROVED LEHMER-EUCLID REDUCTION

The main difficulty in EEA is the expensive cost of long divisions when we deal with large size inputs. In 1938, Lehmer [10] suggested another way to compute the couple (a, b). Roughly speaking (see Knuth [9] for more details), working only with the leading bits of u and v, the author considers two single-precision rationals which approximate the quotient u/v, namely $u'/v' < u/v < u''/v''$. Thus if we carry out EEA simultaneously on the single-precision rationals u'/v' and u''/v'' until we get a different quotient, we obtain the same sequence of quotients had we applied the multi-precision numbers u and v.

Let (a, b) be the last couple obtained by EEA. Then the transformation $R(u, v) = au + bv$ is a *reduction* in the sense of Section 2. However, for random inputs u and v, the sequence of same quotients seems to be equally random. Although a first attempt was made by Sorenson [14] with a slightly modified version of Lehmer's algorithm, no *a priori* estimation of this reduction is known. The author only gave an asymptotic behavior of his reduction since he obtained (with our notation) [14]:

$$R(u, v) = au + bv = O(u/2^m).$$

The reduction we propose in this paper is also based on leading bits and continuants but, by contrast, our reduction satisfies $R(u, v) < 2v/k$ for any positive parameter k. We first specify how to compute the couple (a, b) of the reduction then both a sequential and a parallel version of a GCD algorithm are proposed.

Name	Reduction	Property
Euclid	$u - qv$	$q = \lfloor u/v \rfloor$
binary	$(u - v)/2$	u and v odds
bmod	$\|u - xv\|/2^\rho$	$x = (u/v) \pmod{2^\rho}$
Sorenson	$\|au + bv\|/k$	$au + bv \equiv 0 \pmod{k}$

Table 1: Examples of Reductions.

LEMMA 3.1. *For all positive integers $u \geq v$, $k = 2^m$ and $p = \ell_2(v) > 2m + \rho$, there exists a couple of integers $(a, b) \neq (0, 0)$ s.t. $1 \leq a \leq k$ which satisfies $0 \leq |au - bv| < 2v/k$.*

Moreover, the couple (a, b) is only obtained from the first $2m + 2\rho$ leading bits of u and the first $2m + \rho + 1$ leading bits of v respectively.

PROOF. Let $\lambda = 2m + \rho + 1$ and u_1 and v_1 as previously defined in Section 2. Note that u_1 and v_1 exist since $p - \lambda \geq 0$. Applying Hardy and Wright's theorem [4] (theorem 36, p.30) to the rational v_1/u_1 with $k' = \lceil ku_1/v_1 \rceil$, we obtain a couple (a, b) of integers s.t.

$$1 \leq a, b \leq k' - 1 \quad and \quad |v_1/u_1 - a/b| \leq 1/k'b$$

hence

$$|au_1 - bv_1| \leq u_1/k' \leq \frac{u_1}{\lceil ku_1/v_1 \rceil} \leq v_1/k.$$

We let $R = |au - bv|$. We obtain

$$R = |au - bv| \leq |au_1 - bv_1| 2^{p-\lambda} + |au_2 - bv_2|$$

$$R < 2^{p-\lambda} v_1/2^m + 2^{p-\lambda} k'$$

$$R < v/2^m + 2^{p-\lambda+m+\rho}.$$

From $\lambda = 2m + \rho + 1$, we obtain

$$R < 2v/k.$$

Moreover we have $|a - bv_1/u_1| \leq 1/k'$ so $a \leq bv_1/u_1 + 1/k'$, but $b \leq k' - 1 < ku_1/v_1$ so $a < k + 1/k'$ and

$$a \leq \lfloor k + 1/k' \rfloor = k.$$

□

Remarks

• Note that $b \leq \lceil ku_1/v_1 \rceil - 1 < ku_1/v_1$. The previous reduction satisfies $R = |au - bv| = (au) \pmod{k}$ or $v - au \pmod{k}$ since $R < 2v/k < v$; $m \geq 2$.

• The constant 2 in the inequality is less precise and our first experiments show that, most of the time, we have $R < v/k$ (see Table 2).

DEFINITION 3.1. *Let (a, b) be one of the couples defined in Lemma 3.1. The R_{ILE} transformation is defined by*

$$R_{ILE}(u, v) \overset{def}{=} |au - bv|.$$

Many such couples can be found and R_{ILE} depends on the couple (a, b) considered, but for any one of them, R_{ILE} is a *reduction* which satisfies $R_{ILE}(u, v) = |au - bv| < 2v/k$. We propose in the next Section an easy way to compute one of these couples (a, b) and the reduction R_{ILE}.

