Fast parallel GCD algorithm of many integers

Sidi M. SEDJELMACI
LIPN, CNRS UMR 7030
University of Paris-Nord
Av. J.-B. Clément, 93430 Villetaneuse, France
sms@lipn.univ-paris13.fr

Abstract

We present a new parallel algorithm which computes the GCD of n integers of $O(n)$ bits in $O(n/\log n)$ time with $O(n^{2+\epsilon})$ processors, for any $\epsilon > 0$ in CRCW PRAM model.

1 Introduction

The computation of the GCD of two integers is not known to be in the NC parallel, nor it is known to be P-complete [2]. The best parallel performance was first obtained by Chor and Goldreich [3], then by Sorenson [10] and Sedjelmaci [9] since they propose, with different approaches, parallel integer GCD algorithms which can be achieved in $O(n/\log n)$ time with $O(n^{1+\epsilon})$ number of processors, for any $\epsilon > 0$, in PRAM CRCW model (see [6] for CRCW model of computation). A naive approach, using a binary tree computation to compute the GCD of n integers of $O(n)$ bits would require $O(n)$ parallel time with $O(n^{2+\epsilon})$ processors. One may also use the existing parallel GCD algorithms of two integers and try to adapt them to reach the GCD of many integers. However, it is not obvious how to conserve the same parallel time with $O(n^{2+\epsilon})$ processors, which is roughly the bit-size of all the n input integers. In this paper, we prove that we can compute the GCD of $O(n)$ integers of n bits in only $O(n/\log n)$ parallel time with $O(n^{2+\epsilon})$ processors, for any $\epsilon > 0$ in CRCW PRAM model, in the worst case. Other probabilistic approaches are given in [4, 5]. To our knowledge, it is the first deterministic algorithm which computes the GCD of many integers with this parallel performance and polynomial work.

1.1 The Δ-GCD algorithm

Input: A vector $A = (a_0, a_1, \cdots, a_{n-1})$ of n positive integers, $n \geq 4$.
Output: $\gcd(a_0, a_1, \cdots, a_{n-1})$.

```
 If $A = (a_0, a_0, \cdots, a_0)$  Then Return $a_0$ ;  /* $A$ is a constant vector */
 $\alpha := a_0$ ; $I := 0$ ; $p := n$ ;
 While $(\alpha > 1)$ Do
    For $(i = 0)$ to $(n - 1)$ ParDo
       If $(0 < a_i \leq 2^n/p)$ Then { $\alpha := a_i$ ; $I := i$ ; }
    Endfor
```

```
        **If** $(\alpha > 2^n/p)$ **Then**  /* Compute in parallel $I, J$ and $\alpha$ */
            $\alpha := \min\{\,|\,a_i - a_j\,| > 0\,\} = a_I - a_J$ ;   $a_I := \alpha$ ;
        **Endif**
        **For** $(i = 0)$ **to** $(n-1)$ **ParDo**  /* Reduce all the $a_i$'s */
            **If** $(i \neq I)$ **Then** $a_i := a_i \bmod \alpha$ ;
        **Endfor**  /* $\forall\, i \neq I,\ 0 \leq a_i < \alpha$ */
        **If** $(\forall\, i \neq I,\, a_i = 0\,)$ **Then Return** $\alpha$ ;  /* Here $\alpha = \gcd(a_0, \cdots, a_{n-1})$ */
        $p := np$;
    **Endwhile**
    **Return** $\alpha$.
```

Remarks:
We use a weak version of the function min based the pigeonhole principle, where only the $O(\log n)$ leading bits of the integers are considered. The integer α is, at each while iteration, $O(\log n)$ bits less. More details for the computations of I, J and α are given in APPENDIX, as well as a first C program checking the correctness of the Δ-GCD algorithm.

