Two Fast Parallel GCD Algorithms of Many Integers

Sidi Mohamed SEDJELMACI

Laboratoire d'Informatique Paris Nord, France.

ISSAC 2017, Kaiserslautern, 24-28 July 2017.

Motivations

• GCD of two integers : Used in CAS as a low operation, cryptography, etc.

- *Sequential:* $O(n \log^2 n \log\log n)$, Knuth (70)-Schönhage (71).
- *Parallel:* $O_\epsilon(n/\log n)$ time with $O(n^{1+\epsilon})$ processors, Chor-Goldreich (90), Sorenson (94) and Sedjelmaci (08). This problem is still open in parallel (P-complet or NC ?)

• GCD of many integers : polynomial computations, matrix computations, HNF and SNF.

- *Sequential:* Blan(63), Brad(70), Hav(98), Cop(99), etc.
- *Parallel:* Not addressed ?

Name	Year	Worst-case
Euclid	~ -300	$O(n^2)$
Lehmer	1938	$O(n^2)$
Stein	1961	$O(n^2)$
Knuth	1970	$O(\log^4 n M(n))$
Schönhage	1971	$O(\log n M(n))$
Brent-Kung	1983	$O(n^2)$
Jebelean-Weber	1993	$O(n^2)$
Sorenson	1994	$O(n^2/\log n)$
Stehlé et al.	2004	$O(\log n M(n))$
Möhler	2008	$O(\log n M(n))$

Table 1: Sequential GCD Algorithms for two integers.

Authors	Time	Nb. of proc.	Model
Brent-Kung, 1983	$O(n)$	$O(n)$	Systolic
Purdy, 1983	$O(n)$	$O(n)$	Systolic
Kannan et al., 1987	$O(n\,\frac{\log\log n}{\log n})$	$O(n^{2+\epsilon})$	PRAM-crcw
Adleman et al., rand., 1988	$O(\log^2 n)$	$\mathrm{e}^{O(\sqrt{n \log n})}$	PRAM-crcw
Chor-Goldreich, 1990	$O(n\,/\log n)$	$O(n^{1+\epsilon})$	PRAM-crcw
Sorenson, 1994	$O(n\,/\log n)$	$O(n^{1+\epsilon})$	PRAM-crcw
Sedjelmaci, 2008	$O(n\,/\log n)$	$O(n^{1+\epsilon})$	PRAM-crcw
Sorenson, rand., 2010	$O(n\,\frac{\log\log n}{\log n})$	$O(n^{6+\epsilon})$	PRAM-erew

Table 2: Parallel GCD Algorithms for two integers.

Our results:

• The GCD of n integers of $O(n)$ bits can be achieved in $O(n/\log n)$ time with $O(n^{2+\epsilon})$ processors in CRCW PRAM model in the worst case.

• The GCD of m integers of $O(n)$ bits can be achieved in $O(n/\log n)$ time with $O(mn^{1+\epsilon})$ processors in CRCW PRAM model, with $2 \leq m \leq n^{3/2}/\log n$.

• We suggest an extended GCD version for many integers and a algorithm to solve linear Diophantine equations.

• To our knowledge, it is the first time that we have this parallel performance for computating the GCD of many integers.

Notation:

A is a vector of n (or m) integers of $O(n)$ bits :

$A = (a_0, a_1, \cdots a_{n-1})$, with $a_i \geq 0$, $n \geq 4$

- An integer parameter k satisfying $\log k = \theta(\log n)$.
- $\gcd(A) = \gcd(a_0, a_1, \cdots a_{n-1})$.
- $\gcd(0, 0) = 0$.
- We use the PRAM (Parallel Random Access Machine) model of computation and CRCW PRAM (Concurrent Read Concurrent Write) sub-model.

Main idea for designing fast parallel GCD algorithm for many integers:

Find a small integer α

Repeat

$a_I := \alpha\,;$

$a_j := a_j \bmod \alpha\,;$ (in parallel, $\forall j \neq I$)

Until almost all the integers a_i are zeros.

How to find a small α **?**

Pigeonhole like techniques:

Lemma 1: Let $A = \{ a_1, a_2, \cdots, a_n \}$ be a set of n distinct positive integers, such that $n \geq 2$ and $a_n/n < a_1 < a_2 < \cdots < a_n$. Then

$$\exists\, i \in \{1, 2, \cdots, n-1\} \quad \text{s.t.:} \quad a_{i+1} - a_i < \frac{a_n}{n}\,.$$

A straightforward consequence is the following:

Corollary 1:

Let $A = \{ a_1, a_2, \cdots, a_n \}$ be a set of n distinct positive integers, with $n \geq 2$, then

$$\min\{a_k\,,\, |\, a_i - a_j \,| > 0\} \leq \frac{\max\{a_i\}}{n}\,, \quad \text{where} \;\; 1 \leq k, i, j \leq n\,.$$

We derive the following algorithm:

Input: A set $A = \{ a_0, a_1, \cdots, a_{n-1} \}$ of n integers of $O(n)$ bits, $n \geq 4$.

