The Mixed Binary Euclid Algorithm

Sidi Mohamed SEDJELMACI

LIPN CNRS UMR 7030,
Université Paris-Nord
Av. J.-B. Clment, 93430 Villetaneuse, France.
E-mail: sms@lipn.univ-paris13.fr

Abstract We present a new GCD algorithm for two integers that combines both the Euclidean and the binary gcd approaches. We give its worst case time analysis and prove that its bit-time complexity is still $O(n^2)$ for two n-bit integers. However, our preliminar experiments show that it is very fast for small integers. A parallel version of this algorithm matches the best presently known time complexity, namely $O(\frac{n}{\log n})$ time with $n^{1+\epsilon}$, for any constant $\epsilon > 0$.
Keywords: Integer greatest common divisor (GCD); Complexity analysis; Number theory.

1 Introduction

Given two integers a and b, the greatest commun divisor (GCD) of a and b, denoted $\gcd(a, b)$, is the largest integer which divides both a and b. Applications for GCD algorithms include computer arithmetic, integer factoring, cryptology and symbolic computation.

Most of GCD algorithms follow the same idea of reducing efficiently u and v to u' and v', so that $GCD(u, v) = GCD(u', v')$ [7]. These transformations are applied several times till $GCD(u', v')$ can be computed directly from u' and v'. Such transformations, also called *reductions*, are studied in a general framework in [7]. One can divides these transformations into two classes depending on whether they deal with the most significant digits first (the `MSF` approach) or the last significant digits first (the `LSF` approach). For example the Euclidean algorithm is the first `MSF` algorithm while the binary algorithm of Stein [4] is an `LSF` one. A classification of some GCD algorithms is given in Table 1.

For very large integers, the fastest GCD algorithms [2, 6, 10, 11] are all based on half-gcd procedure and computes the GCD in $O(n \log^2 n \log \log n)$ time. However, all these fast algorithms fall down to more basic algorithms at some point of their recursion, so, other algorithms are needed to medium and small size integers. For example, although the algorithm of T. Jebelean [1] and K. Weber [12] are quadratic in time, they have proven to be highly effecient for large and medium size integers. In this paper, we are interested in small size integers. Usually, the euclidean and the binary gcd works very well in practice for this range of integers. We present a new algorithm that combines both the euclidean

MSF	LSF
Euclid and like	binary
$\rho - Euclid$	bmod
Lehmer-Euclid	Plus-minus
ILE	Jebelean-Weber
Schönhage	Chor & Goldreich

Table 1: MSF-LSF Classification

and the binary gcd in a same algorithm, taking the most of them. We give its worst case time complexity and we suggest a parallel version that matches the best presently known time complexity, namely $O(\frac{n}{\log n})$ time with $n^{1+\epsilon}$, $\epsilon > 0$ (see [3, 9, 8]). In the next Section 2, we describe a new sequential algorithm and study its worst case. Section 3, we suggest a parallel version and study its parallel complexity. The paper ends in Section 4 with some concluding remarks.

2 The Sequentiel Algorithm

2.1 Motivation

Let us start with an illustrative example. Let $(u, v) = (5437, 2149)$. After one euclidean step, we obtain the quotient $q = 2$ and the remainder $r = 1139$. On the other hand, we observe that, in the same time, $u - v = 3288 = 2^3 \times 411$ and the binary algorithm gives $\frac{u-v}{8} = 411$ which is smaller and easy to compute (right-shift). The reverse is also true, Euclid algorithm step may perform much more than the binary one. So the idea is to take the most of both euclidean and binary steps and combine them in a same algorithm. Note that a similar idea was suggested by Harris with a different reduction step.

Lemma 2.1 *Let u and v be two integers such that v odd, $u \geq v \geq 1$ and let $r = u \pmod{v}$. Then we have*

i) $\min \{ v - r,\ r,\ \frac{r}{2} \text{ or } \frac{v-r}{2} \} \leq \frac{v}{3}$

ii) $\gcd(r, \frac{v-r}{2}) = \gcd(u, v)$, *if r is odd*

$\gcd(\frac{r}{2}, v - r) = \gcd(u, v)$, *if r is even.*

Proof: Note that either r or $v - r$ is even, so that either $\frac{r}{2}$ or $\frac{v-r}{2}$ is an integer. Recall the basic gcd property:

$\forall \lambda \geq 1$, $\gcd(u, v) = \gcd(v, u - \lambda v)$. Two cases arise:

Case 1: r is even then $v - r$ is odd. If $r \leq \frac{2v}{3}$ then $\frac{r}{2} \leq \frac{v}{3}$, otherwise $r > 2v/3$ and $v - r < \frac{v}{3}$.

