A Parallel Extended GCD Algorithm

Sidi Mohamed Sedjelmaci

LIPN CNRS UMR 7030,
Université Paris-Nord, 99 Avenue J.B. Clément
93430 Villetaneuse, France.
e-mail: sms@lipn.univ-paris13.fr

Abstract

A new parallel extended GCD algorithm is proposed. It matches the best existing parallel integer GCD algorithms of Sorenson and Chor and Goldreich, since it can be achieved in $O_\epsilon(n/\log n)$ time using at most $n^{1+\epsilon}$ processors on CRCW PRAM. Sorenson and Chor and Goldreich both use a modular approach which consider the least significant bits. By contrast, our algorithm only deals with the leading bits of the integers u and v, with $u \geq v$. This approach is more suitable for extended GCD algorithms since the coefficients of the extended version a and b, such that $au + bv = \gcd(u, v)$, are deeply linked with the order of magnitude of the rational v/u and its continuants. Consequently, the computation of such coefficients is much easier.

Keywords: Integer greatest common divisor (GCD); Parallel GCD algorithm; Extended GCD algorithm; Complexity analysis; Number theory.

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 implementa-

[1] A preliminary version of this paper was presented at ISSAC'01 conference, London Ontario, Canada, ACM Press, 2001, p. 303-308.

tion. In 1987, Kannan, Miller and Rudolph (KMR) [8] 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. In 1994, Sorenson's right- and left-shift k-ary algorithms [15] match Chor and Goldreich's performance.

The extended version of parallel GCD algorithms is also discussed in Sorenson and Chor and Goldreich papers ([15,3]). However, they only give the main ideas. Moreover, the parallel extended GCD algorithm of Sorenson still can be achieved in $O_\epsilon(n/\log n)$ time but it may require more than $n^{1+\epsilon}$ processors on CRCW PRAM (see [15], Section 7.2, p. 141, lines 1, 2 and 3).

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 [10] is tightly linked with the continued fractions [5,10] and is important for its multiple applications (cryptology, modular inversion, etc..). In [8], 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 $|a| \leq kv/u$, $|b| \leq 2k$, s.t. $0 \leq au - bv \leq u/k$ 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 [8] for more details). This paper focus on parallel extended GCD algorithms and 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. This reduction does not introduce any spurious factors to remove afterwards.
- 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 $\epsilon > 0$.
- We also design a new parallel extended GCD with this time bound, where the cofactors a an b such that $au + bv = \gcd(u, v)$, are easily computed.
- Our method can be generalized to all Lehmer-like algorithms and our algorithm may be modified to compute in parallel the continuants of rationals.

This paper is an improved version of a paper presented in ISSAC'01 ACM conference, where we have added the extended GCD versions of the sequential and parallel GCD algorithms.

In Section 2, we recall the basic reduction step used in Kannan, Miller and Rudolph's algorithm [8]. In Section 3, we define a new reduction which uses the same Lehmer idea [11]. 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 $1 \leq a \leq k$, for a given parameter $k = O(n)$. A sequential as well as a parallel algorithms are designed to compute this reduction and their correctness are discussed. Section 4 is devoted to the extended version of the reductions. A new parallel extended GCD algorithm is described in Section 5. Complexity analysis is discussed in Section 6. We conclude with some remarks in Section 7.

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 $u \geq v$. 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. Most of the time the *Common* sub-model is considered, where each processor must write the same value. However in order to allow the priority to the processor with the smallest index, one may choose the *Priority* sub-model [9].

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, called *spurious factor*, which can easily be removed afterwards [6,15,17]. Examples of such basic reductions are given in Table 1.

However, for extended GCD computations, we need matrix-vectors reductions defined by: $(u, v) \longmapsto (u', v') = M \times (u, v)$, where u' and v' are two integers and M is a 2×2 integer matrix. The matrix M preserves GCD's if $\det M = \pm 1$.

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:

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

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 $p > 2m + \rho + 1$.

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 [8,6,17]. 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 *ρ-Euclid*, defined in Section 5.1.

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 $u_2, v_2 < 2^{p-\lambda}$ 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 ([8], page 9):

Lemma 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 $0 \leq au - bv \leq u/k$.*

Name	Reduction	Property
Euclid	$u - qv$	$q = \lfloor u/v \rfloor$
binary	$(u - v)/2$	u and v odds
bmod	$\lvert u - xv \rvert / 2^\rho$	$x = (u/v) \pmod{2^\rho}$
Right-shift k-ary	$\lvert au + bv \rvert / k$	$au + bv \equiv 0 \pmod{k}$
Left-shift k-ary	$\lvert k^e v - bu \rvert$	$\lvert k^e v - bu \rvert = O(u/k)$

Table 1
Examples of Reductions.

Remark 2 *Since* $|au - bv| \leq u/k \leq v$, *then* $|au - bv| = (au) \bmod v$ *or* $v - au \bmod v$.