Output: $\gcd(a_0, a_1, \cdots, a_{n-1})$. $\alpha := a_0\,;\ I := 0\,;\ p := n\,;$

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 α */

    $\alpha :=\ \min\{\,|\,a_i - a_j\,| > 0\,\} = a_I - a_J\ \ ;\ \ a_I := \alpha\ \ ;$

  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 /* $\forall\, i\,,\ 0 \leq a_i \leq \alpha$ */

  If $(\forall\, i \neq I\,,\ a_i = 0\,)$ **Then Return** $\alpha\ \ ;$

  $p := np\,;$ /* p is $O(\log n)$ bits larger */

Endwhile

Return α.

The Δ-GCD Algorithm (Poster, ISSAC 2013)

Example (Δ-GCD): Let $A = (912672, 815430, 721161, 565701, 662592)$.

After 4 iterations, we obtain GCD(A) $= 3$. $n = 20$.

$$\begin{pmatrix} 912672 \\ 815430 \\ 721161 \\ 565701 \\ 662592 \\ \hline \alpha = 58569 \\ (I, J) = (2, 4) \end{pmatrix} \to \begin{pmatrix} 34137 \\ 54033 \\ 58569 \\ 38580 \\ 18333 \\ \hline 4443 \\ (0, 3) \end{pmatrix} \to \begin{pmatrix} 4443 \\ 717 \\ 810 \\ 3036 \\ 561 \\ \hline 93 \\ (1, 2) \end{pmatrix} \to \begin{pmatrix} 72 \\ 93 \\ 66 \\ 60 \\ 3 \\ \hline 3 \\ (4, -) \end{pmatrix} \to \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ \hline 3 \\ \texttt{STOP} \end{pmatrix}$$

Drawbacks of the pigeonhole technique

- The number of distinct integers is important. If there are only $O(\log n)$ distinct integers in A, then the pigeonhole technique will reduce the bit size of the integers by $O(\log \log n)$ bits and the number of iterations in the main while loop will be $O(n/\log \log n)$.

- What happens if $\alpha = 0$? For example, if $n = 8$ and $A = (255, 255, 193, 161, 129, 97, 65, 65)$.

There are only two pairs of integers that match in their 3 most significant bits, namely $(255, 255)$ and $(65, 65)$. Unfortunately, in both cases $\alpha = 0$.

- Comparing the $O(n^2)$ pairs of integers (a_i, a_j) to find a small $\alpha = a_i - a_j > 0$ in constant parallel time needs $O(n^3)$ processors.

Solution: Use other techniques

- Consider $O(\sqrt{n})$ integers and compute their differences $a_i - a_j$ to find $\alpha > 0$. There are $O(n)$ comparisons done in constant time with $O(n^{2+\epsilon})$ processors.

- In case it fails, use a Lehmer-like reduction (R_{ILE}, ISSAC'2001).

- In case all the R_{ILE} give zero, then `reduce` transformation will right-shift all the zeros of A and we continue the process with this new A.

The Lehmer-like reduction: R_{ILE} and Ext-R_{ILE}.

The R_{ILE} and `Ext-`R_{ILE} algorithms are described in Sed-ISSAC'01 and Sed-JDA'08. ILE stands for Improved Lehmer Euclid:

(1) R_{ILE} is defined by

Input: $u \geq v \geq 0\,,\ k = 2^m\,;\, m = \theta(\log n)$.

Output: $R_{ILE}(u, v) = |au + bv| < 2v/k$, with $1 \leq |a| \leq k$.

- Roughly speaking, $R_{ILE}(u, v)$ computes the continued fractions.

