The Accelerated Euclidean Algorithm

Sidi Mohamed Sedjelmaci

Abstract

We propose a new GCD algorithm called Accelerated Euclidean Algorithm, or AEA for short, which matches the $O(n \log^2 n \log \log n)$ time complexity of Schönhage algorithm for $n$-bit inputs. This new GCD algorithm is designed for both integers and polynomials. We only focus our study to the integer case, the polynomial case is currently addressed ([3]).

1 Introduction

The algorithm is based on a half-gcd like procedure, but unlike Schönhage’s algorithm, it is iterative and therefore avoids the penalizing calls of the repetitive recursive procedures. The new half-gcd procedure reduces the size the integers at least a half word-memory bits per iteration only in single precision. By a dynamic updating process, we obtain the same recurrence and the same time performance as in the Schönhage approach.

Throughout, the following notation is used. $W$ is a word memory, i.e.: $W = 16, 32$ or $64$. Let $u \geq v > 2$ be positive integers where $u$ has $n$ bits with $n \geq 32$. Given a non-negative integer $x \in \mathbb{N}$, $\ell(x)$ represents the number of its significant bits, not counting leading zeros, i.e.: $\ell(x) = \lceil \log_2(x+1) \rceil$. For the sake of readability integers $U$ and $V$ will be represented as a concatenation of $l$ sets of $W$ bits integers (except for the last set which may be shorter), i.e.: $U = \sum_{i=0}^{l-1} 2^{iW} U_i$ and $V = \sum_{i=0}^{l-1} 2^{iW} V_i$, then we write symbolically $U = U_1 \cdot U_2 \ldots \cdot U_l$ and $V = V_1 \cdot V_2 \ldots \cdot V_l$.

We use specific vectors which represent interval subsets from of $U$ and $V$, by:

\[ X[i..j] = \begin{pmatrix} U_i \cdot U_{i+1} \cdot \ldots \cdot U_j \\ V_i \cdot V_{i+1} \cdot \ldots \cdot V_j \end{pmatrix} \]

\[ X[i] = \begin{pmatrix} U_i \\ V_i \end{pmatrix} \]

for $1 \leq i < j \leq l$.

Let $M(x)$ be the cost of a multiplication of two $x$-bit integers. The function $M(x)$ depends on the algorithm used to carry out the multiplications. The fast Schönhage-Strassen algorithm ([6]) performs these multiplications in $M(x) = O(n \log n \log \log n)$.

2 AEA: The Accelerated Euclidean Algorithm

The following algorithm is based on two main ideas:

- The computations are done in a Most Significant digit First (MSF) way.
- Update by multiplying, with the current matrix, ONLY twice the number of leading bits we have already chopped, NOT the others leading bits.
Algorithm **AEA**.

**Input:** $u \geq v > 2$ with $u \geq 8W$; \hspace{1cm} $n = \ell(u)$;  
**Output:** a $2 \times 2$ matrix $M$ and $(U', V')$ such that \hspace{1cm} $M \times (U, V) = (U', V')$ and $\ell(V') \leq \ell(U) - 2^{\lfloor \log_2(n/W) \rfloor - 1} W$. 

**Begin** \hspace{1cm} \small{\text{\textbf{1.}} \hspace{0.5cm} $(U, V) \leftarrow (u, v); \hspace{0.5cm} s \leftarrow \lfloor \log_2(n/W) \rfloor$; \textbf{2.} if $\ell(V) \leq \lceil \ell(U)/2 \rceil + 1$ return $I$; \textbf{3.} if $\ell(V) > \lceil \ell(U)/2 \rceil + 1$ \textbf{4.} \textbf{For} $i = 1$ to $2^s - 1$ \textbf{5.} \hspace{0.5cm} if $U_i = 0$ or $V_i \neq 0$ (Regular case) \textbf{6.} \hspace{1cm} $h \leftarrow 0$; \textbf{7.} \hspace{1cm} if $(i$ odd) $L_0 \leftarrow ILE(X[i..i + 1]); \hspace{1cm} \text{update}_L(i, h)$; \textbf{8.} \hspace{1cm} else $R_0 \leftarrow ILE(X[i..i + 1]); \hspace{1cm} \text{update}_R(i, h)$; \textbf{9.} \hspace{1cm} $x \leftarrow i/2; \hspace{1cm} h \leftarrow h + 1;$ \textbf{10.} \hspace{1cm} While $(x$ even) \textbf{11.} \hspace{1cm} $R_h \leftarrow R_{h-1} \times L_{h-1}; \hspace{1cm} \text{update}_R(i, h)$; \textbf{12.} \hspace{1cm} $x \leftarrow x/2; \hspace{1cm} h \leftarrow h + 1;$ \textbf{13.} \hspace{1cm} \textbf{Endwhile}$ \hspace{1cm} \textbf{14.} \hspace{1cm} $L_h \leftarrow R_{h-1} \times L_{h-1}; \hspace{1cm} \text{update}_L(i, h)$; \textbf{15.} \hspace{1cm} \textbf{Endelse}$ \hspace{1cm} \textbf{16.} \textbf{EndFor}$ \hspace{1cm} \textbf{17.} \hspace{0.5cm} \text{Irregular}(i) (U_i \neq 0 \hspace{0.5cm} \text{and} \hspace{0.5cm} V_i = 0)$; \textbf{18.} \textbf{EndFor}$ \hspace{1cm} \textbf{19.} \hspace{0.5cm} \text{Return} \hspace{0.5cm} L_h \hspace{0.5cm} \text{and} \hspace{0.5cm} (U, V)$; \hspace{1cm} \textbf{End}$