Moreover, $\gcd(\frac{r}{2}, v - r) = \gcd(r, v - r) = \gcd(v, r) = \gcd(u, v)$.

Case 2: r is odd then $v - r$ is even. If $v - r \leq \frac{2v}{3}$ then $\frac{v-r}{2} \leq \frac{v}{3}$, otherwise, $v - r > 2v/3$

and

$r < \frac{v}{3}$. On the other hand, $gcd(\frac{v-r}{2}, r) = \gcd(r, v-r) = \gcd(v, r) = \gcd(u, v)$. $\square$

We derive, from Lemma 2.1, the following algorithm.
Algorithm `MBE:` `Mixed Binary Euclid`

```
Input: u>=v>=1, with v odd
Output: gcd(u,v)
Begin
      while (v>1)
           r=u mod v; s=v-r;
           while (r>0 and r mod 2 =0 ) r=r/2;
           while (s>0 and s mod 2 =0 ) s=s/2;
           if (s<r) {u=r; v=s; }
           else     {u=s; v=r; };
       Endwhile
       If (v=1) return 1 Else return u.
End
```

Example: With Fibonacci numbers $u = F_{17} = 1597$ and $v = F_{16} = 987$, we obtain:

q	r	*reduction*	
	1597	u	
	987	v	
1	610	r	
	377	$v - r$	
	305	$r/2$	
1	72	r	
	233	$v - r$	
	9	$r/8$	
25	8	r	
	1	$v - r$	
	1	$r/8$	`STOP`

Note that Euclid algorithm gives the answer after 15 iterations, and its extended version gives $-377\ u + 610\ v = 1 = \gcd(u, v)$, while `MBE` algorithm gives a modular relation

$$(-55\ u + 89\ v) = 8 = 2^3 \gcd(u, v).$$

Moreover, we observe that the coefficients -55 and 89 are smaller than -377 and 610. We know that the cofactors of Bezout relation are as large as the size of the inputs (consider successive Fibonacci worst case inputs). So an interesting question is : What is the upper bound for the modular coefficients a and b in the relation

$$au + bv = 2^t\ \gcd(u, v).$$

Let denote $r = u \bmod v$ and $s = r/2^t$ if r is even and $s = (v-r)/2^t$ otherwise. The reduction step used by Harris is

$$(u, v) \leftarrow (v, s),$$

while the `MBE` reduction step is

$$(u, v) \leftarrow \begin{cases} (r, s) & \text{if} \quad r \geq s, \\ (s, r) & \text{if} \quad s > r. \end{cases}$$

This difference leads to a different algorithm. For example, if $(u, v) = (4901, 2687)$, Harris's algorithm gives the result 1, after 6 iterations, which are respectively $(2687, 1107)$, $(1107, 317)$, $(317, 39)$, $(39, 17)$, $(17, 3)$ and $(3, 1)$, while `MBE` returns 1 after 4 iterations $(1107, 473)$, $(161, 39)$, $(17, 5)$ and $(3, 1)$. Actually, `MBE` replaces many divisions of Harris's reductions by tests or subtractions.

2.2 Complexity analysis

First of all, thanks to Lemma 2.1, we have a upper bound of the number of iterations of the main loop. We have $(u, v) \to (u', v')$, such that $v' \leq v/3$, so after k iterartions, we obtain $1 \leq v/3^k < 2^n/3^k$ or, $3^k < 2^n$, hence a first upper bound

$$k \leq \lfloor (\log_3 2)\ n \rfloor.$$

However, the following lemma proves that the worst case provides a smaller upper bound.

