On a Parallel Extended Euclidean Algorithm

Sidi Mohammed SEDJELMACI
LIPN UPRES-A CNRS 7030
Université Paris Nord. Institut Galilée
Avenue. J.B.Clement. 93430. Villetaneuse. France.

E-mail : sms@lipn.univ-paris13.fr
Telephone : (+33) 01 49 40 35 95
Fax : (+33) 01 48 26 07 12.

Abstract

A new parallelization of Euclid's greatest common divisor algorithm is proposed. It matches the best existing integer GCD algorithms since it can be achieved in parallel $O_\varepsilon(n/\log n)$ time using only $n^{1+\varepsilon}$ processors on a Priority CRCW PRAM.

Keywords : Parallel complexity, parallel algorithms, Greatest Common Divisor (GCD), Euclid algorithm.

I. Introduction

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

The advent of practical parallel computers has caused the re-examination of many existing algorithms with the hope of discorering a parallel implementation. In 1987, Kannan, Miller and Rudolph [5] gave the first sublinear time parallel integer GCD algorithm on a common CRCW PRAM model (See [6] for PRAM's models). Their time bound is O(nlog log n/log n) assuming there are $n^2(\log n)^2$ processors working in parallel. Since 1990, Chor and Goldreich [2] currently have the fastest parallel GCD algorithm; it is based on the systolic array GCD algorithm of Brent and Kung [1]. The time complexity of their algorithm achieves $O_\varepsilon(n/\log n)$ using only $n^{1+\varepsilon}$ processors on a CRCW PRAM. More recently (1994), Sorenson's right- and left-shift k-ary algorithms [8] match Chor and Goldreich's performance.

Euclid's algorithm is one of the simplest and most popular integer GCD algorithm. Its exented version called *Extended Euclidean Algorithm* or *EEA* for short (see Knuth [7]) is tightly linked with the continued fractions [3], [7] and is important for its multiple appplications (cryptology, modular inversion, etc..). In [5], Kannan, Miller and Rudolph proposed a first parallelization of E.E.A. Their algorithm was based on a reduction step which finds a couple (p,q) s.t. : $|p| \leq kv/u$, $|q| \leq 2k$ and the relation $0 \leq pu - qv \leq u/k$, for a given parameter $k > 0$; reducing therefore, at each step, the larger input u by O(log k) bits.

However, one of the major drawback of their algorithm is the computation of the couple (p,q). As a matter of fact, in order to reach an O(1) time computation for their reduction step, they must compare the $O(n^2)$ equations pu - qv all each other. Thus, by assigning $O(\log^2 n)$ processors to each $O(n^2)$ equations, more than $O(n^2 \log^2 n)$ processors are needed (see [5] for more details). The main results of the paper are summarized below:

- We show how to overcome this drawback and obtain a more accurate result with only O(1) comparisons of O(log n) bit-numbers.
- A new GCD algorithm, based on this reduction step is designed, and its performance matches the best known GCD algorithms, i.e. $O_\varepsilon(n/\log n)$ time using at most $n^{1+\varepsilon}$ processors on a *Prioprity* CRCW PRAM model, for any constant $\varepsilon > 0$.
- Moreover a compression method may be considered to improve the complexity.

In Section 2, we define our basic reduction which emphases the Kannan, Miller and Rudolph's one and parallelizes the Etended Euclidean Algorithm. As in [5] our reduction is based on the computation of the remainders $r_i = (iu) \bmod v$, $1 \leq i \leq k-1$, for a given parameter $k>0$. In Section 3, we set necessary conditions on the most significant O(log n) bits of r_i, to find all the remainders satisfying r_i or $v - r_i < v/k$. In Section 4, we show how to find such remainders and a GCD algorithm based on this reduction step is designed. Section 5 is devoted to the complexity analysis. Finally, we end with concluding remarks.

2. Basic Reduction Step

2.1 Notation

Throughout, we restrict ourselves to the set N^* of positive integers. Let u and v be two such integers such that $u \geq v > 0$, u and v are respectively n-bit and p-bit numbers. Let k an integer parameter, such that $k = 2^m$, $m > 0$ and $k = O(\log n)$.