(2) : `Ext-`R_{ILE} is the extended version of R_{ILE} i.e.: we add the Bézout matrix M such that: $(\,0 \leq i, j \leq \lfloor \sqrt{n} \rfloor\,)$

$$M \times (a_i, a_j)^T = (R_i, R_j) \quad ; \quad R_j = R_{ILE}\,.$$

$$0 \leq R_j < R_i \text{ and } \gcd(R_i, R_j) = \gcd(a_i, a_j).$$

$$R_j < (2/k) \max\{a_i, a_j\}.$$

EXAMPLE: Let $u = 1\,759\,291$ and $v = 1\,349\,639$. Their binary representations are respectively:

$$\mathbf{11010110}\ 1100000111011_2 = 1\,759\,291$$
$$\mathbf{10100100}\ 1100000000111_2 = 1\,349\,639$$

We have $n = p = 21$. For $m = 3$, we obtain $\lambda = 2m + 2 = 8$, $u_1 = 214$ and $v_1 = 164$ (the leading bits of u_1 and v_1 are in bold). Using EEA with u_1 and v_1, we obtain in turn q, r, b and a $(r = au + bv)$:

q	r	a	b
	214	1	0
	164	0	1
1	50	1	-1
3	14	-3	4
3	8	10	-13

In our example, we obtain $a = -3$, $b = 4$,
$r = 14 < v_1/k = 164/8 = 20.50$ and

$$R_{ILE} = |-3u + 4v| = 120\,683 < v/8 = 168\,704.88$$

Properties of R_{ILE} and Ext-R_{ILE} :

- Parallel complexity : $O(n/\log n)_\epsilon$ time with $O(n^{1+\epsilon})$ processors on CRCW PRAM (ISSAC'01).
- It computes efficiently in parallel the Bézout coefficients with the same parallel performance (JDA'08).

High level description of Δ-2 GCD algorithm.

- **Test** 1: Is there a small enough $a_i > 0$ so that we can consider it straightforwardly as an α ?
- **Test** 2: Does the pigeonhole algorithm provide an $\alpha > 0$?
- **Test** 3: Use a new transformation R based on continued fractions (Sed-ISSAC'01) and test if $R > 0$?

If Test 3 fails, i.e.: $R_j(a_i, a_j) = 0$ for all (a_i, a_j), with $i, j \leq \sqrt{n}$, then $(R_i, R_j) = (R_i, 0)$ and $(a_i, a_j) \longleftarrow (0, R_i)$.

A new transformation called `reduce` right-shifts all the zeroes in A. We reduce by half the number of $O(\sqrt{n})$ positive integers considered (the other half of integers are all zeroes). Moreover, it could be iterated at most $O(\sqrt{n})$ times since, at each step, we add $O(\sqrt{n})$ new zeros in the vector A.

Δ-2 GCD algorithm,:

Input: A vector $A = (a_0, a_1, \cdots, a_{n-1})$, $n \geq 4$ and $\max\{a_i\} < 2^n$.

Output: $\gcd(a_0, a_1, \cdots, a_{n-1})$.

```
(α, I) := (a_0, 0) ; p := n ; N := ⌊√n⌋ ;
While (α > 1) Do
    For (i = 0) to (n − 1) ParDo
        If (0 < a_i ≤ 2^n/p) then {(α, I) := (a_i, i) ; S := 1 } ;
        else S := 0 ; /* No small a_i */
    Endfor
    If (S = 0) then (α, I) := pigeonhole(A, N) ;
    If (I = −1) then R := 0 ; /* The pigeonhole fails */
       For (i, j = 0) to (N − 1) ParDo x_ij := R_ILE(a_i, a_j) ;
         If (x_ij > 0) then { (α, I) := (x_ij, i) ; R := 1 ; a_I := x_ij }
             /* We can divide all the a_i's by α = x_ij */
           Endif
        Endfor
```

If $(R = 0)$ /* $\forall\, i, j$, $R_{ILE}(a_i, a_j) = 0$ */

then $A := \texttt{reduce}(A, N)$;

Endif

Endif

If $(I \geq 0)$ **then** $A := \texttt{remainder}(A, \alpha, I)$;

If $(\exists\, a_k \neq 0$ s.t.: $\forall\, i \neq k \;\Rightarrow\; a_i = 0)$ **then Return** a_k ;

$p := np$; /* p is $O(\log n)$ bits larger */

Endwhile

Return α.

The `remainder` procedure just divides all the components of A by α and consider their remainders. It proceeds as follows:

Input: $A = (a_0, \cdots, a_{n-1})$, with $n \geq 4$, $0 \leq I \leq n-1$, and $\alpha > 0$.

Output: $A' = (a'_0, \cdots, a'_{n-1})$, s.t.: $a'_i = a_i \bmod \alpha$ for all $i \neq I$ and $a'_I = a_I = \alpha$.

$a_I = \alpha$;
For $(i = 0)$ **to** $(n-1)$ **ParDo**
  If $(i \neq I)$ **then** $a_i := a_i \bmod \alpha$;
Endfor
Return A.

