View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Cleveland-Marshall College of Law

Cleveland State University
EngagedScholarship@CSU

Mathematics Faculty Publications

Mathematics Department

2-1-2005

A Modular Integer GCD Algorithm

Kenneth Weber
Mount Union College

Vilmar Trevisan
Universidade Federal do Rio Grande do Sul

Luiz Felipe Martins
Cleveland State University, L.MARTINS@csuohio.edu

Follow this and additional works at: https://engagedscholarship.csuohio.edu/scimath_facpub

Part of the Mathematics Commons
How does access to this work benefit you? Let us know!

Repository Citation

Weber, Kenneth; Trevisan, Vilmar; and Martins, Luiz Felipe, "A Modular Integer GCD Algorithm" (2005). *Mathematics Faculty Publications*. 156.
https://engagedscholarship.csuohio.edu/scimath_facpub/156

This Article is brought to you for free and open access by the Mathematics Department at EngagedScholarship@CSU. It has been accepted for inclusion in Mathematics Faculty Publications by an authorized administrator of EngagedScholarship@CSU. For more information, please contact library.es@csuohio.edu.

A modular integer GCD algorithm

Kenneth Weber, Vilmar Trevisan, Luiz Felipe Martins

Introduction

By using a *modular representation* for integers [3, Section 4.3.3] we can efficiently perform certain arithmetic operations on parallel processors having a large number of processing elements. Each processor simultaneously performs the addition, subtraction or multiplication operation modulo a separate prime. However, since comparison and general division are expensive when the operands are in modular representation (a conversion of the operands to another representation must be performed), the modular representation has been unattractive for use in integer greatest common divisor (GCD) algorithms.

Chapter 6 of [4] describes an algorithm to compute the greatest common divisor of two n-bit integers using modular representation for intermediate values. As far as we know, it was the first integer GCD algorithm to use a modular representation. This algorithm avoids comparisons altogether, and relies only on special case integer divisions that can be performed with the operands still in modular representation.

The reduction step of this algorithm is the same as used in Sorenson's right-shift k-ary GCD algorithm [5], and as the reduction used when the intermediate values are nearly the same size in the accelerated GCD algorithm, independently discovered by Jebelean and Weber [1,2]. If U and V represent the intermediate values whose GCD is equal to that of the original input pair, then this reduction replaces U by $(aU + bV)/m$, where $|a|, |b| < \sqrt{m}$ and m is one of the moduli used to represent the intermediate values U and V. This reduces the size of the problem by roughly $\log_2 m/2$ bits. However, spurious factors are introduced that must be eliminated *after* converting the result back to standard representation.

The new algorithm is based on the following reduction step. Suppose that $\mathcal{P}$ is a set of (odd) prime numbers relatively prime to V; then the main loop of the new algorithm uses the reduction step $U \leftarrow (U - bV)/p$, where $p \in \mathcal{P}$, and $b \equiv U \cdot V^{-1} \pmod{p}$ is chosen to lie in the interval $(-p/2, p/2)$. The modulus p is chosen so that $|b|$ is minimal among all moduli in $\mathcal{P}$. This is very similar to the reduction used by Jebelean in the EDGCD algorithm [6], and the reduction used by the accelerated algorithm when the difference in sizes of U and V is large, except that in these cases a single fixed modulus p^t is used, and b is computed in the range $[0, p^t - 1]$. Although this reduction is not as efficient as Sorenson's k-ary reduction is when U and V are nearly the same size, "cleanup" of spurious factors is *not* necessary. This allows the GCD operation to be used in combination with other arithmetic operations to perform a larger calculation totally in modular representation before it is necessary to convert back to standard representation.

An algebraic common CRCW PRAM, as described in [7], is a parallel random access machine on which each processor can access w-bit integers in a global shared memory, and compute w-bit addition, subtraction, comparison, multiplication, quotient and remainder of integers, all in unit time. We define a w-bit *modular* common CRCW PRAM to be an algebraic common CRCW PRAM with the additional capability of each processor to compute w-bit modular inverses in unit time.

In the worst case, the new algorithm runs on a w-bit modular common CRCW PRAM in $O(n + 2^{w/2})$ time using $O(n + 2^{w/2})$ processors. At worst the main loop will never iterate more than $n + 2$ times; experimental data taken from execution of a sequential implementation of the algorithm suggests that in the average case the main loop will execute a

little more than $2n/w$ times—as long as the moduli are w-bit primes, and there are at least $n+2^{w/2}$ moduli and processors. A heuristic model of the distribution of the b values allows for an average case analysis that closely matches the experimental data. For enough moduli, this analysis gives an average case running time of $O(n/\log n + 2^{w/2})$ on $O(n+2^{w/2})$ processors, which translates to $O(n/\log n)$ time and $O(n \log n \log\log\log n)$ processors on a bit common CRCW PRAM, when $w = \Theta(\log n)$. Thus the heuristic average case analysis matches the fastest algorithms presently known—those of Chor and Goldreich [8], Sorenson [5], and Sedjelmaci [9], whose worst-case analyses all give $O(n/\log n)$ time on $O(n^{1+\varepsilon})$ processors, where ε is any positive constant. We believe this new algorithm has the potential to be better suited for massively parallel machines than any of these algorithms, especially when embedded inside other computations using a modular representation for intermediate values.