Most of serial or parallel GCD algorithms use one or several transformations $(u,v)\rightarrow(v,R(u,v))$, which reduces the size of current pairs (u,v) till (u,0) is reached. The last value u = gcd(u,v) is then the result we want to find. Such transformations *R* will be called *reductions*. The number of significant bits of an integer x, not counting leading zeros is denoted by $l_2(x)$:

$$l_2(x) = \lfloor \log_2(x) \rfloor + 1 \qquad \text{for all } x \geq 1 \text{ and } l_2(0) = 1.$$

So $n = \lfloor \log_2(u) \rfloor + 1$ and $p = \lfloor \log_2(v) \rfloor + 1$. The number p satisfies $2^{p-1} \leq v < 2^p$.
If we let $\rho = l_2(u) - l_2(v) + 1 = n - p + 1$, then $2^{\rho-2} < u/v < 2^\rho$. We assume that $p - m > 3$ and consider the numbers V and R_i obtained by the (m+2) most significant leading bits of respectively v and r_i :

$$V = \lfloor v/2^{p-m-2} \rfloor, \quad R_i = \lfloor r_i/2^{p-m-2} \rfloor,$$

as well as the number formed by the 2 most significant leading bits of v (or V), i.e. :
$W = \lfloor v/2^{p-2} \rfloor$. Note that W = 2 or 3.

2.2 The Kannan-Miller-Rudolph Reduction

The Kannan Miller Rudolph (*KMR* for short) integer GCD algorithm is based on the following reduction step [5, lemma 2, page 9] :

For all positive integers u, v and k with $u \leq kv$, there exists a pair $(p,q) \neq (0,0)$ s.t. $|p| \leq kv/u$ and $|q| \leq 2k$ which satisfies : $0 \leq |pu-qv| \leq u/k$.

Remark : Since $|pu-qv| < v$, then $|pu-qv| = (pu) \bmod v$ or $v - (pu) \bmod v$.

Thus any couple (p,q) found provides a reduction called *KMR's reduction*. Kannan Miller and Rudolph proposed to compute in parallel all the $O(k^2)$ numbers pu – qv, and select those for which $0 \leq |pu-qv| \leq u/k$. But this latter relation implies $|pu-qv| < v$, thus a couple (p,q) must be chosen from a set of O(k) numbers satisfying this relation. However, in the weak model of parallel processing *Common* CRCW, simultaneous writes are allowed provided that all the processors write the same value. Finally, in order to reach an O(1) time process, more than $O(n^2\log^2 n)$ processors are needed (see [5] for more details).

By contrast, in our reduction, we consider only the two remainders with the smallest index and compare their O(log n) first leadind bits. Thanks to *Priority* CRCW model, these two remainders can be easily reached because this model allows the write to the processor with the smallest index (see Section 4.2).

2.3 Parallelized EEA

Let (r_i) be the sequence of the remainders $r_i = (iu) \bmod v$, for i = 1,..,k - 1. Applying a theorem due to Hardy and Wright [3, theorem 36, p. 30] to u/v, we obtain a more accurate result than KMR's one : there exist an irreducible fraction (q_i/i), s.t. $1 \leq i \leq k-1$ and $|iu - q_i v| < v/k$. Thus for this such index i, we have :

$$r_i < v/k \text{ or } v - r_i < v/k. \qquad (1)$$

Any remainder r_i satisfying (1) will be called a *solution* of order k. Note that if r_i is a solution then q_i/i is a continant fraction [3], [7] of u/v since

$$|u/v - q_i/i| = |iu - q_i v| / iv < (v/k) / iv = 1/ik < 1/i^2.$$

Thus the couple (i,q_i) is also obtained by EEA and this is why the reduction defined by the previous couple is a parallelization of EEA.

3. Finding Solutions

The aim of the next sections is to determine in parallel such solutions, i.e. the couples (i,q_i) satisfying relation (1) with only O(1) comparisons on O(log n) bit-numbers.