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 [8] 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 [11] suggested another way to compute the couple (a, b). Roughly speaking (see Knuth [10] 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 [16] 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) [16]:

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

The reduction we propose in this paper is also based on leading bits and continuants but, by contrast, our reduction satisfies a tighter relation $R(u,v) < 2v/k$ 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.

Lemma 3 *For all positive integers $u \geq v$, $k = 2^m$ and $p = \ell_2(v) > 2m+\rho+1$, 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 [5] (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

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

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

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

then $R < 2^{p-\lambda}v_1/2^m + 2^{p-\lambda}k' \leq v/2^m + 2^{p-\lambda+m+\rho}$. 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 $R < v/k$.

Definition 4 *Let (a, b) be one of the couples defined in Lemma 3. The R_{ILE} transformation is defined by*

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

Many such couples can be found and R_{ILE} depends on the couple (a, b) considered (see examples in Section 4.2), but for any one of them, R_{ILE} is a *reduction* satisfying $R_{ILE}(u, v) = |au - bv| < 2v/k$. We propose in the next Section an easy way to compute one of these couples (a, b) and the reduction R_{ILE}.

3.1 ILE Reductions

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

The Sequential algorithm for computing R_{ILE} : `Seq-ILE`

Input: $u \geq v > 0$, $k = 2^m$ s.t. $\rho = n - p + 1 < m$ and $p > 2m + \rho + 1$.
Output: $R_{ILE}(u, v) = |au + bv| < 2v/k$.

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

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 $R_{ILE} = |au + bv|$;

Return R_{ILE}.

All the triplets (r, a, b) computed in EEA satisfy $r = |a|u_1 \bmod v_1$ or $r = v_1 - (|a|u_1 \bmod v_1)$ and the previous algorithm is easily parallelized as follows.

The Parallel algorithm for computing R_{ILE} : `Par-ILE`

Input: $u \geq v > 0$, $k = 2^m$ s.t. $\rho = n - p + 1 < m$ and $p > 2m + \rho + 1$.
Output: $R_{ILE}(u, v) = |au + bv| < 2v/k$.

Step 1
Compute p, λ, u_1, v_1 as in `Seq-ILE`.

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** $(a, b) := (i, -q_i)$;
if $v_1 - r_i < v_1/k$ **then** $(a, b) := (-i, q_i + 1)$;
End Do

Step 3 Compute in parallel $R_{ILE} = |au + bv|$

Return R_{ILE}.

If many processors are in write concurrency in Step 2 then we can use the *Priority* sub-model of CRCW-PRAM. With this sub-model, we choose the reduction R_{ILE} given by the smallest index i s.t.: $1 \leq i \leq k$, $r_i < v_1/k$ or $v_1 - r_i < v_1/k$. Actually, the *Priority* is a strong sub-model of CRCW-PRAM. The most used sub-model is the *Common* sub-model of CRCW-PRAM where all the processors must write the same value. We show below how to modify `Par-ILE` algorithm in order to compute the GCD in the same parallel performance on the Common CRCW-PRAM sub-model. We only have to change the Step 2 as follows:

Modified Step 2
For $i = 1, 2, .., 2^m$ **Do in parallel** $A[i] := v_1$; $B[i] =: 1$; **End Do**
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** $(A[i], B[i]) := (i, -q_i)$;
if $v_1 - r_i < v_1/k$ **then** $(A[i], B[i]) := (-i, q_i + 1)$;
End Do
Compute in parallel $j := MIN\{ \mid A[i] \mid , i = 1, 2, \ldots, 2^m\}$;
$(a, b) := (A[j], B[j])$;

Note that the minimum finding MIN of the array A (or B) of $O(2^m)$ integers of $O(m)$ bits $A[i], i = 1, 2, \ldots 2^m$, can be achieved in parallel in $O(1)$ time with $O(m \log\log m 2^{2m})$ processors, on the Common CRCW-PRAM. As a matter of fact, we need $O(2^{2m})$ processors to compute, in parallel, a $2^m \times 2^m$ matrix. Each term of the matrix compare in pair, two $O(m)$ bits integers $A[i]$ and $A[j]$, in $O(1)$ time with $O(m \log\log m)$ processors. This will not affect our general

parallel time bound since, if we choose $m = 1/2\ \epsilon \log n$, then the number of processors needed is $O(n^{\epsilon} \log n\ \log\log\log n) = O(n)$ (see Section 6).

Remark 5 *Let $m = O(\log n)$. Even when u and v are very large numbers in size (up to $65,536$-bits, $n, p \leq 2^{16}$) the computations in Step 1 and 2 can be performed in constant time with a single precision since $\log n \leq 16$ (see Section 6).*

3.2 Example of R_{ILE}

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	+6.10

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.