Throughout the sequel we use a mod b, where a is any integer and b is a positive odd integer, to represent the *symmetrical* modular representation [10, Problem 10, p. 277]; that is, the unique integer $c \in (-b/2, b/2)$ such that $c \equiv a \pmod{b}$. The expression $u/v \bmod m$ is, of course, another way to write $uv^{-1} \bmod m$, where v^{-1} is the multiplicative inverse of v in $\mathbf{Z}/m\mathbf{Z}$. In certain parts of this paper, the base of the log function is significant; we use $\ln x = \log_e x$ and $\lg x = \log_2 x$, as in [10], in order to eliminate ambiguity. An integer t is a *w-bit integer* iff $2^{w-1} \leqslant t < 2^w$. Finally, we use $\pi(x)$ to represent the number of primes less than or equal to the real number x, and p_k to represent the kth prime, where 2 is counted as the first prime.

In the next section the basic algorithm is presented and proven to correctly compute the gcd. An upper bound for the number of iterations of the main loop is derived, and the average number of iterations is estimated via the heuristic model mentioned previously. The full algorithm is presented in detail in the section following. The two sections after that are devoted to an analysis of the complexity of the full algorithm: the first presents properties of the modular common CRCW PRAM, and the second uses those properties in the actual analysis of the algorithm. Results from experiments with a sequential implementation are then provided. We conclude with some miscellaneous remarks and ideas for future work.

Basic algorithm

A basic version of the algorithm, suppressing the modular representation for clarity, is displayed in Fig. 1. A set of prime moduli is first selected according to criteria which guarantee that the main loop will execute no more than $n+2$ times. Then the algorithm enters the main loop; in each iteration the reduction step (line 8) reduces the number of bits in U and V, using one of the $U/V \bmod p$ values that is smallest in absolute value. The division by p in the reduction step is exact, and can be performed in modular arithmetic, as long as the divisor is relatively prime to all moduli; thus p must be eliminated from $\mathcal{Q}$ before the division occurs (line 7).

Our first concern is to show that the GCD of the original inputs is preserved throughout the main loop.

Input: Positive integers U and V, with $U \geqslant V$
Output: $\gcd(U, V)$

```
 1 n ← ⌊lg U⌋ + 1
 2 Q ← a set of at least n + 2 primes with π(min Q) > max{n, 9}
 3 Repeat
 4    P ← {q ∈ Q | V mod q ≠ 0}
 5    p ← an element of P for which |U/V mod p| is minimal
 6    b ← U/V mod p
 7    Q ← Q − {p}
 8    [U, V] ← [V, (U − bV)/p]
 9 Until V = 0
10 Return |U|
```

Fig. 1. The basic algorithm.

Lemma 1. *Suppose that p divides $U - bV$, and let $T = (U - bV)/p$. If p and V are relatively prime, then* $\gcd(U, V) = \gcd(T, V)$.

Proof. Let $g = \gcd(U, V)$ and $g' = \gcd(T, V)$. Since g' divides V and $Tp = U - bV$, it is clear that g' also divides U. Hence g' divides g. To show that g divides g', note that g divides $U - bV = Tp$. Taken with the fact that $\gcd(g, p)$ divides $\gcd(V, p) = 1$, we have that g divides T in addition to V. $\square$

We now show that the values of U and V get smaller and smaller, and determine how many iterations of the main loop are needed to make $V = 0$. We shall use the following notation throughout the remainder of the paper. Let U_0, V_0 and $\mathcal{Q}_0$ be the initial values of U, V and $\mathcal{Q}$; $U_i, V_i, b_i, \mathcal{P}_i$ and $\mathcal{Q}_i$ the values of $U, V, b, \mathcal{P}$, and $\mathcal{Q}$ at the end of the ith iteration of the loop; and $m = \min \mathcal{Q}_0$ and $M = \max \mathcal{Q}_0$.