1.2 Complexity analysis

Lemma 1: Let n, I be two integers, $n \geq 2$ and $0 \leq I \leq n-1$. Let $A = (a_0, a_1, \cdots, a_{n-1})^t$ and $V = (v_0, v_1, \cdots, v_{n-1})^t$ be two integral vectors defined by $v_I = a_I$ and $\forall i \neq I$, $v_i = a_i - q_i a_I$ for some integers q_i. Let M be the associated matrix defined by $V = M\,A$, then $\det(M) = 1$.

Proof: By induction on the size of the matrices $n \geq 2$.

Lemma 2: Let $n \geq 2$ be an integer and let $A = (a_0, a_1, \cdots, a_{n-1})^t$ and $V = (v_0, v_1, \cdots, v_{n-1})^t$ be two integral vectors. and let M be a square $n \times n$ matrix with integral entries, such that $V = M\,A$. If M is unimodular, i.e.: $\det(M) = \pm 1$, then $\gcd(a_0, a_1, \cdots, a_{n-1}) = \gcd(v_0, v_1, \cdots, v_{n-1})$.

Proof: Let $d = \gcd(a_0, a_1, \cdots, a_{n-1})$ and $\delta = \gcd(v_0, v_1, \cdots, v_{n-1})$. Since each v_i is a linear combination of the a_j's, then $d\,|\,v_i$ for all index i, so $d\,|\,\delta$. The matrix M^{-1} exists and has integral entries because $\det(M) = \pm 1$, so $A = M^{-1}V$. Similarly, each a_i is a linear combination of the v_i's, so $\delta\,|\,d$ and $\delta = d$.

Proposition 1 (Correctness) Let A_k be the vector obtained at the end of the k-th while loop iteration and let denote $\gcd(A) = \gcd(a_0, a_1, \cdots, a_{n-1})$. Then $\forall k \geq 1,\ \gcd(A_k) = \gcd(A_0)$.

Proof: We consider two cases $\alpha = a_I$ or $\alpha = |a_I - a_J| > 0$.
Case 1: Let $\alpha = a_I$ and the transformation $a'_i = a_i \bmod a_I$ for $i \neq I$ and $a'_I = a_I$. Then, by Lemma 1 and Lemma 2, we have $\gcd(a'_0, a'_1, \cdots, a'_{n-1}) = \gcd(a_0, a_1, \cdots, a_{n-1})$. In fact, if $i \neq I$, then $a'_i = a_i \bmod a_I$ and the matrix associated to this transformation has determinant ± 1.
Case 2: Let $\alpha = |a_I - a_J|$ for some indices I, J with $0 \leq I < J < n$ and let T be the transformation defined by $a'_i = a_i \bmod \alpha$ for $i \neq I$ and $a'_I = \alpha$. We split this transformation T in two parts $T = T_2 \circ T_1$. T_1 is the elementary transformation: $a'_I = \alpha = |a_I - a_J|$ and $a'_i = a_i$, for $i \neq I$. The determinant of the matrix associated to T_1 is obviously ± 1. The transformation T_2 is similar to the transformation described in Case 1, since α becomes one of the component of the vector $A = (a_i)$.

Theorem 1 *The $\Delta - GCD$ algorithm computes in parallel the GCD of n integers of $O(n)$ bits in length, in $O(n/\log n)$ time using $O(n^{2+\epsilon})$ processors in CRCW PRAM model, with $\epsilon > 0$.*