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

3.3 Correctness of `Seq-ILE` *and* `Par-ILE`

Proposition 6 *The sequential algorithm* `Seq-ILE` *ends with* r *and* R_{ILE} *satisfying*

$$\begin{aligned} &i) \quad r = |au_1 + bv_1| < v_1/k \\ &ii) \quad R_{ILE} = |au + bv| < 2v/k. \end{aligned}$$

Proof Consider EEA (Extended Euclidean Algorithm) applied to the pair (u_1, v_1). Let (a_s, b_s, r_s) and $(a_{s+1}, b_{s+1}, r_{s+1})$ be the two consecutive triplets such that $|a_s| \leq k = 2^m < |a_{s+1}|$. Then $r = r_s = a_s u_1 + b_s v_1$ and $r_{s+1} = a_{s+1}u_1 + b_{s+1}v_1$. Solving this system, we obtain $v_1 = |a_s r_{s+1} - a_{s+1} r_s| = |a_{s+1}|r_s + |a_s|r_{s+1}$ since $|a_s b_{s+1} - a_{s+1} b_s| = 1$ and $a_s a_{s+1} < 0$. Hence $r = r_s = (v_1 - r_{s+1}|a_s|)/|a_{s+1}| < v_1/|a_{s+1}| < v_1/k$ and $i)$ is proved. Moreover,

$$R_{ILE} = |a_s u + b_s v| \leq |a_s u_1 + b_s v_1| 2^{p-\lambda} + |a_s u_2 + b_s v_2| < 2^{p-\lambda} v_1/k + 2^{p-\lambda}|b_s|$$

but $2^{p-\lambda}v_1/k < v/k$ and $|b_s| = |(r_s - a_s u_1)/v_1| \leq r_s/v_1 + |a_s|(u_1/v_1) < 1/k + k(u_1/v_1)$. But since $u_1/v_1 < 2^\rho$ then $|b_s| < 1/k + 2^{m+\rho}$ and $|b_s| \leq 2^{m+\rho}$, so

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

From $\lambda = 2m + \rho + 1$, we obtain $R < 2v/k$ and $ii)$ is proved.

□

Proposition 7 *The output* R_{ILE} *of the parallel algorithm* `Par-ILE` *satisfies* $R_{ILE}(u, v) < 2v/k$.

Proof In Step 2 (or Modified Step 2), we select the value r with the smallest index i satisfying $r = r_i < v_1/k$ or $r = v_1 - r_i < v_1/k$ with $1 \leq i \leq k$. Then the output $R_{ILE} = |au + bv|$ of step 3 satisfies $R_{ILE}(u, v) \leq |au_1 + bv_1|2^{p-\lambda} + |au_2 + bv_2| < 2^{p-\lambda}v_1/k + 2^{p-\lambda}|b|$. But, as proved previously in Proposition 6, the inequality $|au_1 + bv_1| < v_1/k$ gives $|b| \leq 2^{m+\rho}$ and $R_{ILE}(u, v) < 2v/k$.

□

4 Extended Reductions

Extended GCD algorithms compute the GCD of two integers u and v and also supplies integers a and b such that $au + bv = \gcd(u, v) = d$. They also play a key role in many applications in computer algebra systems [4]. The fastest

presently known parallel GCD algorithms are those of Chor and Goldreich [3], Sorenson [15] and Sedjelmaci [14]. They all compute parallel integer GCD in $O_\epsilon(n/\log n)$ time using at most $n^{1+\epsilon}$ processors on CRCW PRAM, for any constant $\epsilon > 0$. However, the algorithms described in [3] and [15] follow a Least Significant digit First approach (`LSF`). Although possible, this approach is not well suited for computing extended version of GCD since these algorithms end with three integers α, β and p satisfying the modular relation

$$\alpha u + \beta v = 2^p \gcd(u, v).$$

So some extra computations are necessary to recover the Bezout cofactors (a, b) satisfying

$$au + bv = \gcd(u, v) = d \quad \text{with } |a| \leq v/2d,\ |b| \leq u/2d.$$

By contrast, the parallel GCD algorithm described in [14] follows a Most Significant digit First approach (`MSF`) which is more suitable to compute extended GCD versions. In this Section, we show how to slightly modify the previous algorithms `Seq-ILE` and `Par-ILE` to easily derive sequential and parallel extended `ILE` reductions.

First, we slightly modify the algorithm `Seq-ILE` as follows.

The Sequential Extended R_{ILE} algorithm : `Seq-Ext-ILE`

Input: $u \geq v > 0$, $k = 2^m$ s.t. $\rho = n - p + 1 < m$ and $p > 2m + \rho + 1$.

Output: $M = \begin{pmatrix} c & d \\ a & b \end{pmatrix}$ and $\begin{pmatrix} R \\ R_{ILE} \end{pmatrix} = M \begin{pmatrix} u \\ v \end{pmatrix}$

with $|cb - ad| = 1$, $GCD(u, v) = GCD(R, R_{ILE})$, $R_{ILE} < 2v/k$;

Step 1

$n := l_2(u)$; $\quad p := l_2(v)$;

$\lambda := 2m + \rho + 1$;

$u_1 := \lfloor u/2^{p-\lambda} \rfloor$; $\ v_1 := \lfloor v/2^{p-\lambda} \rfloor$;

Step 2 Run EEA with (u_1, v_1) and compute successive triplets (r_s, a_s, b_s), until $|a_s| \leq 2^m < |a_{s+1}|$.