Theorem 2. *If $V_i \neq 0$, then $\lg |V_i| < n - i + 2$, and the loop will finish in no more than $n + 2$ iterations.*

Proof. First note that the upper bound on the number of iterations may be derived from the inequality, since it implies that $|V_i| = 1$ after $i = n + 1$ iterations, forcing $V_{i+1} = 0$. To establish the inequality, one can prove by induction that

$$|V_i| < \frac{U_0}{2^i} \prod_{p \in \mathcal{Q}_0 - \mathcal{Q}_i} \left(1 + \frac{3}{p}\right).$$

From this, and the fact that $\lg(1 + x) \leqslant x \lg e$ for nonnegative x, we get $\lg |V_i| < n - i + 3 \lg e S_i$, where $S_i = \sum_{p \in \mathcal{Q}_0 - \mathcal{Q}_i} 1/p$. Let $\bar{p}_i = p_{\pi(m)-1+i}$ and $\hat{p} = p_{2\pi(m)}$; since $\pi(m) > n$, we have $\bar{p}_i < \hat{p}$ and

$$S_i \leqslant \sum_{p \leqslant \bar{p}_i} \frac{1}{p} - \sum_{p \leqslant m} \frac{1}{p} + \frac{1}{m} < \sum_{p \leqslant \hat{p}} \frac{1}{p} - \sum_{p \leqslant m} \frac{1}{p} + \frac{1}{m}$$

for all $1 \leqslant i \leqslant n + 1$, where the index p in the sums takes on only prime values. One can show that $3 \lg e S_i < 2$ for all iterations of the loop and for all primes $m \geqslant 29 = p_{10}$ by using this last inequality, together with the following three inequalities from Rosser and

Schoenfeld [11, ineqs. 3.6, 3.13, 3.17 and 3.20]: $p_k < k(\ln k + \ln\ln k)$ for $k \geqslant 6$, $\pi(x) < 1.25506x/\ln x$ for $x > 1$, and

$$\ln\ln x + B - \frac{1}{2\ln^2 x} < \sum_{p \leqslant x} \frac{1}{p} < \ln\ln x + B + \frac{1}{\ln^2 x}$$

for $x > 1$, where B is a real constant. □

The next lemma gives a lower bound on the size of $\mathcal{P}_i$. Note that it also shows that the choice of $\mathcal{Q}_0$ guarantees that $\mathcal{P}_i$ is never empty.

Lemma 3. *There will be at least $\|\mathcal{Q}_0\| - \lfloor i + (n+3-i)/\lg m - 1 \rfloor \geqslant \|\mathcal{Q}_0\| - n - 1$ elements in $\mathcal{P}_i$ in any iteration of the main loop.*

Proof. Let $q_1, \ldots, q_k$ be the primes in $\mathcal{Q}_{i-1}$ that divide $V = V_{i-1}$. Clearly $|V_{i-1}| \geqslant q_1 \cdots q_k \geqslant m^k$. By Theorem 2, we have that $k \lg m \leqslant \lg |V_{i-1}| < n - (i-1) + 2$. The result follows from this and the facts that $\|\mathcal{Q}_{i-1}\| = \|\mathcal{Q}_0\| - i + 1$ and, by the previous theorem, $i \leqslant n + 2$ for any iteration of the loop. □

The remainder of this section is dedicated to a heuristic analysis of the average number of iterations required by the main loop. Experience with the experimental implementation of the algorithm, discussed in Section 6, suggests that, when using w-bit moduli, as long as there are at least $\lceil 2^{w/2} + n \rceil$ moduli, and as long as U_{i-1} and V_{i-1} each has more than $2w$ bits, the values $|U_{i-1}/V_{i-1} \bmod p|$ for $p \in \mathcal{P}_i$ are fairly uniformly distributed over the interval $[0, p/2)$. This suggests that we may treat them as uniformly distributed, mutually independent random variables in order to obtain a first approximation to the expected value $\mathrm{E}[|b_i|] = \mathrm{E}[\min_{p \in \mathcal{P}_i}\{|U_{i-1}/V_{i-1} \bmod p|\}]$. Based on this assumption, and the fact that the minimum of ν independent and identically distributed random samples from $[0, 1)$ has expected value $1/(\nu + 1)$ (see [12, p. 182], for example), we get $\mathrm{E}[|b_i|] \leqslant 0.5M/(\|P_i\| + 1) < 0.5M/\|P_i\|$. We can now use this to provide an upper bound on the average number of iterations of the main loop, provided we make one more simplifying assumption: that b_i and V_{i-1} are independent as well. We will refer to the complete set of assumptions as our *heuristic model.*

Lemma 4. *Based on our heuristic model, if the set of moduli $\mathcal{Q}$ is chosen so that $\|Q_0\| \geqslant n + 1 + 0.5m^{-1/2}M$, then an upper bound on the average number of iterations of the main loop is $\lceil 2n/\lg(m/\phi^2) \rceil + 1$, where $\phi = (1 + \sqrt{5})/2$.*

Proof. By combining Lemma 3 with the assumptions of this lemma, we get $\mathrm{E}[|b_i|] < m^{1/2}$. Using this as an upper bound for $|b_i|$, and the assumption that b_i and V_{i-1} are independent, one can show by induction that $|V_i| < U_0 \phi^i m^{-i/2}$ when $|b_i|$ achieves its expected value. Then $\lg |V_i| < 1$ when $i \geqslant 2n/\lg(m/\phi^2)$. At most one more iteration may be needed to reduce V to zero. □

When $\lceil 2^{w/2} + n \rceil$ w-bit primes are used, then the bound above is approximately $2n/w$. This approximation closely matches the results obtained from our experimental implemen-

tation; see Fig. 5. If we use a set of $2^{17} = 131{,}072$ moduli between 2^{31} and 2^{32}, we can easily handle input sizes up to $2^{16} = 65{,}536$ bits, and still expect the average number of iterations of the main loop to be roughly $n/16$. These observations are formalized by the following theorem, which can be obtained from the previous lemma.