3.1 Trivial cases

It is easy to prove that the necessary conditions are the following :

i) $r_i < v/k \Rightarrow R_i \leq W$

ii) $v - r_i < v/k \Rightarrow V - R_i \leq W + 1$,

and the trivial cases are:

1) $R_i < W \Rightarrow r_i < v/k$

2) $V - R_i < W \Rightarrow v - r_i < v/k$

3) $R_i > W \Rightarrow r_i > v/k$

4) $V - R_i > W + 1 \Rightarrow v - r_i > v/k$.

By contrast there are some cases where no decision can be done from R_i. The latter will be called *doubtful cases*. We identify two types:

$R_i = W$, (Type I)

$V- R_i = W$ or $W+1$, (Type II).

We show in the next section that only R_i and R_j, the (m+2) first leading bits of the two solutions with the smallest indexes are needed to determine a solution.

3.2 A solution

We assume that no solution r_i is already known, and r_i is not trivial. Let E be the set of doubtful cases, i.e. $E = \{ r_i / R_i = W$ or $V-R_i = W$ or $W+1\}$. Thus, two cases arise :

Case 1 : $E = \{r_i\}$, then r_i is a solution if $R_i = W$, otherwise it is $v- r_i$.

Case 2 : $\#(E) \geq 2$ (E has at least two different elements). Let I and J be the two smallest indexes of the elements of E, i.e. : $(I < J)$

$$\forall \lambda \in \{1,..,k-1\} \ r_\lambda \in E \Rightarrow \lambda = I \text{ or } \lambda \geq J.$$

We prove that r_I and r_J are not of the same type and :

1) $|v – (r_I + r_J)| = r_{I+J}$ or $v - r_{I+J}$

2) $|v – (r_I + r_J)|$ is a solution if $I+J <k$, otherwise r_I or r_J is a solution.

Examples :

- **Case I+J < k** : Let (u,v) = (316,269) and k = 8, $r_1 = 47$ and $v- r_5 = 34$ are doubtful cases since $R_1 = V - R_5 = W = 2$ and $|v – (r_1 + r_6)| = r_6 = 13$ is a solution.
- **Case I+J ≥ k** : Let (u,v) = (379,337) and k = 8, $r_1 = 42$ and $v - r_7 = 43$ are doubtful cases since $R_1= W = 2$ and $V - R_7 = W + 1 = 3$. In this case r_1 is a solution. However $|v – (r_1 + r_7)| = |v – r_8| = 1$ is a solution of order 2k = 16, at least.

4. The BA-GCD Algorithm

We let $k = 2^m$, where m is a multiple of a memory word ω (ω = 16, 32 or 64 bits). The reduction described in the previous section will be called the *B.A* reduction which stands to *Best Approximation* reduction. Recall that we have assumed k = O(n) and m = O(log n) for computing the remainders; this value yields O(n/log n) iterations at most.

4.1 High level description

We give below a top-down description of our a GCD algorithm based on BA reduction. Let d=gcd(u,v).

Step 1: Find d_1 s.t. d_1 equals the product of all common divisors to u and v which are less than k. (In the algorithm $d:=d_1\ d$)

Step 2: Perform reductions until $v < k$: if $\rho \leq m$, then perform BAs; else, perform the bmod reduction.
Compute d = gcd(u,v), where (u,v) is the last pair obtained from Euclid's algorithm.

Step 3: Remove all divisors < k from d.

Step 4: Perform the product $d \times d_1$.

The BA-GCD Algorithm.

Step 1, 3 and 4 are similar to the phases of KMR's gcd algorithm [5], while Step 2 is designed below (Refer to [9] for the bmod). The computation of BA is done as follows :

