The Accelerated Euclidean Algorithm

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

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 the 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]. 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 and, by a dynamic updating process, we obtain the same recurrence and the same time performance as in the Schönhage approach.
In order to explain how our algorithm works, the following notation is used. W is a word memory, i.e.: 32 or 64. Let $u \geq v > 2$ be positive integers where u has a n bits with $n \geq 32$. Given a non-negative integer $x \in N$, $\ell(x)$ is the number of its significant bits, $\ell(x) = \lceil \log_2(x+1) \rceil$. Integers U and V are represented as a concatenation of l sets of W bits integers (except for the last set which may be shorter), with $l = \lceil n/W \rceil$, i.e.: if $U = \sum_{i=0}^{l-1} 2^{iW}\ U_{l-i}$ with $U_1 \neq 0$ and $V = \sum_{i=0}^{l-1} 2^{iW}\ V_{l-i}$, then we write symbolically

$$U = U_1 \bullet U_2 \ldots \bullet U_l \quad ; \quad V = V_1 \bullet V_2 \ldots \bullet V_l.$$

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

$$X[i..j] = \begin{pmatrix} U_i \bullet U_{i+1} \bullet \cdots \bullet U_j \\ V_i \bullet V_{i+1} \bullet \cdots \bullet V_j \end{pmatrix} \ ; \ X[i] = \begin{pmatrix} U_i \\ V_i \end{pmatrix} \ ; \text{ for } 1 \leq i < j \leq l.$$

Let $M(x)$ be the cost of a multiplication of two x-bit integers. The fast Schönhage-Strassen algorithm [6] performs these multiplications in $M(x) = O(n \log n \log\log n)$. The algorithm AEA is based on two main ideas:

- The computations are done in a Most Significant digit First (MSF) way.
- We update by multiplying, with the current matrix, `ONLY` twice the number of leading bits we have already chopped, `NOT` the others leading bits. The multiplication with the other leading bits is postponed.

The algorithm starts straightforward from the leading bits. Let r be a given integer parameter, $r \geq 1$. Let $t(r)$ denotes the time cost for reducing $2r$-bit inputs to their half, i.e.: r bits, by running a Lehmer-like GCD algorithm ([2, 7]). Suppose we want to reduce the $2r$ leading bits from the n-bits inputs say u and v. Observe that we only need to consider the $4r$ leading bits, not any other bit. We suggest to do it as follows.

Begin

- **Step 1:** First, we consider the $2r$ leading bits and reduce them to their half, it costs $t(r)$. We obtain also a matrix N_1 of length r, i.e.: $\ell(N_1) = r$.
- **Step 2:** Update the $2r$ next leading bits by multiplying them with the matrix N_1. A possible carry must be added in $O(r)$ time. It costs $O(M(r) + O(r)) = O(M(r))$ in time. We observe that similarly, we just have to repeat the same process as before by considering the new $2r$ next leading bits, hence the next steps:
- **Step 3:** Consider the $2r$ new leading bits and reduce them to their half, it costs $t(r)$. We obtain then a new matrix N_2 of length r.
- **Step 4:** Update the new r next leading bits with the $O(r)$-bit length matrix N_2 which costs $O(M(r))$ in time.
- **Step 5:** Compute the matrix $M = N_2 \times N_1$, this matrix will be used to reduce the next $4r$ leading to their half. It costs $O(M(r))$ in time.

End.
Hence the relation $t(2r) = 2t(r) + O(M(n))$ for any $r > 1$ and the total time complexity is $t(n) = O(\log n\ M(n)) = O(n \log^2 n \log\log n)$.
The four first steps are summarized in Figure 1. Each line represents the size of the two inputs u and v at each step, starting from the top. The leading bits we want to halve are under a circular line. After each reduction, the updated bits are represented by black segments. It is worth to notice that the fundamental idea is that we do not multiply the matrix N_1 or N_2 with all the bits of the inputs, but only with **few** of them, just for computing the next first quotients. The multiplication with the other bits is postponed. The iterative algorithm AEA, described below, follows these ideas.

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

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

End

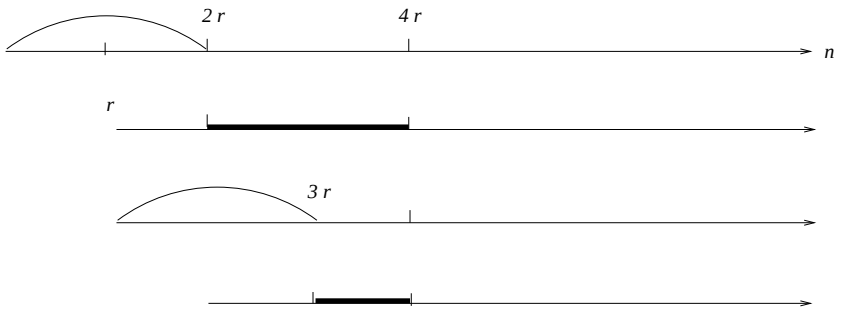

Figure 1: The new half-gcd principle.

The algorithm `ILE` ([7]) runs the extended Euclidean algorithm and stops when the remainder has roughly the half size of the inputs. L_h and R_h stand for left and right matrix and are computed at the depth level h of the binary tree computations of AEA. The functions $\texttt{update}_L$ or $\texttt{update}_R$ update only the next few leading bits of the current vectors, in order to compute the next matrices. When the function `Irregular` is called ($V_i = 0$), more updated bits are computed in order to perform a correct euclidean division and continue the process.

Unlike the recursive versions of GCD algorithms (see [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 every time, computing therefore the maximum of quotients in single precision. This lead to different computations. Moreover the fundamental difference between AEA and Schönhage's approach is that AEA starts 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. Finally, the derived GCD algorithm deals with many applications where long euclidean divisions scheme is needed. We have identified and started to study many applications such as subresultants and Cauchy index computation, *Pade*-approximates or LLL-algorithms.

References

[1] J. von zur Gathen and J. Gerhard. 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 Multitape Turing Machine Implementation, BI-Wissenschaftsverlag, Mannheim, Leipzig, Zürich, 1994.

[6] A. Schönhage, V. Strassen. Schnelle Multiplication grosser Zalen 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.