Proof: The algorithm is correct thanks to Proposition 1. We use CRCW PRAM model of computations and additions/subtractions can be done in $O(1)$ time with $O(n^{1+\epsilon})$ processors with table look-up [9, 10]. We only consider the $O(\log n)$ leading bits of the a_i's and use the pigeonhole principle to reduce α, at each while step, by $O(\log n)$ bits. It costs $O(1)$ parallel time with $O(n)$ processors (see details of the computation of α in **Appendix**). So the algorithm terminates after $O(n/\log n)$ iterations of the while loop and we have to prove that each iteration can be achieved in constant time with $O(n^{2+\epsilon})$ processors. Obviously, the most expensive operation is the computation of the remainders. Thus it remains to prove that the total cost of all the remainder computations can be achieved in $O(1)$ parallel time with the same number of processors, at each iteration. Let t_i be the time cost at iteration i, $1 \le i \le N$, with $N = O(n/\log n)$. So the total parallel time is $t(n) = \sum_{i=1}^{N} t_i$. Let k_i be the maximum bit length of all the quotients $q_j = \lfloor \frac{a_j}{\alpha} \rfloor$, with $\sum_{i=1}^{N} k_i \le n$. Then we have $t_i = O(\min\{\frac{k_i}{\log n}, \log n\})$. In fact, if $k_i \le \log n$ then $t_i = O(1)$. If $k_i > \log^2 n$, then one division between two n bits integers costs $t_i = O(\log n)$. Otherwise, if $\log n < k_i < \log^2 n$, then each quotient q_i has $k_i/\log n$ digits of $O(\log n)$ bits, so $t_i = O(\frac{k_i}{\log n})$ since arithmetic operations with $O(\log n)$ bits can be done in $O(1)$ parallel time with table look-up [9, 10]. The total number of processors is $n \times O(n^{1+\epsilon}) = O(n^{2+\epsilon})$ and the parallel time is then (recall that $\sum_{i=1}^{N} k_i \le n$)

$$t(n) = \sum_{i=1}^{N} \min(\{\frac{k_i}{\log n}, \log n\}) = \sum_{k_i < \log n} 1 + \sum_{\log n < k_i < \log^2 n} \frac{k_i}{\log n} + \sum_{k_i > \log^2 n} \log n = A + B + C.$$

We have $A \le \sum_{i=1}^{N} 1 = N = O(\frac{n}{\log n})$ and $B \le \sum_{k_i < \log^2 n}^{N} \frac{k_i}{\log n} = \frac{1}{\log n} \sum_{i=1} k_i \le \frac{n}{\log n}$.
$C = \sum_{k_i > \log^2 n}^{N} \log n = \log n \sum_{k_i > \log^2 n} 1$. Let P be the number of all the k_i's satisfying $k_i > \log^2 n$. Then $P \log^2 n \le k_1 + k_2 + \cdots + k_P < n$, so $P < \frac{n}{\log^2 n}$, hence $C \le \frac{n}{\log n}$ and $t(n) = O(\frac{n}{\log n})$. $\square$

Concluding remarks and directions for future research:

- Based on the same ideas, one may consider a fast sequential GCD algorithm for n integers of $O(n)$ bits in $O(n^2/\log n)$.
- An improvement can be done by dividing systematically α by all primes of $O(\log n)$ bits as described in a recent paper of Sorenson [11].
- Similarly, a least Significant Bit first approach (LSB) may be considered.
- Similarly to Blankinship algorithm [1], an extended GCD may be computed with identity matrix juxtaposed to vector A, as well as an upper bound of the multipliers [5] should be considered.
- We can improve the Rosser's algorithm [8, 7] to solve linear Diophantine equation. Instead of division by the first smallest non-zero integer of the columns, one may consider division by α. The resulting algorithm should be $O(\log n)$ faster.

References

[1] W. A. Blankinship, A new version of the Euclidean algorithm, Amer. Math. Monthly, 70 (1963), 742-745

[2] A. Borodin, J. von zur Gathen and J. Hopcroft, Fast parallel matrix and GCD computations *Information and Control* 52, 3, 1982, 241–256

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

[4] G. Cooperman, S. Feisel, J. von zur Gathen and G. Havas, GCD of many integers, *Lect. Notes in Comp. Sci., Springer-Verlag, Berlin,* 1627 (1999), 310–317

[5] G. Havas, S. Majewski, Extended gcd calculation, *Congressus Numerantium, 111,* 1627 (1998), 104-114

[6] R. M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines, Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity, J. van Leeuwen, Ed., New York, Elsevier (1991), 869-941