Theorem 5. *The heuristic model predicts that the number of iterations in the average case is $O(n/w)$ when $w \geqslant 4$, $\|\mathcal{Q}_0\| \geqslant n + 2^{w/2}$ and $2^w > M \geqslant m > 2^{w-1}$.*

Modular algorithm

In Fig. 2 we give the full version of the algorithm, in which we finally include the details of the modular representation. Step MGCD1 chooses a set of w-bit primes for the moduli. Note that the requirements in line 2 of Fig. 1 are met; we have $\|\mathcal{Q}\| = \lceil 2^{w/2} + n \rceil \geqslant n + 2$ and $\pi(m) > \pi(2^{w-1}) > \pi(2^w) - \pi(2^{w-1}) \geqslant \max\{n, 9\}$, since $\pi(2x) < 2\pi(x)$ for $x \geqslant 3$ (from [13], quoting [14]). Step MGCD2 converts the input integers U and V into modular representation. The construct "For all $i \in \mathcal{I}$ do ..." indicates parallel execution. Set notation is used for indexing here because it does not implicitly specify an order of execution. Step MGCD3 is the actual reduction loop, resulting in a modular representation of the greatest common divisor of U and V. Step MGCD4 computes a *balanced mixed-radix*

Input: Positive integers U and V, with $U \geqslant V$
Output: $\gcd(U, V)$

MGCD1: [Find suitable moduli]
1 $n \leftarrow \lfloor \lg U \rfloor + 1$
2 $w \leftarrow$ an integer satisfying $\pi(2^w) - \pi(2^{w-1}) \geqslant \max\{\lceil 2^{w/2} + n \rceil, 9\}$
3 $\mathcal{Q} \leftarrow$ the set of $\lceil 2^{w/2} + n \rceil$ largest primes less than 2^w

MGCD2: [Convert to modular representation]
1 For all $q \in \mathcal{Q}$ do $[u_q, v_q] \leftarrow [U \bmod q, V \bmod q]$

MGCD3: [Reduction loop]
1 Repeat
2 $\quad \mathcal{P} \leftarrow \{q \in \mathcal{Q} \mid V \bmod q \neq 0\}$
3 $\quad p \leftarrow$ an element of $\mathcal{P}$ for which $|u_p/v_p \bmod p|$ is minimal.
4 $\quad b \leftarrow u_p/v_p \bmod p$
5 $\quad \mathcal{Q} \leftarrow \mathcal{Q} - \{p\}$
6 $\quad$ For all $q \in \mathcal{Q}$ do $[u_q, v_q] \leftarrow [v_q, (u_q - bv_q)/p \bmod q]$
7 Until $\forall q \in \mathcal{Q}, v_q = 0$

MGCD4: [Return standard representation]
1 $G \leftarrow 0$
2 Repeat
3 $\quad p \leftarrow$ an element of $\mathcal{Q}$
4 $\quad \mathcal{Q} \leftarrow \mathcal{Q} - \{p\}$
5 $\quad G \leftarrow u_p + pG$
6 $\quad$ For all $q \in \mathcal{Q}$ do $u_q \leftarrow (u_q - u_p)/p \bmod q$
7 Until $\forall q \in \mathcal{Q}, u_q = 0$
8 Return $|G|$

Fig. 2. Modular GCD algorithm.

representation [10, ex. 10 soln., p. 586], using modular arithmetic, and simultaneously produces a standard representation of the result from the mixed-radix representation.

There is a rather large supply of w-bit primes. Using the approximation $\pi(x) \approx \int_2^x \mathrm{d}t/\ln t$ [10, pp. 366–367], we see that there are roughly 9.8×10^7 32-bit primes and 2.1×10^{17} 64-bit primes. In addition, w-bit primes *and* smaller may be used, if necessary, although the details for such a modification are left to the interested reader. Thus, for all practical purposes, the number of available primes seems sufficient to utilize even the large numbers of processors current research suggests will be available in future systems (see [15], for example).

We close this section by showing that there will be enough primes left in $\mathcal{Q}$ to reconstruct the standard representation of the result.

Theorem 6. *Let N be the number of iterations of the loop in Step MGCD3. Then* $2|U_N| < \prod_{q \in \mathcal{Q}_N} q$.