Lemma 2.2 *Let $k \geq 1$ and let us consider the sequence of vectors $\begin{pmatrix} r_k \\ s_k \end{pmatrix}$ defined by*

$$\forall k \geq 1, \quad \begin{pmatrix} r_{k+1} \\ s_{k+1} \end{pmatrix} = \begin{pmatrix} 2r_k + 2s_k \\ 2r_k + s_k \end{pmatrix} \text{ and } \begin{pmatrix} r_1 \\ s_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Then the worst case of algorithm `MBE` *occurs when the inputs (u, v) are equal to*

$$\begin{pmatrix} u_k \\ v_k \end{pmatrix} = \begin{pmatrix} 2r_k + s_k = s_{k+1} \\ r_k + s_k = r_{k+1}/2 \end{pmatrix},$$

and the gcd is given after k iterations.

Proof: First of all, we can easily prove by induction that

$$\begin{cases} \forall k \geq 1, \ r_k \text{ is even, } s_k \text{ and } \frac{r_k}{2} \text{ are odd} \\ \forall k \geq 2, \ \frac{r_k}{2} < s_k < r_k \\ \forall k \geq 2, \ \lfloor \frac{u_k}{v_k} \rfloor = 1. \end{cases}$$

We call an *iteration*, each iteration of the (**while** $v > 1$) loop. We prove by induction that, at each iteration k, we have $q_k = 1$ and the triplets $(r_k, s_k, \frac{r_k}{2})$, for $k \geq 2$.
After the first iteration with the inputs $(u_k = 2r_k + s_k, v_k = r_k + s_k)$, we obtain the

triplet $(r_k, s_k, \frac{r_k}{2})$ since r_k is even and $\frac{r_k}{2}$ is odd. The relation $\frac{r_k}{2} < s_k < r_k$ yields and the next quotient q_{k-1} will be $q_{k-1} = \lfloor \frac{s_k}{r_k/2} \rfloor = 1$. We repeat the same process with the new triplet$(r_{k-1}, s_{k-1}, \frac{r_k}{2})$ until we reach the triplet $(r_1, s_1, \frac{r_1}{2}) = (2, 1, 1)$ which is the smallest possible. □

Example: For $k = 7$ we have $u_7 = 9805$ and $v_7 = 6279$. We obtain 7 iterations. Note that Euclid algorithm gives the answer after 12 iterations.

Proposition 2.1 *Let $u \geq v \geq 11$ be two integers, where u is an n-bit integer. If k is the number of iterations when algorithm* `MBE` *is applied then*

$$k \leq \lceil \frac{n}{\log_2 \lambda} \rceil, \quad \text{with } \lambda = \frac{3+\sqrt{17}}{2}.$$

Proof: Let $u \geq v \geq 11$ be two integers, where u is an n-bit integer, so that $2^{n-1} \leq u < 2^n$. Let us denote $A = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$, so, for each $k \geq 1$,

$$\begin{pmatrix} r_{k+1} \\ s_{k+1} \end{pmatrix} = A \begin{pmatrix} r_k \\ s_k \end{pmatrix}.$$

Let $\lambda_1 = \frac{3+\sqrt{17}}{2}$ and $\lambda_2 = \frac{3-\sqrt{17}}{2}$ be the enginevalues of A. Then the worst case occurs after k iterations with $u \leq C\ (\lambda_1)^k < 2^n$, where C is some positive constant. As a matter of fact we prove easily by induction or by diagonalization of matrix A, that $\forall k \geq 1$

$$\begin{cases} r_k = & \frac{2}{\sqrt{17}}\ (\lambda_1^k - \lambda_2^k) \\ s_k = & (\frac{\sqrt{17}-1}{2\sqrt{17}})\ \lambda_1^k\ +\ (\frac{\sqrt{17}+1}{2\sqrt{17}})\ \lambda_2^k \end{cases}$$