Step 3 Compute $R = |a_{s-1}u + b_{s-1}v|$; $\quad R_{ILE} = |a_s u + b_s v|$;

Return $M = \begin{pmatrix} a_{s-1} & b_{s-1} \\ a_s & b_s \end{pmatrix}$ and $\begin{pmatrix} R \\ R_{ILE} \end{pmatrix}$.

It is worth to note that algorithm `Seq-ILE` introduces spurious factors (see Section 2.1), while `Seq-Ext-ILE` does not and the transformation $(u, v) \leftarrow (R, R_{ILE})$ preserves the GCD property (see Section 4.1).

The parallel algorithm described in `Par-ILE` does not return the smallest reduction R_{ILE} as the example of Section 4.2 shows. Moreover, we need to store a matrix in order to compute the Bezout cofactors. Hence some modifications are needed to design a parallel extended GCD. We suggest the following.

The Parallel Extended R_{ILE} algorithm : `Par-Ext-ILE`

Input: $u \geq v > 0$, $k = 2^m$ s.t. $\rho = n - p + 1 < m$ and $p > 2m + \rho + 1$.

Output: $M = \begin{pmatrix} c & d \\ a & b \end{pmatrix}$ and $\begin{pmatrix} R_1 \\ R_2 \end{pmatrix}$ with $1 \leq R_1 \leq v$, $0 \leq R_2 < 2v/k$,
$|bc - ad| = 1$ and $\mathrm{GCD}(R_1, R_2) = \mathrm{GCD}(u, v)$.

Step 1 Compute p, λ, u_1 and v_1 as in Figure 1.

Step 2 /* Computation of (a, b) and R_2 */
$X := v_1$; $Y := v_1$;
For $i = 1, 2, .., 2^m$ **Do in parallel**
$q_i := \lfloor iu_1/v_1 \rfloor$; $r_i := iu_1 - q_i v_1$; $s_i := v_1 - r_i$;
If $(r_i < v_1/k)$ $X := r_i$; $(a, b) := (i, q_i)$;
If $(s_i < v_1/k)$ $Y := s_i$; $(c, d) := (-i, q_i + 1)$;
EndFor
If $(X > Y)$ $(a, b) := (c, d)$;
$R_2 := au + bv$;
if $R_2 < 0$ **then** $(a, b) := (-a, -b)$; $R_2 := -R_2$;

Step 3 /* Computation of (c, d) and R_1 */
$(c, d) :=$`Bezout`(a, b); $R_1 := cu + dv$;
if $R_1 < 0$ **then** $(c, d) := (-c, -d)$; $R_1 := -R_1$;
if $ac < 0$ **then** $t := 1$; **else** $t := -1$;
$(c', d') := (c + ta, d + tb)$; $R_1' := c'u + d'v$;
if $R_1' < 0$ **then** $(c', d') := (-c', -d')$; $R_1' := -R_1'$;
if $R_1' < R_1$ **then** $R_1 := R_1'$; $(c, d) := (c', d')$;

Return $M = \begin{pmatrix} c & d \\ a & b \end{pmatrix}$ and $\begin{pmatrix} R_1 \\ R_2 \end{pmatrix}$.

If many processors are in write concurrency in Step 2, then we may use the *Priority* sub-model of CRCW-PRAM, which select the smallest index i s.t.: $1 \leq i \leq k$, $r_i < v_1/k$ or $s_i = v_1 - r_i < v_1/k$. However, similarly, as we did for `Par-ILE` in Section 3.1, we can easily modify the parallel loop of **Step** 2, to obtain the same parallel performance with the *Common* sub-model of CRCW-PRAM (we also need to store the respective values of r_i and s_i in two arrays).

Given two integers (a, b), such that $GCD(a, b) = 1$, the function `Bezout`(a, b) returns the unique pair (c, d) such that $|c| \leq |a|/2$, and $c|b| + d|a| = 1$. It can be computed by EEA or in parallel as follows:

The `Bezout` Algorithm :

Input: A pair of integers (a, b), with $|b| \geq |a|$ and GCD$(a, b) = 1$.
Output: A pair of integers (c, d) s.t.: $c|b| + d|a| = 1$ with $|c| \leq |a|/2$.