Proof. Note that $U_N = V_{N-1} \neq 0$. By Theorem 2, $\lg|V_{N-1}| < n - N + 3$, so

$$\prod_{q \in \mathcal{Q}_N} q \geqslant 2^{(w-1)\lceil 2^{w/2}+n-N \rceil} > 2^{n-N+4} > 2|V_{N-1}|. \qquad \square$$

Modular common CRCW PWAM

The full algorithm presented in the previous section seems naturally suited for implementation on the algebraic common concurrent read, concurrent write, parallel random access machine (CRCW PRAM) used in [7], augmented by a constant-time modular inverse operation. In this section we show that the minimum needed on line 3 of MGCD3 can be computed in $O(1)$ time on such a computational model, using only a small number of processors. We also compute the additional costs for emulating on a bit common CRCW PRAM an algorithm analyzed for this model.

Define a w-bit *modular* common CRCW PRAM to be an algebraic common CRCW PRAM, in which each processor is capable of performing w-bit memory accesses and w-bit integer arithmetic in unit time, and is also able to compute the inverse of a w-bit integer modulo another w-bit integer in unit time.

Lemma 7. *The maximum and minimum of a set $\mathcal{A}$ of integers in the range* $[0, 2^w - 1]$ *can be found in $O(1)$ time on a w-bit modular common CRCW PRAM with* $\max\{\|\mathcal{A}\|, 2^{\lfloor w/4 \rfloor}(2^{\lfloor w/4 \rfloor} - 1)\}$ *processors.*

Proof. Define $\mathcal{A}_0 = \mathcal{A}$ and $\mathcal{A}_i = \{x \in \mathcal{A} \mid f_i(x) = \max_{y \in \mathcal{A}_{i-1}}\{f_i(y)\}\}$ for $1 \leqslant i \leqslant 4$, where $f_i(x) = \lfloor x/2^{(4-i)\lfloor w/4 \rfloor} \rfloor$; then $\max \mathcal{A} = \max \mathcal{A}_4$. Since a w-bit modular PRAM compares two w-bit integers in unit time, it can be thought of as a comparison PRAM for integers in the range $[0, 2^w - 1]$, allowing us to use an algorithm given in [16, p. 888] to find the maximum of a set of 2^k values in $O(\log\log 2^k - \log\log(2^k - 1)) + O(1) = O(1)$ time on a comparison-PRAM with $2^k(2^k - 1)$ processors. Then $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$, and $\max \mathcal{A}_4 =$

$\max \mathcal{A}$ can all be computed with a total cost of $O(1)$ time on $\max\{\|\mathcal{A}\|, 2^{\lfloor w/4 \rfloor}(2^{\lfloor w/4 \rfloor} - 1)\}$ processors. Since $\min \mathcal{A} = 2^w - 1 - \max_{a \in \mathcal{A}}\{2^w - 1 - a\}$, we get the same cost for the minimum. □

Lemma 8. *A w-bit modular common CRCW PRAM can be emulated on a bit common CRCW PRAM, using $O(w \log\log w)$ bit processors to emulate each modular processor, and auxiliary tables of size $O(w2^{w/2})$ bits which take $O(2^{w/4})$ time and $O(2^{w/4} w \log\log w)$ bit processors to create.*

Proof. According to the literature [5], additions, subtractions, moves and comparisons can all be done in $O(1)$ time using $O(w \log\log w)$ bit processors. Quotient and multiplication tables for integers in the range $[0, 2^{\lfloor w/4 \rfloor} - 1]$ can be built using only repeated addition; each table has $O(2^{w/2})$ entries, requiring $O(w2^{w/2})$ bits of global memory and setup costs of $O(2^{w/4})$ time and $O(2^{w/4} w \log\log w)$ processors. After preparation of these tables, w-bit multiplications, quotients and remainders can be computed in $O(1)$ time using $O(w \log\log w)$ processors, by using naive arithmetic algorithms on five digit operands in base $2^{\lfloor w/4 \rfloor}$. Finally, modular inverses can be computed in $O(1)$ time and $O(w \log\log w)$ bit processors by using an extended GCD algorithm constructed by applying the comments of Sorenson [5, Section 7] to the EDGCD algorithm [6] with radix $\beta = 2^{\lfloor w/4 \rfloor}$. This would require construction of a table that would hold, for each integer $x \in [1, 2^{\lfloor w/4 \rfloor} - 1]$, the values $x^{-1} \bmod 2^{\lfloor w/4 \rfloor}$ and the largest integer i such that 2^i divides x. The costs to construct the table is $O(w^2)$ time and $O(2^{w/4} w \log\log w)$ processors; $O(w2^{w/4})$ bits of memory are needed to store the table. □

Performance analysis

In this section we provide an asymptotic analysis of the performance of the full algorithm. We first analyze the algorithm on the w-bit modular CRCW PRAM, then extend the analysis to the bit-common CRCW PRAM model, as used in [5]. First we compute the total cost of the algorithm on a w-bit modular common CRCW PRAM.

Theorem 9. *On a w-bit modular common CRCW PRAM the modular GCD algorithm takes $O(n + 2^{w/2})$ time and $O(n + 2^{w/2})$ processors. The heuristic model for* $\mathrm{E}[|b_i|]$ *gives an average running time of $O(n/w + 2^{w/2})$ for the same number of processors.*

Proof. The total worst-case cost is obtained by taking the maximum of the costs of the major steps, which are cataloged below.

