Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Discrete Mathematics 35 (2009) 169–176
Electronic Notes in DISCRETE MATHEMATICS
www.elsevier.com/locate/endm

The Mixed Binary Euclid Algorithm

Sidi Mohamed Sedjelmaci

LIPN CNRS UMR 7030
Université Paris-Nord
Av. J.-B. Clément, 93430, Villetaneuse, France
Email: 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 we prove that its bit-time complexity is still $O(n^2)$ for two n-bit integers in the worst case. Our preliminar experiments show a potential speedup for small integers. A parallel version matches the best presently known time complexity, namely $O(n/\log n)$ time with $O(n^{1+\epsilon})$ processors, for any constant $\epsilon > 0$.

Keywords: Greatest common divisor (GCD); Parallel Complexity; Algorithms.

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 $GCD(u, v) = GCD(u', v')$ [6]. These transformations are applied several times till a pair $(u', 0)$ is reached. Such transformations, also called *reductions*, are studied in a general framework in [6].

1571-0653/$ – see front matter
doi:10.1016/j.endm.2009.11.029

For very large integers, the fastest GCD algorithms [2,5,9,10] are all based on half-gcd procedure and computes the GCD in $O(n \log^2 n \log\log n)$ time, where n is the size of the larger input. Although the algorithm of T. Jebelean [1] and K. Weber [11] is quadratic in time, it has proven to be highly effecient for large and medium size integers. 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.

In this paper, we are interested in small and medium 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 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 $n^{1+\epsilon}$ processors, $\epsilon > 0$ (see [3,8,7]).

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 algorithm with some other integers, especially when the quotients are large. So, the idea is that, instead of choosing one of them, one may 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 (cited by Knuth [4]) 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. The basic gcd property is $\forall \lambda \geq 0,\ \gcd(u, v) = \gcd(v, u - \lambda v)$. 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, $\gcd(\frac{v-r}{2}, r) = \gcd(r, v - r) = \gcd(v, r) = \gcd(u, v)$. □

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)
      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.
```

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

q	r	*reduction*	
	1597	u	
	987	v	
1	610	$r = u - qv$	
	377	$s = v - r$	
	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 ex-

tended version gives: $-377\ u + 610\ v = 1 = \gcd(u, v)$, while MBE algorithm gives a modular relation $(-55\ u + 89\ v) = 8 = 2^3 \gcd(u, v)$, after 3 iterations. Moreover, we observe that the coefficients -55 and 89 are smaller than -377 and 610. We know that the cofactors of Bézout relation are roughly 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 $au + bv = 2^t\ \gcd(u, v)$?

2.2 Complexity analysis

First of all, thanks to Lemma 2.1, we have an upper bound for the number of iterations of the main loop. We have $(u, v) \to (u', v')$, such that $v' \leq v/3$, so after k iterations, we obtain $1 \leq v/3^k < 2^n/3^k$ or, $3^k < 2^n$, hence a first upper bound $k \leq \lfloor(\log_3 2)\ n\rfloor$. So the algorithm is quadratic in bit complexity as the binary or Euclidean algorithms. However, the following lemma proves that the worst case provides a smaller upper bound for the number of iterations.

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*

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

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. Roughly speaking, the worst case is reached when, at each time, the quotient is 1 (the smallest), only one division by 2 occurs and the output is the smallest one. 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 output triplet 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 2.3 *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 $A = \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix}$, 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, after a bit of calculation, we obtain

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

□

Remark: Note that $k \sim (\frac{\log 2}{\log \lambda})\ n \sim 0,54\ n$, while, when euclidean algorithm is applied to n-bit integers, the number of iterations is bounded by $k' \leq (\frac{\log 2}{\log \phi})\ n \sim 1,44\ n$, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. Indeed, a

first experiment on 1000 pair of 32-bit integers shows that our algorithm is about 3 time faster than Euclid algorithm. Other experiments, kindly done by Ken Weber, show a clear speed up at least for single precision with 32 bits and double precision with 64 bits (i.e.: 128 bits). A complete study of these experiments will appear later in a long version of this paper.

3 The 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. The algorithm is based on the following multi-precision reduction step (sketch) called `MP-MBE`. The integer m is a parameter choosen as in [6].

$M = Id$;
Step 1**:** Consider u_1 and v_1 the first $2m$ leading bits of respectively u and v. Similarly, u_2 and v_2 are the last $2m$ significant bits of respectively u and v.

Step 2**:** By Euclid algorithm, compute $q_1 = \lfloor u_1/v_1 \rfloor$. Compute $r_1 = |u_1 - q_1 v_1|$ and $s_1 = v_1 - r_1$. Similarly, compute $r_2 = |u_2 - q_1 v_2|$ and $s_2 = v_2 - r_2$ (see [6] for more details).

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

$$N = \begin{pmatrix} 1/2^{t_1} & -q/2^{t_1} \\ -1 & q+1 \end{pmatrix}, \text{ otherwise } N = \begin{pmatrix} -1 & q+1 \\ 1/2^{t_1} & -q/2^{t_1} \end{pmatrix}.$$

Case 2**:** s_2 is even. If $s_1/2^{p_1} \geq r_1$ then

$$N = \begin{pmatrix} -1/2^{p_1} & (q+1)/2^{p_1} \\ 1 & -q \end{pmatrix}, \text{ otherwise } N = \begin{pmatrix} 1 & -q \\ -1/2^{p_1} & (q+1)/2^{p_1} \end{pmatrix}.$$

Example: Let u and v be two odd integers such that: $u = 1617 \ldots 309$, and $v = 1045 \ldots 817$. We obtain, 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:

A parallel GCD algorithm can be designed based on the following `Par-MBE` reduction:

Begin ($k = 2^m$ is a parameter)
Step 1 **:** (in parallel)
  For $i = 1$ **to** n $R[i] = v$, $S[i] = v$; $q_i = \lfloor u_i/v_i \rfloor$; (as in Step 2 of `MP-MBE`)
  For $i = 1$ **to** n $r_i = |iu - q_i v|$ and $s_i = v - r_i$; (see [7])
Step 2 **:**
  While ($r_i > 0$ and r_i even) **Do** $r_i \leftarrow r_i/2$;
  If ($r_i < 2v/k$) **then** $R[i] = r_i$, in parallel.
Step 3 **:**
  While ($s_i > 0$ and s_i even) **Do** $s_i \leftarrow s_i/2$;
  If ($s_i < 2v/k$) **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.

4.1 Complexity Analysis

The complexity analysis of the parallel GCD algorithm based on `Par-MBE` reduction is similar to that of `Par-ILE` in [7]. We compute in turn q_i, $r_i = |iu - q_i v|$, $s_i = v - r_i$ and test if $r_i < 2v/k$ or $s_i = v - r_i < 2v/k$ to select the index i. Note that there is no write concurrency. Recall that $k = 2^m$ is a parameter. 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 [7,8]).
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 [8] or [3] for more details). The computation of $r_i = |iu - q_i v|$ and $s_i = v - r_i$ require only two products iu and $q_i v$ with the selected index i. Thus the reduction `Par-MBE` can be computed in parallel in $O(1)$ time with:

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

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

Acknowledgement: The author would like to thank Professor Ken Weber for experimenting the algorithm `MBE`.

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

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

[7] S.M. Sedjelmaci, *A Parallel Extended GCD Algorithm*, Journal of Discrete Algorithms, 6, 2008, 526-538

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

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

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

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