3.1 The Algorithms

We give below a sequential and a parallel algorithm for computing our reduction R_{ILE}.

```
Input: $u \geq v$ and $k = 2^m$ s.t. $\rho = n - p + 1 < m$ and $p > 2m + 3$.
Output: $R_{ILE}(u, v)$.

Step 1
    $p := l_2(v)$;
    $\lambda := 2m + \rho + 1$;
    $u_1 := \lfloor u/2^{p-\lambda} \rfloor$; $v_1 := \lfloor v/2^{p-\lambda} \rfloor$;
    $u_2 := u \pmod{2^{p-\lambda}}$; $v_2 := v \pmod{2^{p-\lambda}}$;

Step 2 Run EEA with the couple $(u_1, v_1)$ and compute successive triplets $(r, b, a)$, where $r = au_1 + bv_1$ until $|a| > 2^m$. Save the previous triplet $(r, a, b)$. Note that $ab < 0$.

Step 3 Compute $R_{ILE} = |au + bv| = |r2^{p-\lambda} + au_2 + bv_2|$;

Return $R_{ILE}$.
```

Fig. 1 *The Sequential Algorithm for computing R_{ILE}.*

All the triplets (r, a, b) computed in EEA satisfy $b = \lfloor au_1 \rfloor$ or $b = \lfloor au_1 \rfloor + 1$. Therefore r expresses as $r = au_1 \bmod v_1$ or $r = v_1 - (au_1 \bmod v_1)$ and the previous algorithm is easily parallelized as follows.

```
Input: $u \geq v$ and $k = 2^m$ s.t. $\rho = n - p + 1 < m$ and $p > 2m + 3$.
Output: $R_{ILE}(u, v)$.

Step 1
    Compute $p$, $\lambda$, $u_1$, $u_2$, $v_1$ and $v_2$ as in Figure 1.

Step 2 For $i = 1, 2, .., 2^m$ Do in parallel
    $q_i := \lfloor iu_1/v_1 \rfloor$; $r_i := iu_1 - q_i v_1$;
    if $r_i < v_1/k$ then $r := r_i$; $a := i$; $b := q_i$;
    if $v_1 - r_i < v_1/k$ then $r := v - r_i$; $a := i$; $b := q_i + 1$;
End Do
```

Step 3 Compute in parallel $R_{ILE} = |au - bv| = |r2^{p-\lambda} + au_2 - bv_2|$;

Return R_{ILE}.

Fig. 2 *The Parallel Algorithm for computing* R_{ILE}.

If many processors are in write concurrency in Step 2 then we use the *Priority* sub-model of CRCW-PRAM. With this sub-model, we obtain the largest a s.t. $1 \le a \le k$ and the smallest reduction R_{ILE}.

Remark: Let $m = O(\log n)$. Even when u and v are very large numbers in size (up to 65,536-bits, $n, p \le 2^{16}$) the computations in Step 1 and 2 can be performed in constant time with a single precision since $\log n \le 16$ (see Section 5).

3.2 An Example

Let $u = 1,759,291$ and $v = 1,349,639$. Their binary representations are respectively:

11010110 1100000111011 $= 1,759,291$
10100100 1100000000111 $= 1,349,639$

We obtain $n = p = 21$ so that $\rho = 1$. If we take $m = 3$, we obtain $\lambda = 2m + 2 = 8$, $u_1 = 214$ and $v_1 = 164$ (the bits representing u_1 and v_1 are in bold). Applying the EEA to u_1 and v_1 yields the first successive integers q, r, b and a ($r = au + bv$).

q	r	b	a
	214	0	1
	164	1	0
1	50	−1	1
3	14	4	−3
3	8	−13	+10

In our example we obtain $a = -3$, $b = 4$, $r = 14 < v_1/k = 164/8 = 20.50$ and

$$R_{ILE} = |-3u + 4v| = 120,683.$$

Note that $R_{ILE} < v/k = 168,704.88$. On the other hand the computation of the *bmod* reduction and that of Sorenson R_S yield: (see Section 6.2 for the computation of R_S)

$bmod(u, v) = |u - v|/2 = 204,826$ and
$R_S = |7u + 5v|/64 = 297,863$, with $k = 64$,
$R_S = |u + 3v|/16 = 363,013$, with $k = 16$.

4. ILE-GCD: THE IMPROVED LEHMER-EUCLID GCD ALGORITHM

Given integers $u \ge v > k > 0$ s.t. $\gcd(v, k) = 1$, we assume that when the algorithm starts, u is n bits large. Recall that the parameter m is such that $m = O(\log n)$ for R_{ILE} thus this value yields at most $O(n/\log n)$ iterations.