Step MGCD1 requires $O(2^{w/2})$ time and $O(2^{w/2})$ processors

This can be done by emulating a sieve algorithm for an EREW PRAM, described by Sorenson and Parberry [7], with no increase in time or number of processors [16, Section 3].

Step MGCD2 requires $O(n/w)$ time and $O(n + 2^{w/2})$ processors
A naive algorithm can be used to divide an n-bit integer by a w-bit integer in $O(n/w)$ time, so this step can be computed by first calculating one half of the $2\lceil 2^{w/2} + n\rceil$ divisions simultaneously, then the other half.

Step MGCD3 requires $O(n)$ time and $n + O(2^{w/2})$ processors
Each line in the main loop takes $O(1)$ time on $\lceil 2^{w/2} + n\rceil$ processors. We get the cost for line 3 by applying Lemma 7 twice; once to compute the minimum value, and once to choose a prime for which the minimum is achieved. Note also that the universal quantifier in line 7 can be calculated as the negation of an existential quantifier, which can be computed as an inclusive-OR operation in $O(1)$ time on an $\lceil 2^{w/2} + n\rceil$ processor common CRCW PRAM [16, p. 896]. By Theorem 2, the number of iterations in the worst case in the main loop in Step MGCD3 is bounded by $n + 2$, thus giving the time and number of processors listed.

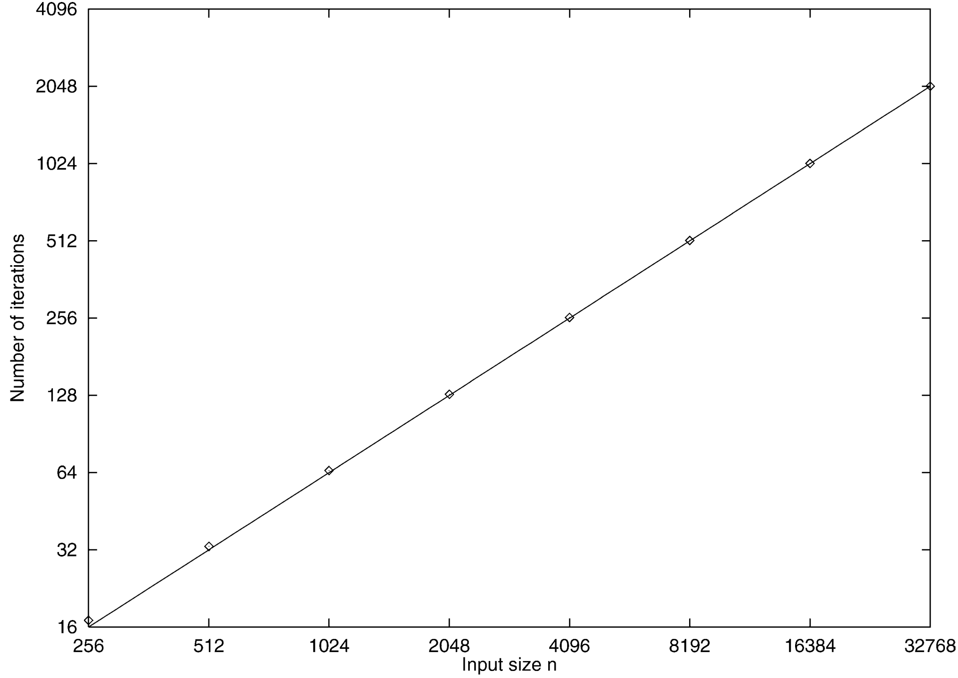
Step MGCD4 requires $O(n/w)$ time and $O(n + 2^{w/2})$ processors
Line 5 causes $|G| \leqslant U_0$ to grow by no more than w bits each iteration of the loop, so the main loop can execute no more than n/w times. The multiplication on this line can be performed in $O(1)$ time by $O(n)$ processors, by adapting a technique for bit common CRCW PRAMs given in [8]. The remainder of the lines in the loop, as well as the lines before and after the loop, can be performed in constant time, as in Step MGCD3. Note particularly that the choice of an element of $\mathcal{Q}$ in line 3 may be done in $O(1)$ time using $O(n + 2^{w/2})$ processors, according to Lemma 7.
The average cost is obtained by using Theorem 5 to get $O(n/w)$ iterations of the main loop on average. □

By combining Lemma 8 with the previous theorem, and noting that the requirements placed on w by Step MGCD1 are met when $w = \Theta(\log n)$, we get the following.