The algorithm **ILE** (borrowed from [7]) runs the extended Euclidean algorithm and stops when the remainder has roughly the half size of the inputs.

Algorithm **ILE**.

**Input:** $u_0 \geq u_1 \geq 0$ \hspace{1cm} $n = \ell(u_0); \hspace{1cm} p = \ell(u_1)$;  
**Output:** a $2 \times 2$ matrix $M$ and $(u_{i-1}, u_i)$ such that $M \times (u_0, u_1) = (u_{i-1}, u_i)$ and $\ell(u_i) \leq \frac{1}{2} \ell(u_0)$. 

**Begin** \hspace{1cm} \small{\text{\textbf{1.}} \hspace{0.5cm} $n = \ell(u_0); \hspace{0.5cm} p = \ell(u_1);$ \textbf{2.} if $p < \lceil n/2 \rceil + 1$ return $M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$; \textbf{3.} if $p \geq \lceil n/2 \rceil + 1$ \textbf{4.} \hspace{0.5cm} $m = p - \lceil n/2 \rceil - 1;$ \textbf{5.} \hspace{0.5cm} Apply Extended Euclid Algorithm until $|a_i| \leq 2^m < |a_{i+1}|;$ \textbf{6.} \hspace{0.5cm} \text{return} \hspace{0.5cm} M = \begin{pmatrix} a_{i-1} & b_{i-1} \\ a_i & b_i \end{pmatrix}$ and $(u_{i-1}, u_i)$. \hspace{1cm} \textbf{End}$.
The functions $\text{update}_L$ or $\text{update}_R$ update not all the bits of the operands, but only the next useful small vectors, in order to get the next needed matrix. The irregular case is when $U_i \neq 0$ and $V_i = 0$, i.e.: one or many components are all equal to zero. Roughly speaking, we perform an euclidean division in order to make an efficient reduction and continue our process. The aim is to preserve, at most, the general scheme of our process. The basic idea is to use full updated vectors, i.e.: vectors updated with all the previous matrices $L_h$.

3 An Example

In order to carry out single-precision computations we take $W = 20$. Recall that the notation $\left( \begin{array}{c} A \\ B \end{array} \right) = \left( \begin{array}{c} a \cdot c \\ b \cdot d \end{array} \right)$ means $\left( \begin{array}{c} A \\ B \end{array} \right) = \left( \begin{array}{c} a \\ b \end{array} \right) \times 2^W + \left( \begin{array}{c} c \\ d \end{array} \right)$, where $A, B, a, b, c$ and $d$ are integers (c.f. notation in Section 1).

Let $\left( \begin{array}{c} u \\ v \end{array} \right) = \left( \begin{array}{c} 922375420941707599307587 \\ 674819 \end{array} \right)$, then, with our notation, we obtain $\left( \begin{array}{c} U \\ V \end{array} \right) = \left( \begin{array}{c} 879645 \\ 674819 \end{array} \right) \times 2^{20} + \left( \begin{array}{c} 785421 \\ 299843 \end{array} \right) = \left( \begin{array}{c} 879645 \cdot 785421 \\ 674819 \cdot 299843 \end{array} \right)$.

We have $U_1 = 879645$ and $V_1 = 674819$ then $\ell(U_1) = \ell(V_1) = 20$, $n = p = 20$ and $m_1 = p - 1 - \lceil \frac{n}{2} \rceil = 9$. Thus, in order to compute $ILE(U_1, V_1)$, we must stop the Extended Euclid Algorithm at $|a_i| \leq 2^9$. We obtain the matrix ($|a_i| < 2^9 = 512$)

$N_1 = ILE(U_1, V_1) = \left( \begin{array}{c} 369 \\ -425 \end{array} \right)$

then:

$N_1 \left( \begin{array}{c} U \\ V \end{array} \right) = N_1 \left( \begin{array}{c} U_1 2^{20} + U_2 \\ V_1 2^{20} + V_2 \end{array} \right) = N_1 \left( \begin{array}{c} U_1 \\ V_1 \end{array} \right) 2^{20} + N_1 \left( \begin{array}{c} U_2 \\ V_2 \end{array} \right)$.