Then

$$2^{n-1} \leq u_k = 2r_k + s_k = s_{k+1} = \lambda_1^{k+1}\ (C + \epsilon_k) < 2^n, \quad \text{with } \lim_{k\to\infty} \epsilon_k = 0,$$

where $C = \frac{\sqrt{17}-1}{2\sqrt{17}}$, $\epsilon_k = \frac{\sqrt{17}+1}{\sqrt{17}-1}\ (\frac{\lambda_2}{\lambda_1})^{k+1}$ and $\lambda_1 = \frac{3+\sqrt{17}}{2} \sim 3,561552813$.
We have

$$n - 1 \leq (k+1)\log_2 \lambda_1 + \log_2(C + \epsilon_k) < n$$

and after a bit of calculation, we find that $\forall k \geq 3,\ \frac{1}{4} < C + \epsilon(k) < \frac{1}{2}$.
Hence $1 < -\log_2(C + \epsilon_k) < 2$ and $\frac{n}{\log_2 \lambda_1} < k + 1 < \frac{n+2}{\log_2 \lambda_1}$, so

$$k = \lfloor \frac{n}{\log_2(\lambda_1)} \rfloor \quad \text{or} \quad k = \lfloor \frac{n}{\log_2(\lambda_1)} \rfloor + 1.$$

□

Remark: We have $k \sim (\frac{\log 2}{\log \lambda})\ n \sim 0,545700691\ n$. By contrast, when euclidean algorithm is applied to n-bit integers, the number of iterations is bounded by $k' \leq (\frac{\log 2}{\log \phi})\ n \sim 1,440420091\ n$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. Indeed, a first experiments on 1000 pair of 32-bit integers shows that our algorithm is about 3 time faster than Euclid algorithm.

3 Multi-precision Algorithm

In order to avoid long divisions, we must consider some leading bits of the inputs (u, v) for computing the quotients and some other last significant bits to know if either $r = u \bmod v$ or $s = v - r$ is even. We propose the following multi-precision algorithm (sketch).

$M = Id$;

Step 1: Consider u_1 and v_1 the first $2m$ leading bits of respectively u and v. Similarly, consider u_2 and v_2 the last $2m$ significand bits of respectively u and v.

Step 2: By ρ-Euclid algorithm, compute q_1 . Compute $r_1 = |u_1 - q_1 v_1|$ and $s_1 = v_1 - r_1$. Similarly, compute $r_2 = |u_2 - q_2 v_1|$ and $s_2 = v_2 - r_2$.

Step 3: Compute t_1 and p_1 such that $r_2/2^{t_1}$ and $s_2/2^{p_1}$ are both odd.

Step 4: Save the computations: $M \leftarrow M \times N$, where N is defined by:

Case 1: r_2 is even

If $r_1/2^{t_1} \geq s_1$ then $N = \begin{pmatrix} 1/2^{t_1} & -q/2^{t_1} \\ -1 & q+1 \end{pmatrix}$, otherwise $N = \begin{pmatrix} -1 & q+1 \\ 1/2^{t_1} & -q/2^{t_1} \end{pmatrix}$.

Case 2: s_2 is even

If $s_1/2^{p_1} \geq r_1$ then $N = \begin{pmatrix} -1/2^{p_1} & (q+1)/2^{p_1} \\ 1 & -q \end{pmatrix}$, otherwise $N = \begin{pmatrix} 1 & -q \\ -1/2^{p_1} & (q+1)/2^{p_1} \end{pmatrix}$

Example: Let u and v be two odd integers such that:

$u = 1617 \ldots 309$, and

$v = 1045 \ldots 817$.

We obatin, in turn, $N_1 = \begin{pmatrix} -1 & 2 \\ 1/4 & -1/4 \end{pmatrix}$ and $N_2 = \begin{pmatrix} -1 & 5 \\ 1/4 & -1 \end{pmatrix}$. Then the two steps are saved in the matrix

$$M = N_2 \times N_1 = \begin{pmatrix} 9/4 & -13/4 \\ -1/2 & 3/4 \end{pmatrix}.$$

4 The Parallel Algorithm

It is based one parallel `MBE` reduction:
Begin
Step 1 **:**
For $i = 1$ **to** n $R[i] = 0$, $S[i] = 0$.
Compute, in parallel, $r_i = |iu - q'_i v|$ and $s_i = v - r_i$, **for** $i = 1, 2, \ldots, n$,
(see JDA'08 or ISSAC'01).
Step 2 **:**
While ($r_i > 0$ and r_i even) **Do** in parallel $r_i \leftarrow r_i/2$;
If ($r_i < 2v/n$) **then** $R[i] = r_i$, in parallel.
Step 3 **:**
While ($s_i > 0$ and s_i even) **Do** in parallel $s_i \leftarrow s_i/2$;
If ($s_i < 2v/n$) **then** $S[i] = s_i$, in parallel.
Step 4 **:**
$r = \min\ \{R[i]\}$; $s = \min\ \{S[i]\}$; in $O(1)$ parallel time;
If $r \geq s$ **Return** (r, s) **Else Return** (s, r).
End.