For $i = 1, 2, \ldots, \lfloor |a|/2 \rfloor$ **Do in parallel**
  $\alpha_i := i|b| \bmod |a|$; $\beta_i := |a| - \alpha_i$;
  if $\alpha_i := 1$ **Return** $(i, -\lfloor \frac{i|b|}{|a|} \rfloor)$;
  if $\beta_i := 1$ **Return** $(-i, \lfloor \frac{i|b|}{|a|} \rfloor + 1)$;
EndFor

This pair (c, d) can be computed in $O(1)$ parallel time (see Section 6) with $|a|/2 = O(k)$ processors in a weaker PRAM model CREW (Concurrent Read Exclusive Write), since only one index i, $1 \leq i \leq |a|/2$, satisfies $\alpha_i = 1$ or $\beta_i = 1$, so no write concurrency could happen. Note that $|cb - ad| = 1$ since $ab < 0$ and $c|b| + d|a| = 1$.

4.1 Correctness of `Seq-Ext-ILE` *and* `Par-Ext-ILE`

Lemma 8 *If* $M = \begin{pmatrix} c & d \\ a & b \end{pmatrix}$ *with* $|cb - ad| = 1$ *and* $\begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = M \begin{pmatrix} u \\ v \end{pmatrix}$, *then* $GCD(R_1, R_2) = GCD(u, v)$.

Proof Let $\alpha = GCD(u,v)$ and $\beta = GCD(R_1, R_2)$ then $\begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = M \begin{pmatrix} u \\ v \end{pmatrix} = \alpha M \begin{pmatrix} u/\alpha \\ v/\alpha \end{pmatrix}$ and $\beta \geq \alpha$. Moreover, since $|cb - ad| = 1$, M^{-1} exists and has integers entries so $\begin{pmatrix} u \\ v \end{pmatrix} = M^{-1} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = \beta M^{-1} \begin{pmatrix} R_1/\beta \\ R_2/\beta \end{pmatrix}$, hence $\alpha \geq \beta$ and the result $\alpha = \beta$.

□

Lemma 9 *Let $u > v \geq 1$ be two positive integers. Let (a,b) be the pair given by EEA, satisfying $au + bv = \gcd(u,v) = d$ and let (α, β) be any other pair of integers such that $\alpha u + \beta v = \gcd(u,v) = d$. Then:*

(1) *There exists an integer A such that*
$\alpha = a + A(v/d)$ *and* $\beta = b - A(u/d)$.

(2) *(a,b) is the unique pair such that: $|a| \leq v/2d$ and $|b| \leq u/2d$.*

(3) $|a| = \min\ \{\ |\alpha| \bmod (v/d),\ v/d - |\alpha| \bmod (v/d)\ \}$.

Proof If there is only one iteration in EEA then $u = qv$ with $q \geq 2$ because $u > v$. So we take $(a,b) = (0,1)$. Otherwise we assume that there is at least two iterations in EEA and the last quotient must satisfy $q \geq 2$. Let (a', b') and (a,b) be the two last cofactors of EEA and q be the last quotient then:

$$|a'| + q|a| = v/d \geq q|a|,$$
$$|b'| + q|b| = u/d \geq q|b|.$$

Hence $|a| \leq v/qd \leq v/2d$ and $|b| \leq u/qd \leq u/2d$. Moreover, if (α, β) is another pair of integers such that $\alpha u + \beta v = d$, then

$$(\alpha - a)u/d + (\beta - b)v/d = 0, \text{ with } \gcd(u/d, v/d) = 1.$$

Thus v/d divides $(\alpha - a)u/d$ but $\gcd(u/d, v/d) = 1$, so v/d divides $\alpha - a$ and there exists an integer A such that $(\alpha - a) = A(v/d)$ and$(\beta - b) = -A(u/d)$, with $|a| \leq \frac{v}{2d}$, $|b| \leq \frac{u}{2d}$ and (1) is proved. Let us prove the unicity of the pair (a,b) satisfying:

$$au + bv = \gcd(u,v) = d \text{ with } |a| \leq \tfrac{v}{2d}.$$

In fact, if we suppose that there are two such pair (a,b) and (α, β) then $|a| \leq \frac{v}{2d}$, $|\alpha| \leq \frac{v}{2d}$ so $|\alpha - a| \leq \frac{v}{d}$. Hence $A = 0$ or $A = 1$. If $A = 1$ then $|a| = \frac{v}{2d}$ and $au + bv = \epsilon\frac{vu}{2d} + bv = d$, with $\epsilon = \pm 1$. Then $v(\frac{\epsilon u}{2d} + b) = d$ and $v = d$. So $b = -\epsilon\frac{u}{2d}$ and $a = \frac{v}{2d} = 1/2$, which is impossible. Hence $A = 0$ and (2) is proved. The equality (3) is proved as follows:

Case 1: $\alpha = a$. Obvious, since $|\alpha| \leq \frac{v}{2d}$, then $|\alpha| \bmod v/d = |\alpha| = |a|$.

Case 2: $\alpha > a$. We have $\alpha = a + |A|(v/d)$. If $a \geq 0$ then $a = \alpha \bmod v/d$. If $a < 0$ then $\alpha = |A|(v/d) - |a| > 0$ because $|A| \geq 1$, since $\alpha \neq a$. Hence $\alpha = (|A|-1)(v/d)+(v/d)-|a|$, with $\frac{v}{2d} \leq (v/d)-|a| < (v/d)$ then $(v/d)-|a| = \alpha \bmod v/d$ and $|a| = (v/d) - \alpha \bmod v/d \leq \frac{v}{2d}$.