The **`pigeonhole`** algorithm is based on Corollary 1 with the first $O(\sqrt{n})$ integers of A, namely $(a_0, a_1, \cdots, a_{N-1})$, with $N = \lfloor \sqrt{n} \rfloor$.

The algorithm returns a pair (α, I) such that $\alpha = a_I - a_J > 0$ is small enough or, in the case there is no such pair, it returns $(\alpha, I) = (a_0, -1)$.

- Unlike the pigeonhole principle, the transformation `reduce` will guarantee the termination and the parallel performance of the Δ2-GCD algorithm. In fact, it could be iterated at most $O(\sqrt{n})$ times since, at each step, we add $O(\sqrt{n})$ new zeros in the vector A.

- An example for `reduce`: Let $n = 10$ and $N = \lfloor \sqrt{n} \rfloor = 3$. Let $A = (350, 150, 260, 390, 330, 550, 343, 411, 503, 739)$, with $\max\{A\} < 2^n = 1024$. We only consider the first $6 = 2N$ integers of A, i.e.: $(350, 150, 260, 390, 330, 550)$. We obtain for $(a_0, a_1) = (350, 150)$, the Bézout matrix $M = \begin{pmatrix} 1 & -2 \\ -3 & 7 \end{pmatrix}$ and $M \times (350, 150) = (R_0, R_1) = (50, 0)$. Similarly $(R_2, R_3) = (130, 0)$, $(R_4, R_5) = (110, 0)$ and `reduce` returns: $A = (50, 130, 110, 343, 411, 503, 739, 0, 0, 0)$.

- So $\texttt{reduce}(A, 3)$ gives rise to 3 zeroes in A.

BA GCD algorithm (Best Approximation), no `pigeonhole`:

Input: $A = (a_0, a_1, \cdots, a_{n-1})$, $a_i > 0$, $n \geq 4$, $\max\{a_i\} < 2^n$.

Output: $\gcd(a_0, a_1, \cdots, a_{n-1})$.

$(\alpha, I) := (a_0, 0)$; $p := n$; $N := \lfloor \sqrt{n} \rfloor$;

While $(\alpha > 1)$ **Do**

  For $(i = 0)$ **to** $(n-1)$ **ParDo**

    If $(0 < a_i \leq 2^n/p)$ **then** $(\alpha, I) := (a_i, i)$ **else** $I := -1$;

  Endfor

  If $(I = -1)$ **then** /* No small a_i */

    $R := 0$;

    For $(i, j = 0)$ **to** $(N-1)$ **ParDo**

      $x_{ij} := R_{ILE}(a_i, a_j)$;

      If $(x_{ij} > 0)$ **then** $\{(\alpha, I) := (x_{ij}, i)\,; a_I := x_{ij};\ R = 1\}$

        /* We can divide all the a_i's by x_{ij} */

      Endif

    Endfor

If $(R = 0)$ **then** $A :=$ `reduce`(A, N) ;

/* $R = 0$ means $\forall\, i, j\,,\ \ R_{ILE}(a_i, a_j) = 0$ */

Endif

If $(I \geq 0)$ **then** $A :=$ `remainder`(A, α, I) ;

/* We divide all the a_i's but a_I by $\alpha > 0$ */

If $(\exists\, a_k \neq 0$ s.t.: $\forall\, i \neq k \Rightarrow a_i = 0)$ **then**

Return a_k ;

$p := np$;

Endwhile

Return α.

Correctness of Δ-2 and BA GCD algorithms

- **Main idea**: Unimodular matrices preserve GCD, i.e.:

$$\det(M) = \pm 1 \implies \gcd(M \times A) = \gcd(A).$$

- The matrices associated with `pigeonhole`, `remainder` and Ext-R_{ILE} are all unimodular.

$$\begin{vmatrix} 1 & 0 & 0 & \cdots & 0 \\ -q_1 & 1 & 0 & \cdots & 0 \\ -q_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ -q_{n-1} & 0 & 0 & \cdots & 1 \end{vmatrix}.$$

Matrix associated with $\alpha = a_0 < \max\{A\}/n$.

$$\begin{vmatrix} 1 & -1 & 0 & \cdots & 0 \\ -q_1 & q_1+1 & 0 & \cdots & 0 \\ -q_2 & q_2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ -q_{n-1} & q_{n-1} & 0 & \cdots & 1 \end{vmatrix}.$$

