Available online at www.sciencedirect.com

ScienceDirect

Journal of Discrete Algorithms 6 (2008) 526–538

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

A parallel extended GCD algorithm [☆]

Sidi Mohamed Sedjelmaci

LIPN CNRS UMR 7030, Université Paris-Nord, 99 Avenue J.B. Clément, 93430 Villetaneuse, France

Received 11 May 2006; received in revised form 4 December 2006; accepted 5 December 2006

Available online 8 November 2007

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 $u \geqslant v$. 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.
© 2007 Elsevier B.V. All rights reserved.

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 implementation. In 1987, Kannan, Miller and Rudolph (*KMR*) [5] gave the first sub-linear 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 [1] 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 [10] match Chor and Goldreich's performance.

The extended version of parallel GCD algorithms is also discussed in Sorenson and Chor and Goldreich papers [1, 10]. 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 [10], Section 7.2, p. 141, lines 1, 2 and 3).

[☆] A preliminary version of this paper was presented at ISSAC'01 Conference, London Ontario, Canada, ACM Press, 2001, pp. 303–308.
E-mail address: sms@lipn.univ-paris13.fr.

1570-8667/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2006.12.009

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 [7] is tightly linked with the continued fractions [3,7] and is important for its multiple applications (cryptology, modular inversion, etc.). In [5], 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 $|a| \leqslant kv/u$, $|b| \leqslant 2k$, s.t. $0 \leqslant au - bv \leqslant u/k$ 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 [5] 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 [5]. In Section 3, we define a new reduction which uses the same Lehmer idea [8]. 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 $1 \leqslant a \leqslant k$, 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 $u \geqslant v$. Let k be an integer parameter s.t. $k = 2^m$ with $m \geqslant 2$ 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 sub-models 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 [6].

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:

$$(P_1) \quad 0 \leqslant R(u, v) < v.$$

$$(P_2) \quad \gcd(v, R(u, v)) = \alpha \gcd(u, v), \quad \text{with } \alpha > 0.$$

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 [4,10,12]. Examples of such basic reductions are given in Table 1.