Case 3: $\alpha < a$. We have $\alpha = a - |A|(v/d)$. If $a < 0$, then $|\alpha| = |a| + |A|(v/d)$ and $|\alpha| \bmod v/d = |a|$. If $a > 0$ then $\alpha < 0$ because $|a| \leq \frac{v}{2d}$. Thus $|\alpha| =$

$|A|(v/d) - |a| = (|A|-1)(v/d) + (v/d) - |a|$, with $\frac{v}{2d} \leq (v/d) - |a| < (v/d)$, so $(v/d) - |a| = \alpha \bmod v/d \geq \frac{v}{2d}$, hence $|a| = \frac{v}{2d} - \alpha \bmod v/d \leq \frac{v}{2d}$ and (3) is proved.

□

Definition 10 *Let $u > v \geq 1$ be two non-negative integers. The unique pair of integers such that $au + bv = gcd(u, v)$ with $|a| \leq v/2d$ and $|b| \leq u/2d$ is called the* `Bezout` *cofactors of (u, v).*

Lemma 11 *Let i be the smallest index in Step 2 of* `Par-Ext-ILE` *algorithm such that $1 \leq i \leq k$ and $iu_1 - q_i v_1 < v_1/k$ or $(q_i + 1)v_1 - iu_1 < v_1/k$, then $GCD(i, q_i) = 1$ or $GCD(i, q_i + 1) = 1$ respectively.*

Proof Let $\mathrm{GCD}(i, q_i) = \delta$. If we assume $\delta \geq 2$, then $i = \delta i'$ and $q_i = \delta q_i'$ for some non-negative integers i' and q_i'. Hence $iu_1 - q_i v_1 = \delta(i'u_1 - q_i' v_1) < v_1/k$. So $i'u_1 - q_i' v_1 < v_1/k\delta < v_1/k$ which is impossible since i is the smallest index such that $1 \leq i \leq k$ and $iu_1 - q_i v_1 < v_1/k$. The other case is similar.

□

Lemma 12 *Let (a, b) and (c, d) be the two pair of integers obtained in* `Par-Ext-ILE` *such that $GCD(a, b) = 1$ and $(c, d) =$* `Bezout`*(a, b), then*

1) $|ad - bc| = 1$ *with* $|c| \leq |a|/2$
2) $|cu_1 + dv_1| \leq v_1$, *with equality if* $|a| = 1$.
3) $|cu + dv| \leq v$, *with equality if* $|a| = 1$.

Proof Since $GCD(a, b) = 1$, then EEA applied to $(|b|, |a|)$ returns a pair (c, d) such that $c|b| + d|a| = 1$ with $|c| \leq |a|/2$ (Lemma 9). Moreover, since only one index i, $1 \leq i \leq |a|/2$, satisfies $\alpha_i = 1$ or $\beta_i = 1$, then (i, q_i) or $(-i, q_i + 1)$ coincides with (c, d), hence 1). If $|a| = 1$ then $|c| \leq |a|/2 = 1/2$, so $c = 0$ and $|d| = 1$. Hence $|cu_1 + dv_1| = v_1$. If we assume $|a| \geq 2$, then we obtain $|a(cu_1 + dv_1)| = |c(au_1 + bv_1) \pm v_1| \leq |c||au_1 + bv_1| + v_1$ since $ad = bc \pm 1$. From Proposition 7, we have $|au_1 + bv_1| \leq 2v_1/k$ and since $k \geq 2^m \geq 4$:
$|cu_1 + dv_1| \leq (\frac{2|c|}{|a|k} + \frac{1}{|a|})\ v_1 \leq (\frac{1}{k} + \frac{1}{2})v_1 < v_1$.
Moreover $|cu + dv| \leq |cu_1 + dv_1|2^{p-\lambda} + |cu_2 + dv_2|$ and similarly to the proof of Proposition 6, we obtain $|cu + dv| \leq (\frac{1}{2k} + \frac{1}{|a|})v_1 2^{p-\lambda} + v/k \leq (\frac{1}{2k} + \frac{1}{|a|} + \frac{1}{k})v < v$ since $k \geq 4$ and $|a| \geq 2$. The case $|a| = 1$ gives $c = 0$, $|d| = 1$ and $|cu + dv| = v$.

□

Proposition 13 *The outputs (R, R_{ILE}) of* `Seq-Ext-ILE` *satisfy $GCD(R, R_{ILE}) = GCD(u, v)$ and $R_{ILE} < 2v/k$.*

Proof The entries of matrix M are obtained by EEA, so they satisfy the relation $|cb-ad|=1$. The first property derive from Lemma 8 and the second one from the proof of Proposition 6.