As to the stop test in the routine, we use $v \ge 8k^2$ (R_{ILE} is undefined when $v < 8k^2$). We find it easier to take m as a "threshold" (the borderline choice between R_{ILE} and the *bmod* reductions); likewise, we might choose a varying threshold, depending upon v and experimental data [5, 16].

4.1 High Level Description

Step 1: Let us note $d = \gcd(u, v)$. Find d_1, s.t. d_1 equals the product of all common divisors of u and v which are less than k.

Step 2: Perform reductions until $v < 8k^2$: if $\rho < m$, then perform R_{ILE}'s; else, perform the *bmod* reduction.

Compute $d = \gcd(u, v)$ with *Euclid*'s (or *bmod*) algorithm, where (u, v) is the last pair satisfying $v < 8k^2$.

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

Step 4: Return $d \times d_1$.

Fig. 3 *The Parallel ILE-GCD Algorithm.*

Step 1, 3 and 4 are similar to the phases in KMR algorithm. Step 2 is designed in Figure 2.

5. COMPLEXITY ANALYSIS

We give below the complexity analysis of the parallel ILE-GCD Algorithm. First note that the computation of $\ell_2(u)$ and $\ell_2(v)$ can be computed in $O(1)$ time in parallel with $O(n)$ processors in CRCW (Priority). Observe that u_1 and v_1 can be found by extraction; $2^{p-\lambda}$ is not needed, nor is the multiprecision division.

We compute $r_i = iu_1 - q_i v_1$ and test if $r_i < v_1/k$ or $v_1 - r_i < v_1/k$ to select the index i. Then $iu_2 - q_i v_2$ can be computed in parallel as well as $R_{ILE} = |2^{p-\lambda} r + iu_2 - q_i v_2|$. All these computations can be done in $O(1)$ time with $O(n2^{2m}) + O(n \log\log n)$ processors. Indeed, precomputed table lookup can be used for multiplying two m-bit numbers 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 carried out in $O(\log m)$ time with $O(M(m)2^{2m})$ processors, where

$$M(m) = m \log m \log\log m$$

(see [13] or [3] for more details).

The computation of $R_{ILE} = |iu - q_i v|$ requires (see Figure 2) only two products iu and $q_i v$ with the selected index i. Thus R_{ILE} can be computed in parallel in $O(1)$ time with: ($\rho < m$)

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

R_{ILE} reduces the size of the smallest input v by at least $m-1$ bits. Hence the $ILE-GCD$ algorithm runs in $O(n/m)$ iterations. For $m = 1/2\ \epsilon \log n$, ($\epsilon > 0$) the parallel $ILE-GCD$ algorithm matches the best previous GCD algorithms in $O_\epsilon(n/\log n)$ time using only $n^{1+\epsilon}$ processors on a CRCW PRAM.

6. COMPARISON TO OTHER REDUCTIONS

6.1 Sorenson's Reduction

Sorenson's reduction R_S is based on the modular relation

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

and R_S is defined by $R_S(u,v) = |au + bv|/k$. However the inequality $|au+bv|/k < v$ does not always hold with R_S and this somehow restrict the domain of use of R_S. We prove the following.

LEMMA 6.1. *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 [5] use Sorenson's reduction only when $k > 2^{4\rho-2}$ or $\rho < m/4 + 1/2$, while R_{ILE} can be used for $\rho < m$.

Actually one of the major drawbacks of R_S is that the value of $|au+bv|$ might be large, especially when a and b are both positive (See the example given in Section 3.2).

6.2 The Weber's Algorithm

Sorenson suggested in [13] table lookup to find a couple of integers (a,b) satisfying relation (1). By contrast, Jebelean [5] and Weber [16] both propose an easy algorithm which finds such couple (a,b). These couples (a,b) are obtained by running EEA (with one column added instead of two) until $b < k$. We give below this algorithm.

Input: x, $y > 0$, $k > 1$, and $\gcd(k,x) = \gcd(k,y) = 1$.
Output: (n,d) s.t. $0 < n$, $|d| < \sqrt{k}$, and $ny \equiv dx \pmod{k}$.

```
$c := x/y \bmod k$ ;      /* initialization */
$f_1 = (n_1, d_1) := (k, 0)$ ;
$f_2 = (n_2, d_2) := (c, 1)$
while $n_2 \geq \sqrt{k}$ do
    $f_1 := f_1 - \lfloor n_1/n_2 \rfloor f_2$
    swap $(f_1, f_2)$
endwhile
return $f_2$
```

Fig. 4 *The Weber's Algorithm for computing the couples* (a,b) *of* R_S.

Reduction	R_S	R_{ILE}
Average Time	0.05	0.05
Average Ratio R/v	.079343	.058005

Table 2: First Results.