Begin
Type_1 := - 1 ; Type_2 := - 1 ;
begin (Steps 1, 2 and 3 are done in parallel for i = 1,2,..,k – 1).
Step 1 : Compute r_i (See Section 4.3) ;
Step 2 : Compute R_i, V and W (see notation 2.1).
Step 3 : **/*** Trivial cases */
If R_i < W **then** BA := r_i ; **Exit** ; **end**.
Else
If V - R_i < W **then** BA := v - r_i ; **Exit** ; **end**.
Else
If (R_i > W or V - R_i > W+1) **then Exit** ; **end**.
Else Step 4.
end
Step 4 : **/*** Doubtful cases */
If R_i = W **then** Type_1 := r_i ; I := i
Else
If (V - R_i = W **or** W+1) **then** Type_2 := v - r_i ; J := i ;
Case Type_1 > 0 **and** Type_2 < 0
BA := Type_1 ; **Exit** ; **end**.
Case Type_1 < 0 **and** Type_2 > 0
BA := Type_2 ; **Exit** ; **end**.
Case Type_1 > 0 **and** Type_2 > 0
If I + J < k **then** BA := |Type_1 – Type_2| ; **Exit** ; **end**.
Else Step 5.
Step 5 :
Let V', R'_I and R'_J are the first leading (2m+2) bits of respectively v, r_I and r_J.
Case V' – R'_J < R'_I
B.A := Type_2 ;
Case R'_I < V' – R'_J - 1
B.A := Type_1 ;
Case R'_I = V' – R'_J **or** R'_I = V' – R'_J – 1
B.A := |Type_1 – type_2| ; /* A solution of order k^2.*/
End.

Step 2 of BA-GCD Algorithm.

If many remainders are sent as result then we take the remainder with the smallest index. This choice is straightforward obtained by the *Priority* CRCW PRAM model [6]. We must specify how to compute the couple (i,q_i) and the remainders r_i.

4.2 Computation of the Remaiders r_i

BA reduction is performed in BA-GCD algorithm only when $\rho < m$ ($\rho = l_2(u) - l_2(v) + 1$). For each processor i, ($1 \le i \le k - 1$) an approximation of $q_i = \lfloor iu/v \rfloor$ is found as follows.

Let $l = l_2(iu) - l_2(v)$. With the (l + 2m) first leading bits of iu, denoted by $(iu)_1$ and the first 2m bits of v denoted by v_1, the quotient $q'_i = \lfloor (iu)_1/v_1 \rfloor$ is computed. This approximation of q_i is very close to q_i. More precisely we can prove (easy calculation) that $q'_i = q_i$ or $q'_i = q_i + 1$. This

result generalizes a previous Kannan Miller and Rudolph's lemma [5], but their result was less accurate since they obtain $| q - q' | \leq 3$.

Moreover, we do not need to know exactly the quotient q_i and the remainder r_i, and it is worth noting that if we use q_i' and r_i' instead of q_i and r_i, then the result of the following tests remains the same :

$$r_i < v/k \text{ or } v - r_i < v/k.$$

5. Complexity Analysis.

Precomputation table look up can be used for multipying two m-bit numbers in O(1) time using $O(n2^{2m})$ processors on a CRCW PRAM [8], [2]. Therefore, the computation of B.A can be achieved in constant time with : $O(k) + O(n2^{2m}) + O(n \log\log n) = O(n2^{2m})$ processors. Since there is at most O(n/log n) iterations, and taking $m = ½ \varepsilon \log n$, the complexity of the B.A-GCD algorithm matches the best previous GCD algorithms with $O_\varepsilon(n/\log n)$ time with $n^{1+\varepsilon}$ processors on a CRCW PRAM.

Conclusion

Since this last decade, no major improvement has been done for the computation of the integer GCD and the performance of $O_\varepsilon(n/\log n)$ time with $n^{1+\varepsilon}$ processors on a CRCW PRAM seems to be a « limit » not easy to overcome.

We have designed an integer GCD algorithm based on this reduction which matches the best existing GCD algorithms [2] [8] without compression method. Our algorithm follows and improves the same *Most Significant digit First* (MSF) approach of Kannan Miller and Rudolph. Almost all the decisions are made with the first leading bits of the operands, thus, the techniques used in our algorithm suits well for such compression method. We are currently investigate this way with the hope of improving the best performance of parallel integer GCD algorithms.

REFERENCES

[1] 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.

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

[3] G.H Hardy and E.M. Wright, "An Introduction to the Theory of Numbers", 5th ed., Oxford Univ. Press, London, 1979.

[4] 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.

[5] 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.

[6] R. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. Van Leeuwen, editor, Algorithms and complexity. Elsvier and MIT Press, 1990. Handbook of Theoretical Computer Science, volume A.

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

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

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