□

Proposition 14 *The outputs M and (R_1, R_2) of* `Par-Ext-ILE` *satisfy* $1 \leq R_1 \leq v,\ \ 1 \leq R_2 < 2v/k,\ |bc-ad|=1$ *and* $GCD(R_1, R_2) = GCD(u, v)$.

Proof Straightforward from Lemma 11, 12 and 8 and the observation that the transformations $(a, b) := (-a, -b)$, $(c, d) := (-c, -d)$ and $(c, d) := (a+c, b+d)$ preserve the relation $|bc-ad|=1$ as well as $GCD(R_1, R_2)$.

□

4.2 Example of Par-Ext-ILE

If $(u_1, v_1) = (1137, 1001)$ and $k = 8$, we obtain in turn

i	r_i	$s_i = v_1 - r_i$
1	136	865
2	272	729
3	408	593
4	544	457
5	680	321
6	816	185
7	952	49
8	87	914

and $X = 87$, $Y = 49$, then $(a, b) = (-7, 8)$ and $(c, d) = (1, -1)$. The parallel algorithm `Par-Ext-ILE` returns $(R_1, R_2) = (136, 49)$ and the matrix $M = \begin{pmatrix} 1 & -1 \\ -7 & 8 \end{pmatrix}$. Note that $v_1/k = 1001/8 = 125.125$, so there are two possible values for R_{ILE}, namely 49 and 87. However, it may be the case where the smallest index do not give the smallest reduction with `Par-Ext-ILE`. Consider for example $(u_1, v_1) = (747, 403)$ with $k = 8$.

5 Par-Ext-GCD: The Parallel Extended GCD Algorithm

Given integers $u \geq v > k > 0$ 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 $v \geq 8k^2$ (R_{ILE} is undefined when $v < 8k^2$). In case the difference of bit size between u and v is large, i.e.: $\rho > m$, we choose another Euclid-like reduction called the ρ-`Euclid`. We find it easier to take m as a "threshold" (the borderline choice between R_{ILE} and the ρ-`Euclid` reductions); likewise, we might choose a varying threshold, depending upon v and experimental data [6,17].

5.1 The ρ-Euclid Reduction

Proposition 15 *Let $u \geq v > 0$ and $q = \lfloor u/v \rfloor$. We consider a parameter λ s.t. $0 \leq \lambda \leq p$. We define u_1 and v_1 by: $u_1 = \lfloor u/2^{p-\lambda} \rfloor$ and $v_1 = \lfloor v/2^{p-\lambda} \rfloor$. We let $q' = \lfloor u_1/v_1 \rfloor$, then:*

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

Proof $v_2 = v \bmod 2^{p-\lambda}$, so that

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

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

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

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

□

Remark 16 *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 Kannan, Miller and Rudolph in [8] (p. 9). They took $\lambda = n - p$ and obtain $|q'_i - q_i| \leq 3$.*

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

Definition 17 *With the notation described in* `Seq-ILE`*, if u and v are such that $2p \geq n + 2$ and $\lambda = n - p + 2$, the ρ-Euclid reduction is defined by*

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

Remark 18 *A new integer GCD algorithm similar to Euclid's one can be designed with the ρ-Euclid reduction R_ρ and this algorithm avoids many long divisions. Moreover, if we store the matrices* $M = \begin{pmatrix} 0 & 1 \\ 1 & -q' \end{pmatrix}$ *at each step, then we obtain another extended GCD algorithm, i.e.: we find a pair of integers (a, b) s.t.: $au + bv = \gcd(u, v)$, but this pair may differ from that of EEA. Note that a similar algorithm is proposed in [10] (p.*376*, exe.* 30*).*

Example: Let $u = 26,977$ and $v = 8,737$, we have

$$\begin{aligned} \mathbf{1101}\ 00101100001 &= 26,977 \\ \mathbf{100}\ 01000100001 &= 8,737. \end{aligned}$$

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 $R_\rho = |u - 3v| = 766$.

5.2 High Level Description

The following algorithm computes the parallel extended GCD of two integers u and v.

The Parallel Extended GCD Algorithm `Par-Ext-GCD`.

Step 1: Perform reductions until $v < 8k^2$: if $\rho < m$, then perform `Par-Ext-ILE` reduction; else, perform the ρ-`Euclid` reduction. Store each matrix M_i in an array A of $O(n)$-bits in size, i.e.: $A[i] := M_i$;

Step 2: Compute $d = \gcd(u, v)$ and the matrix of cofactors N with EEA, where (u, v) is the last pair satisfying $v < 8k^2$.

Step 3: Compute $M = N \times \prod_i M_i$.

Step 4: $(a, b) :=$ `Recover`(α, β, d).

Step 5: Return d and (a, b).