5 Complexity Analysis

We give below the complexity analysis of the parallel MBE-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 [9]).

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 [9] or [3] for more details). The computation of $R_{MBE} = |iu - q_i v|$ requires (see Figure 2) only two products iu and $q_i v$ with the selected index i. Thus R_{MBE} can be computed in parallel in $O(1)$ time with: $(\rho < m)$

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

R_{MBE} reduces the size of the smallest input v by at least $m - 1$ bits. Hence the $MBE-GCD$ algorithm runs in $O(n/m)$ iterations. For $m = 1/2\ \epsilon \log n$, $(\epsilon > 0)$ the parallel

$MBE - GCD$ 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 Conclusion

Instead of simplifying by 2 at each step, we may consider to simplify by several consecutive primes $p_1 = 2, p_2 = 3, \ldots p_k$ (Filter process).

A big Example (given by program).

```
u = 21441679871021215487845145411121017
v = 12125999210313477414021337054676451
 q = 1    k = 4657840330353869036911904178222283   d =2810318549605739340197528698231885
 q = 1    k = 962796768857609643483153218241487    d =923760890374064848357187739995199
 q = 1    k = 884725011890520053231222261748911    d =2439742405221549695372842390393
 q = 362   k = 15382612003190635062533164266645    d =225370301225621547279881490937
 q = 6    k = 186039392965334222574027481023       d =19665454130143662352927004957
 q = 9    k = 10615148336102400955242568547        d =4525152897020630698842218205
 q = 2    k = 1564842542061139557558132137         d =740077588739872785321021517
 q = 2    k = 327695112079239399202466207          d =84687364581393986916089103
 q = 3    k = 36816509167528719227099449           d =11054346246336548461890205
 q = 3    k = 7400875817817474620461371    d =1826735214259536920714417
 q = 4    k = 866400126740104991555357    d =93934960779326937603703
 q = 9    k = 72949481053164384481673     d =10492739863081276561015
 q = 6    k = 9993041874676725115583      d =62462248550568930679
 q = 159     k = 30772177568132568811     d =917893414303793057
 q = 33      k = 436198518196395127       d =240847448053698965
 q = 1    k = 97675535071348081    d =45496377911002803
 q = 2    k = 6682779249342475     d =4851699832707541
 q = 1    k = 3020620416072607     d =915539708317467
 q = 3    k = 641538417197261      d =137000645560103
 q = 4    k = 93535834956849       d =21732405301627
 q = 4    k = 7563095775643        d =6606213750341
 q = 1    k = 5649331725039        d =478441012651
 q = 11   k = 193240292939         d =91960426773
 q = 2    k = 20660246845          d =9319439393
 q = 2    k = 3649035667           d =2021368059
 q = 1    k = 393700451    d =203458451
 q = 1    k = 13216451     d =11890125
 q = 1    k = 10563799     d =663163
 q = 15   k = 308177       d =46809
 q = 6    k = 27323        d =9743
 q = 2    k = 7837         d =953
```

```
 q = 8   k = 213         d =185
 q = 1   k = 157         d =7
 q = 22  k = 3           d =1
Nb. of iterations:  34

And for u =  125545454541212197979612012145663217
68754132115212487879421021215415451521454854854811471
and v =
100215479651212167975956297491595921909921972l9
5192162197541971062912979219071990090299957
We find
Nb. of iterations:  81
```

References

[1] **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.

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

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

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

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

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

[7] **M.S. Sedjelmaci.** A Modular Reduction for GCD Computation, *Journal of Computational and Applied Mathematics*, Vol. 162-I, 2004, 17-31.

[8] **M.S. Sedjelmaci.** On A Parallel Lehmer-Euclid GCD Algorithm, *in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'2001)*, 2001, 303-308.

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

[10] **D. Stehle, P. Zimmermann.** A Binary Recursive Gcd Algorithm, , *in Proc. of ANTS VI,University of Vermont,USA,* , June 13-18,2004, 411-425.

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

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