The reductions R_{ILE} and R_S are theoretically quite similar. However the key problem for reductions is the computation cost of the couples (a,b). It is worth noting that the computation of R_{ILE} is easier than that of R_S proposed by Weber. As a matter of fact, if we compare the algorithms for computing both R_S and R_{ILE}, we can observe (see Fig. 3 and 4) that the computation of R_S proposed by Weber needs an extra amount of time spent for computing the modular quotient $c = x/y \bmod k$.

6.3 First Experiments

Due to the similarity between the GCD algorithms based on R_S and R_{ILE} [16], it is sufficient to compare the reductions R_S and R_{ILE} with each other rather than with all the GCD algorithms.

We have compared sequential algorithms for computing R_S and R_{ILE} reductions. The implementation is written in C with GNU C Compiler *gcc* (Stallman, 1991 [15]) on Unix system. The average times are in seconds. The source files were not optimized and R_S is used as a benchmark.

The experiments were done on N random numbers u and v of size 30 to 32 bits, $20 \leq N \leq 50$. The parameters were $m = 3$, $\lambda = 10$, thus only the reductions R_S and R_{ILE} are considered since $\rho \leq 3$.

Our preliminary results are described in Table 2. It seems that for the same average time, the average ratio R/v is slightly better when R_{ILE} is used.

7. CONCLUSION

This last decade, no major improvement has been made for parallel complexity of integer GCD computation and a performance of $O_\epsilon(n/\log n)$ time with $n^{1+\epsilon}$ processors on a CRCW PRAM seems to be a "limit" not easily surpassed.

Reduction methods are widely used as a tool in most integer GCD algorithms. We propose a new reduction called R_{ILE} where both theoretical and practical aspects are considered. R_{ILE} reduction presented in this paper may be used as a basic transformation for the best current GCD algorithms, as in [13, 16] for example. We have designed an integer GCD algorithm based on this reduction which matches the best existing algorithms.

Although its complexity remains the same, a compression method may be suggested [7, 3]. It is worth noting that, as far as R_{ILE} reduction is considered, all the decisions are made only from the first $O(m)$ leading bits of the current couple (u, v) at each step. Thus our algorithm adapts for such compression methods. We are currently investigating this idea with the hope of improving the performance of parallel integer GCD algorithms.

8. REFERENCES

[1] **A.V. Aho, J.E. Hopcroft and J.D. Ullman.** *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

[2] **R.P. Brent and H.T. Kung.** Systolic VLSI arrays for linear-time GCD computation, *in VLSI'83*, Anceau and Aas eds., 1983, 145-154.

[3] **B. Chor and O. Goldreich.** An improved parallel algorithm for integer GCD, *Algorithmica*, 5, 1990, 1-10.

[4] **G.H. Hardy and E.V. Wright.** *An Introduction To The Theory Of Numbers*, Oxford University Press., London, 1979.

[5] **T. Jebelean.** A Generalization of the Binary GCD Algorithm, *in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'93)*, 1993, 111-116.

[6] **T. Jebelean.** An Algorithm for Exact Division, *J. of Symbolic Computation*, 15, 1993, 169-180.

[7] **R. Kannan, G. Miller and L. Rudolph.** Sublinear Parallel Algorithm for Computing the Greatest Common Divisor of Two Integers, *SIAM J. on Computing*, Vol. 16, No 1, 1987, 7-16.

[8] **R. Karp, V. Rammachandran.** Parallel Algorithms for Shared-memory Machines, *In J. Van Leeuwen, Editor, Algorithms and Complexity*, Elsvier and MIT Press, 1990, Handbook of Theoretical Computer Science, Vol. A.

[9] **D.E. Knuth.** *The Art of Computer Programming*, Vol. 1-2, 2nd ed., Addison Wesley, 1981-1982.

[10] **D.H. Lehmer.** Euclid's algorithm for large numbers, *American Math. Monthly*, 45, 1938, 227-233.

[11] **A. Schönhage.** Schnelle Berechnung von Kettenbruchentwicklugen, *Acta Informatica*, 1, 1971, 139-144.

[12] **M.S. Sedjelmaci and C. Lavault.** Improvements on the accelerated integer GCD algorithm, *Information Processing Letters*, 61, 1997, 31-36.

[13] **J. Sorenson.** Two Fast GCD Algorithms, *J. of Algorithms*, 16, 1994, 110-144.

[14] **J. Sorenson.** An Analysis of Lehmer's Euclidean Algorithm, *in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, 1995, 254-258.

[15] **R.M. Stallman.** *Using and Porting GCC*, Free Software Foundation, 1991.

[16] **K. Weber.** Parallel implementation of the accelerated integer GCD algorithm, *J. of symbolic Computation (Special Issue on Parallel Symbolic Computation)*, 21, 1996, 457-466.