Theorem 10. *On a bit common CRCW PRAM, the modular integer GCD algorithm takes $O(n+2^{w/2})$ time and $O((n+2^{w/2})w \log\log w)$ processors. The heuristic model for* $\mathrm{E}[|b_i|]$ *gives an average running time of $O(n/w + 2^{w/2})$ for the same number of processors. By choosing $w = \Theta(\log n)$, the costs become $O(n)$ for worst-case time, $O(n/\log n)$ for average-case time, and $O(n \log n \log\log\log n)$ for the number of processors.*

Experimental implementation

The following tables and graphs summarize the results obtained from experimentation with a sequential implementation of the modular algorithm. The experimental implementation allows one to choose w and $\|\mathcal{Q}_0\|$ without requiring that $\|\mathcal{Q}_0\| \geqslant n + 2^{w/2}$. The table in Fig. 3 registers the average number of iterations required as a function of input size n and the average size, in bits, of the minimum of the b values. For each size n the algorithm was run with ten different pairs of input values, pseudorandomly chosen in the range $[0, 2^n)$, using $\|\mathcal{Q}_0\| = 2^{17}$ consecutive prime moduli. The largest modulus was $2^{32} - 5$;

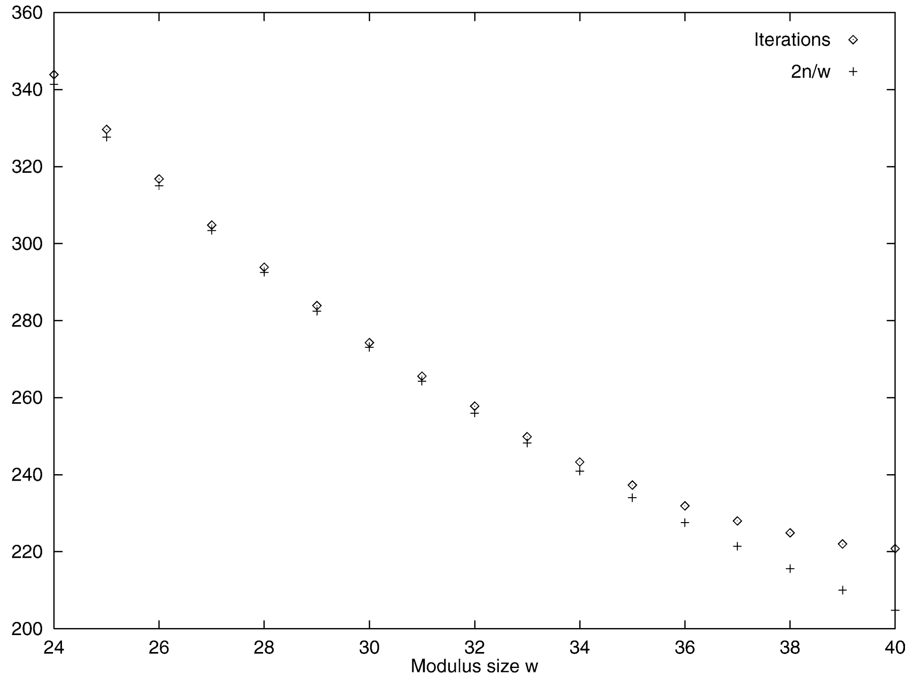


Input size n	Average number of iterations	Average b size
2^8	17.0	13.74
2^9	33.0	13.92
2^{10}	65.2	13.75
2^{11}	129.2	13.73
2^{12}	257.6	13.63
2^{13}	513.9	13.70
2^{14}	1026.2	13.68
2^{15}	2051.9	13.73

Fig. 3. Comparing $2n/w$ to number of iterations as n varies ($w = 32$, $\|\mathcal{Q}_0\| = 2^{17}$).

the smallest modulus was $2^{32} - 2{,}910{,}755$. Notice that the average b sizes are well below the upper bound of $0.5M/\|\mathcal{P}_i\| \approx 15$ bits predicted by the heuristic model (see discussion preceding Lemma 4). This strongly suggests that the heuristic model overestimates the average b sizes. The graph in Fig. 3 compares the number of iterations the algorithm performs (diamonds) with $2n/w$ (solid line). Notice how closely the number of iterations tracks $2n/w$ when $\|\mathcal{Q}_0\| \geqslant n + 2^{w/2}$. This fits nicely with intuition, since one would expect each iteration to shorten one of U and V by w bits.

The table in Fig. 4 reveals the number of iterations the modular algorithm performs with n fixed at 2^{12} and $\|\mathcal{Q}_0\|$ at 2^{17}. We plot $\lg(w)$ vs the number of iterations and $\lg(w)$ vs $2n/w$ in the graph in Fig. 4. As w grows larger, the gap between the number of iterations