So $N_1 \times \left( \begin{array}{c} U_1 2^{20} + U_2 \\ V_1 2^{20} + V_2 \end{array} \right) = \left( \begin{array}{c} 1066 \\ 601 \end{array} \right) \times 2^{20} + \left( \begin{array}{c} 138 \\ -160 \end{array} \right) \times 2^{20} + \left( \begin{array}{c} 892378 \\ 81257 \end{array} \right) = \left( \begin{array}{c} 1204 \\ 441 \end{array} \right) \times 2^{20} + \left( \begin{array}{c} 892378 \\ 81257 \end{array} \right)$.

hence

$\left( \begin{array}{c} U_1 \\ V_1 \end{array} \right) = \left( \begin{array}{c} 1204 \\ 441 \end{array} \right)$ and $\left( \begin{array}{c} U_2 \\ V_2 \end{array} \right) = \left( \begin{array}{c} 892378 \\ 81257 \end{array} \right)$.

Now we can apply $ILE$ to the new integers (less than 32 bits) $U = 1263377882$ and $V = 462503273$. We have $\ell(U) = 31$ and $\ell(V) = 29$. We obtain $m = 8$ and the second matrix (in 32-bits single precision)

$N_2 = ILE(U, V) = \left( \begin{array}{c} -41 \\ 231 \end{array} \right)$

and $M = N_2 \times N_1 = \left( \begin{array}{c} -41 \\ 231 \end{array} \right) \times \left( \begin{array}{c} 369 \\ -425 \end{array} \right) = \left( \begin{array}{c} -62729 \\ 353414 \end{array} \right)$.

Moreover, at this level, we may consider that $U_1$ and $V_1$ are eliminated (chopped), even if $U_2$ has some extra bits, and that $U_2$ and $V_2$ are updated as follows:
\[
\begin{pmatrix} U_2 \\ V_2 \end{pmatrix} = \begin{pmatrix} 1873414 \\ 725479 \end{pmatrix} = \begin{pmatrix} 1 \cdot 824838 \\ 0 \cdot 725479 \end{pmatrix}; \quad \ell(U_2) = 21; \quad \ell(V_2) = 20.
\]

On the other hand, it is easy to check that the final matrix \( M = \begin{pmatrix} a_{t-1} & b_{t-1} \\ a_t & b_t \end{pmatrix} \) satisfies
\[
M \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} -62729 & 81769 \\ 353414 & -460685 \end{pmatrix} \times \begin{pmatrix} 922375420941 \\ 707599307587 \end{pmatrix} = \begin{pmatrix} 1873414 \\ 725479 \end{pmatrix}.
\]

Now, if \( u \) and \( v \) were larger in size, namely:
\[
\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 879645 \cdot 785421 \cdot u_3 \cdot u_4 \ldots \cdot u_k \\ 674819 \cdot 299843 \cdot v_3 \cdot v_4 \ldots \cdot v_k \end{pmatrix},
\]
then we have to continue the half-gcd process. Since \( W \) bits have been already chopped, we have to update only the double, i.e.: multiply the next 2\( W \) leading bits of \( U \) and \( V \) by \( M \), namely perform:
\[
\begin{pmatrix} U_3 \cdot U_4 \\ V_3 \cdot V_4 \end{pmatrix} \leftarrow M \begin{pmatrix} U_3 \cdot U_4 \\ V_3 \cdot V_4 \end{pmatrix},
\]
and disregard all the other bits of \( U \) and \( V \).

Then do the same process as before with \( \begin{pmatrix} U_2 \cdot U_3 \\ V_2 \cdot V_3 \end{pmatrix} \) instead of \( \begin{pmatrix} U_1 \cdot U_2 \\ V_1 \cdot V_2 \end{pmatrix} \) to chop the vector \( \begin{pmatrix} U_2 \\ V_2 \end{pmatrix} \), and so on, repeating the process till we reach the middle of the size of \( U \).

4 An Example

Let us consider two consecutive Fibonacci numbers \((u, v) = (F_{59}, F_{58})\), i.e.: \( \begin{pmatrix} F_{59} \\ F_{58} \end{pmatrix} = \begin{pmatrix} 956722026041 \\ 591286729879 \end{pmatrix} = \begin{pmatrix} 956722 \cdot 026041 \\ 591286 \cdot 729879 \end{pmatrix} \).