[7] M. Khorramizadeh and N. Mahdavi-Amiri, On solving linear Diophantine systems using generalized Rosser's algorithm, Bulletin of the Iranian Mathematical Society, 34, 2, (2008), 1-25

[8] J.B. Rosser, A note on the linear Diophantine equation, Amer. Math. Monthly, 48 (1941), 662-666

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

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

[11] J. Sorenson, A Randomized Sublinear Time Parallel GCD Algorithm for the EREW PRAM, Information Processing Letters, 110, 2010, 198-201

2 Appendix

Here are some details for the computation of (I, J), and then we compute $\alpha = |a_I - a_J|$.

Lemma A: Let $t_0, t_1, \cdots, t_{n-1}$ be n positive integers i.e.: $0 < t_i < n$, so that there exists at least two terms t_I and t_J such that $t_I = t_J$ with $0 \leq I, J < n$ and $I \neq J$. The determination of I and J can be achieved in $O(1)$ parallel time with $O(n)$ number of processors in CRCW PRAM model.

Proof: The main idea is described by the following function `pigeonhole` which, for a given array $(t[i])_i$, $0 \leq i < n$, with $0 < t_i < n$, returns a pair of integers (I, J), s.t.: $t_I = t_J$ with $0 \leq I, J < n$ and $I \neq J$.

```
Step 1: (Initialization)
        For $k = 0$ to $n - 1$ ParDo
            $s[k] = 0$;
        Endfor;

Step 2: (Detecting write concurrencies)
        For $i = 0$ to $n - 1$ ParDo
            $s[t[i]] = t[i]$;
            If (write concurrency and processor $i$ is chosen)   Index$[t[i]] = i$;
            Else Index$[t[i]] = -1$;
        Endfor;
```

/* We have $s[j] = j$ if j is a value reached by the array t, and $s[j] = 0$ otherwise. In case of write concurrency, we use the Arbitrary submodel of PRAM CRCW. Let i be the index of the processor allowed to write the common value $t[i] = k$. At this point we may have several write concurrency and Index integers as well. In the next step, we will choose arbitrarily one of them called I: */

```
Step 3: (Select one write concurrency: $I$)
    For $i = 0$ to $n - 1$ ParDo
        If ($Index[i] > 0$)   $I = Index[i]$ ;
        /* Use Arbitrary submodel of PRAM CRCW if write concurrency */
    Endfor ;

Step 4: (Determine $J$)
    For $j = 0$ to $n - 1$ ParDo
        If ($j \neq I$ and $t[j] = t[I]$)   $J = j$ ;
    Endfor ;
Return $(I, J)$.
```

Note finally that testing equality between two n bits integers can be done easily in $O(1)$ parallel time with $O(n)$ number of processors in Arbitrary CRCW PRAM model.
Example: With $n = 10$.

i	0	1	2	3	4	5	6	7	8	9
t	9	5	3	4	3	7	9	9	7	1
s	0	1	0	3	4	5	0	7	0	9
Index	-1	-1	-1	4	-1	-1	-1	5	-1	6

If $I = 6$ is selected, we obtain $J = 0$ and return $(6, 0)$. We have indeed $t[6] = t[0] = 9$.

Remark: Only $O(n^c)$ distinct integers of A, $0 < c < 1$, are needed for the function `pigeonhole`. In case there is not enough distinct integers in the set A, one can easily generate n new distinct integers in parallel: $a_k = k\beta \ \operatorname{cmod} \alpha$, for $1 \leq k < n$. In fact, if there are not all distinct, then there exists $0 \leq p, q < n$, $p \neq q$ such that $a_p = a_q$, and $p\beta \equiv q\beta \mod \alpha$, so $(p - q)\beta \equiv 0 \mod \alpha$ with $0 < |p - q| < n$. But in that case, $\gcd(\alpha, \beta)$ can be easily computed in parallel since $m\beta = k\alpha$ with $0 < m < n$ (only $O(\log n)$ Euclidean steps are needed).