Matrix associated with $\alpha = a_0 - a_1 < \max\{A\}/n$:

$$a'_0 = a_0 - a_1\,;$$
$$a'_i = a_i - q_i\alpha = -q_i a_0 + q_i a_1 + a_i\,;$$

$$\begin{vmatrix} s_0 & t_0 & 0 & 0 & 0 & \cdots & 0 \\ s_1 & t_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & s_2 & t_2 & 0 & \cdots & 0 \\ 0 & 0 & s_3 & t_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & s_{n-2} & t_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & s_{n-1} & t_{n-1} \end{vmatrix} .$$

Matrix associated with $(a'_{2i}, a'_{2i+1}) = \text{Ext-}R_{ILE}(a_{2i}, a_{2i+1})$:

$$(a'_0, a'_1) = (s_0 a_0 + t_0 a_1, s_1 a_0 + t_1 a_1)\,;$$
$$(a'_2, a'_3) = (s_2 a_2 + t_2 a_3, s_3 a_2 + t_3 a_3)\,;$$
$$\cdots \quad \cdots \quad \cdots \quad \cdots$$

Complexity analysis of Δ-2 and BA algorithms

Let S = number of iterations in the while loop.

At each iteration i, $1 \leq i \leq S$, we note

- $A^{(i)} = (a_0^{(i)}, \cdots, a_j^{(i)}, \cdots, a_{n-1}^{(i)})$.
- k_i = the largest bit size of the quotients $\lfloor a_j^{(i)}/\alpha_i \rfloor$.

Then the key points are:

- $S = O(n/\log n)$.
- $\sum_{i=1}^{S} k_i = O(n)$.

- The proof is given in details in the paper.

Proposition : (Complexity of `remainder`)

- Let t_i be the parallel time cost at iteration i.
- Let k_i be the the largest bit size of the quotients $\lfloor a_j^{(i)}/\alpha_i \rfloor$.

Then the time complexity of **`remainder`** is :

Total time :	$\sum_{i=1}^{S} t_i = O(n/\log n)$
Nb. of processors :	$O(n^{2+\epsilon})$.

Ideas of the proof :

- Use look-up tables (arithmetics with big numbers)
- Split the sum in three parts w.r.t. the bit size of k_i :

$$k_i \leq \log n \text{ or } \log n \leq k_i \leq \log^2 n \text{ or } k_i \geq \log^2 n \,.$$

Theorem : The $\Delta 2$-GCD and BA algorithms compute in parallel the GCD of m integers of $O(n)$ bits in length, in $O(n/\log n)$ time using $O(m\,n^{1+\epsilon})$ processors in CRCW PRAM model, for any $\epsilon > 0$ and m, such that: $2 \leq m \leq n^{3/2}/\log n$.

Proof (sketch) :

• Ext-R_{ILE}, `pigeonhole` and `remainder` can be done with this parallel bound. (They all deal with the bit size of integers)

• Since `reduce` (deals with the number of non zero integers) adds $O(\sqrt{n})$ zeroes in A and A has initially m integers, so the number of calls is at most $O(m/\sqrt{n})$. So

$$m/\sqrt{n} \leq n^{3/2}/(\sqrt{n}\log n) = n/\log n.$$

CONCLUSION

- We generalize the parallel performance of computing the GCD of two integers (CHG'90, SOR'94, SED'01) to the case of many integers.
- The parallel time for computing the GCD of m integers of $O(n)$ bits can be achieved in $O(n / \log n)$ parallel time with $O(m\, n^{1+\epsilon})$ processors.
- The parallel time does not depend on the number m of integers if it satisfies $2 \leq m \leq n^{3/2} / \log n$.
- We suggest an extended GCD version for many integers as well as an algorithm to solve linear Diophantine equations.
- To our knowledge, it is the first time that we find deterministic algorithms which compute the GCD of many integers with this parallel performance and polynomial work.

LATEST NEWS !!

No **pigeonhole** in BA-GCD algorithm $\Longrightarrow$ no comparison , we can consider all the m integers (not only $\sqrt{n}$)

$$(a_{2i}, a_{2i+1}) \longrightarrow (R_{2i}, R_{2i+1})\,, \;\; 0 \le i \le \lfloor (m-1)/2 \rfloor .$$

- There are at most $O(\log m)$ calls for `reduce` (A is halved each time).
- $\log m = O(n/\log n) \Longrightarrow m = 2^{O(n/\log n)}$.

Theorem (Modified BA-GCD algorithm) : There exist a parallel algorithm computing the GCD of m integers of $O(n)$ bits in $O(n/\log n)$ time with $O(mn^{1+\epsilon})$ processors. This result is valid for any m in the range : $2 \le m \le 2^{O(n/\log n)}$.