Note that there are no spurious factors to remove at the end of the GCD algorithm ([15,16,6]) because all the matrices have a determinant equal to ± 1. In case the pair (α, β) of the second line in the output matrix M is not the Bezout cofactors, we can easily recover them thanks to Lemma 9, as follows:

The Bezout recovering cofactors function `Recover`:

$(a, b) := (\alpha, \beta)$;
If $|\alpha| > v/2d$
$a := |\alpha| \bmod v/d$;
If $a > \frac{v}{2d}$ **then** $a := v/d - a$;
If $au \bmod v \neq d$ **then** $a := -a$;
$A := (\alpha - a)/(v/d)$; $b := \beta + A(u/d)$;
Return (a, b).

This determination of the Bezout cofactors is correct since $(au) \bmod v = d$ or $(au) \bmod v = v - d$ and, a parallel computation will require $O(\log n)$ time with $O(n \log n \log\log n)$ number of processors.

6 Complexity Analysis

Recall that parallel multiplication of two n bits integers can be achieved in $O(\log n)$ time with $O(n \log n \log\log n)$ processors. The parallel addition/subtraction of two n bits integers can be achieved in $O(1)$ time with $O(n \log\log n)$ processors. The complexity analysis of `Par-Ext-GCD` is given below.

Lemma 19 *The computation of $M = \prod_i M_i$ can be achieved in parallel in $O(\log^2 n)$ time with $O(n \log n \log\log n)$ processors.*

Proof Multiplying in parallel each pair of consecutive matrix $M_{2i+1} \times M_{2i+2}$, $i = 1, 2, \ldots, I$ for some index I, requires $O(\log n)$ time with a total of $O(n \log n \log\log n)$ processors. Repeating this process to each new pair of consecutive matrix requires at most $O(\log n)$ steps of a binary tree computation of depth $O(\log n)$, thus the total parallel time is $O(\log^2 n)$ with $O(n \log n \log\log n)$ processors.

□

Theorem 20 *The algorithm described in* `Par-Ext-GCD` *can be achieved in parallel in $O_\epsilon(n/\log n)$ time with $O(n^{1+\epsilon})$ processors on a CRCW PRAM.*

Proof 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. 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 (either by Step 2 or Modified Step 2). Then $R_{ILE} = |iu - q_i v|$ is computed in parallel. 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 [15] or [3] for more details).

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 [15] or [3] for more details).

The computation of $R_{ILE} = |iu - q_i v|$ requires (see `Par-ILE`) 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: $(\rho < m)$

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

The parallel computation of $M = \prod_i M_i$ can be achieved in $O(\log^2 n)$ time with $O(n \log n \log\log n)$ processors (Lemma 19). R_{ILE} reduces the size of the smallest input v by at least $m-1$ bits. Hence the `Par-Ext-GCD` algorithm runs in $O(n/m)$ iterations. For $m = 1/2\ \epsilon \log n$, $(\epsilon > 0)$ the parallel `Par-Ext-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. Recovering the Bezout cofactors costs $O(\log n)$ in time with $O(n \log n \log\log n) = O(n^{1+\epsilon})$ processors.

□

7 Conclusion

Since Chor and Goldreich's paper [3] in 1990, 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.

Based on a new reduction step, new sequential and parallel GCD algorithms are designed. The parallel algorithm matches the best known GCD algorithms. This paper focuses on the parallel extended version of the GCD. We designed a new parallel extended GCD with the same parallel performance.

The cofactors a an b such that $au + bv = \gcd(u, v)$, are easily computed and do not introduce any spurious factors. This algorithm is described in detail with correctness proofs and complexity analysis.

Our method can be generalized to all Lehmer-like algorithms and our algorithm may be modified to compute in parallel the continuants of rationals.

Although its complexity remains the same, a compression method may be suggested [8,3]. 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 and we are currently investigating this idea with the hope of improving the performance of parallel integer GCD algorithms.

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: Anceau and Aas eds., *Proceedings of VLSI'83* (1983) 145–154.

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

[4] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra.* (Cambridge University Press, 1st ed., 1999).

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

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

[7] T. Jebelean, An Algorithm for Exact Division, *J. of Symbolic Computation.* **15** (1993) 169–180.

[8] 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, 1**, (1987) 7–16.

[9] 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**.

[10] D.E. Knuth, *The Art of Computer Programming* (Addison Wesley, 3rd ed., 1998), **Vol. 2**.

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

[12] A.J. Manezes, P.C.van Oorschot, S.,A., Vanstone, *Handbook of Applied Cryptography* (CRC Press, 2nd ed., 1997), **Vol. 1-2**.

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

[14] M.S. Sedjelmaci, On a Parallel Lehmer-Euclid GCD Algorithm, *Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'2001* (2001) 303–308.

[15] J. Sorenson, Two Fast GCD Algorithms, *J. of Algorithms* **16** (1994) 110–144.

[16] J. Sorenson, An Analysis of Lehmer's Euclidean GCD Algorithm, *Proceedings of the International Symposium on Symbolic and Algebraic Computation ISSAC'95* (1995) 254–258.

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