First, we must compute \( mMAX \) which gives \( 2^mMAX \), the maximum size of the output matrix \( L_1 \)
\[
mMAX = \ell(v) - \left\lfloor \frac{\ell(u)}{2} \right\rfloor - 1 = 19.
\]

Let \((U_1, V_1) = (956722, 591286)\) then \( \ell(U_1) = \ell(V_1) = 20, n = p = 20 \) and \( m = p - 1 - \lceil \frac{n}{2} \rceil = 9 \). We run the Extended Euclid algorithm and stops at \(|a_i| \leq 2^9\). We obtain the matrix \( L_0 \) and the remainders \((r_1, r_2)\):
\[
L_0 = LIE(U_1, V_1) = \begin{pmatrix} -144 & 233 \\ 233 & -377 \end{pmatrix} \quad \text{and} \quad (r_1, r_2) = (1670, 1404).
\]

Hence \( L_0 \begin{pmatrix} U \\ V \end{pmatrix} = L_0 \begin{pmatrix} U_1 10^9 + U_2 \\ V_1 10^9 + V_2 \end{pmatrix} = L_0 \begin{pmatrix} U_1 \\ V_1 \end{pmatrix} 10^9 + L_0 \begin{pmatrix} U_2 \\ V_2 \end{pmatrix}. \)

By \text{update}(1, 0) \) we compute \( L_0 \begin{pmatrix} U_2 \\ V_2 \end{pmatrix} \), so
Now, if \( u \) and \( v \) were larger in size, namely:

\[
\begin{pmatrix}
 u \\
 v
\end{pmatrix} = \begin{pmatrix}
 879645 & 785421 & u_3 & u_4 & \ldots & u_k \\
 674819 & 299843 & v_3 & v_4 & \ldots & v_k
\end{pmatrix}
\]

then we have to continue the half-gcd process. Since \( W \) bits have been already chopped, we have to update only the double, i.e.: multiply the next \( 2W \) leading bits of \( U \) and \( V \) by \( M \), namely perform:
\[
\begin{pmatrix}
U_3 \cdot U_4 \\
V_3 \cdot V_4
\end{pmatrix}
\leftarrow M \times \begin{pmatrix}
U_3 \cdot U_4 \\
V_3 \cdot V_4
\end{pmatrix},
\] and disregard all the other bits of \(U\) and \(V\).

Then do the same process as before with \(\begin{pmatrix}
U_2 \cdot U_3 \\
V_2 \cdot V_3
\end{pmatrix}\) instead of \(\begin{pmatrix}
U_1 \cdot U_2 \\
V_1 \cdot V_2
\end{pmatrix}\) to chop the vector \(\begin{pmatrix}
U_2 \\
V_2
\end{pmatrix}\), and so on, repeating the process till we reach the middle of the size of \(U\).

Here we stress that this is the main difference between our approach and the other Sorenson’s like algorithms, where all the bits of \(U\) and \(V\) are updated by multiplying them with the matrix \(M\). In our approach, we only update the double of what we have already chopped.

5 Remarks

Unlike the recursive versions of GCD algorithms (([1],[4])), our aim is not to balance the computations on each leaf of the binary tree computations, but to make full of single precision everytime, computing therefore the maximum of quotients in single precision. This lead to different computations each step in the algorithm AEA and the other recursive GCD algorithms. Moreover the fundamental difference between AEA and Schönhage’s approach is that AEA deals straightforward with the most significant leading bits first (MSF computation). Consequently, we can stop the algorithm AEA at any time and still obtain the leading bits of the result. Thus AEA is a strong MSF algorithm and can be considered for ”on line” arithmetics, where all the basic operations can be carried out simultaneously as soon as only the needed bits are available. This new algorithm should be an alternative to the Schönhage GCD algorithm. On the other hand, the derived GCD algorithm deals with many applications where long euclidean divisions are needed. We have identified and started to study many applications such as subresultants and Cauchy index computation, Padé-approximates or LLL-algorithms.

6 References

[1] Gathen, J. von zur, Gerhard, G. Modern Computer Algebra. In Cambridge University Press (1999).
[2] D.H. Lehmer. Euclid’s algorithm for large numbers, American Math. Monthly, 45, 1938, 227-233.
[3] M.F. Roy and S.M. Sedjelmaci. The Polynomial Accelerated Euclidean Algorithm, Subresultants and Cauchy index, work in progress, 2004.
[4] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklugen, Acta Informatica, 1, 1971, 139-144.
[5] A. Schönhage, A.,F.,W. Grotefeld and E. Vetter. Fast Algorithms, A Multi-tape Turing Machine Implementation, BI-Wissenschaftsverlag, Mannheim, Leipzig, Zürich, 1994.
[6] A. Schönhage, V. Strassen. Schnelle Multiplication grosser Zahlen Computing 7, 1971, 281-292.
[7] 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.

Sidi Mohamed Sedjelmaci
LIPN CNRS UMR 7030,
Université Paris-Nord
Av. J.-B. Clément, 93430 Villetaneuse, France.
E-mail: sms@lipn.univ-paris13.fr