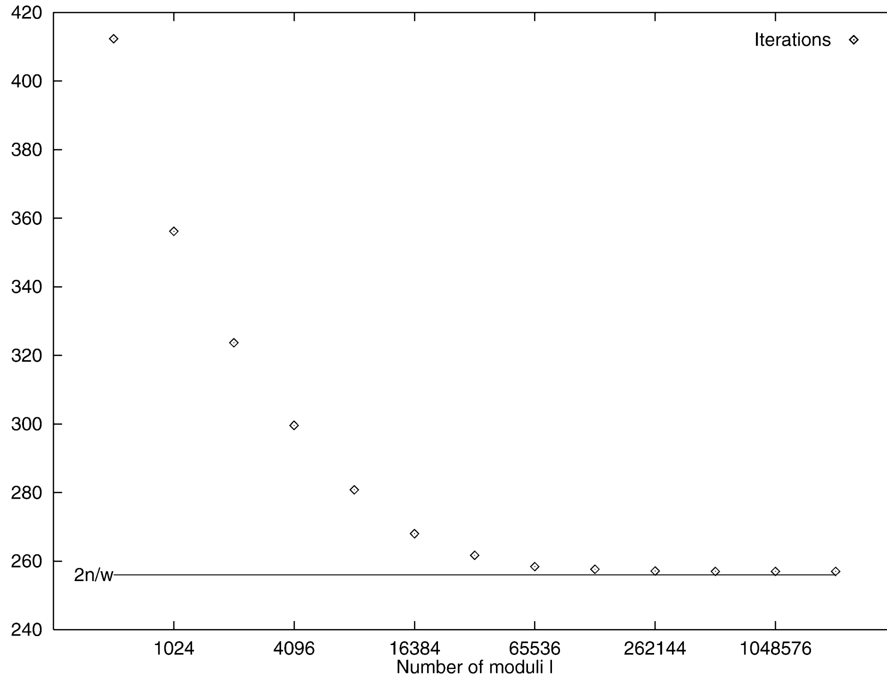


Size of moduli w	Average number of iterations	Average b size
24	343.9	5.68
25	329.7	6.64
26	316.8	7.69
27	304.8	8.68
28	293.9	9.63
29	283.9	10.65
30	274.3	11.73
31	265.6	12.71
32	257.8	13.75
33	249.9	14.69
34	243.3	15.68
35	237.3	16.65
36	231.9	17.72
37	228.0	18.73
38	224.9	19.72
39	222.0	20.69
40	220.8	21.64

Fig. 4. Comparing $2n/w$ to number of iterations as w varies ($n = 2^{12}$, $\|\mathcal{Q}_0\| = 2^{17}$).

and $2n/w$ also grows. This is because the critical relationship $\|\mathcal{Q}_0\| \geqslant n + 2^{w/2}$ is no longer maintained for $\|\mathcal{Q}_0\| \geqslant 34$.

The table in Fig. 5 presents the number of iterations executed by the modular algorithm as a function of $\|\mathcal{Q}_0\|$, the number of moduli. In this table we fix $w = 2^{32}$ and $n = 2^{12}$

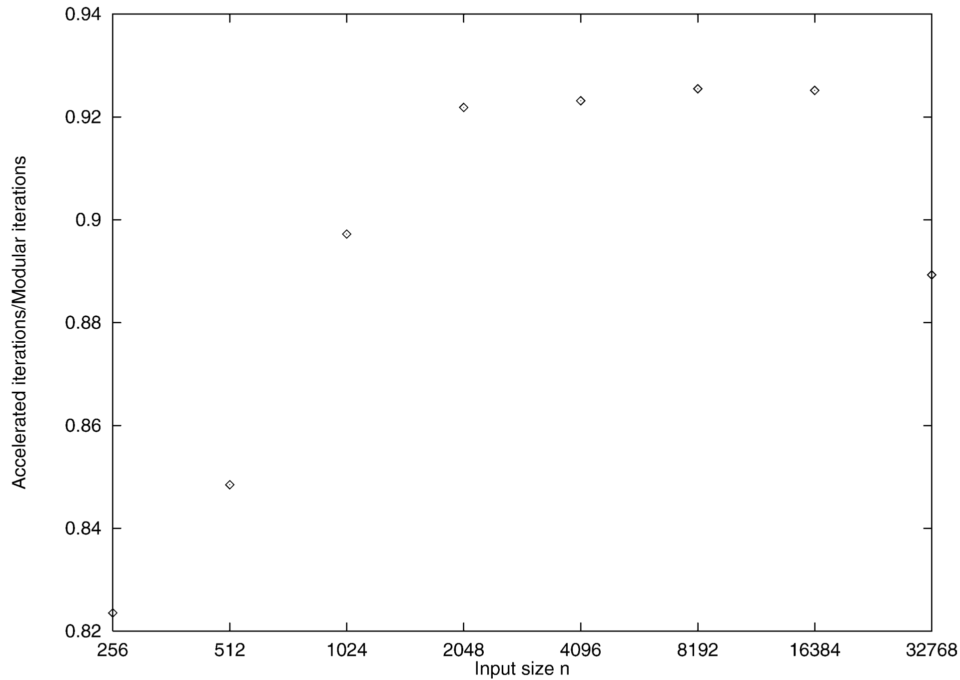


Number of moduli $\|\mathcal{Q}_0\|$	Average number of iterations	Average b size
2^9	412.4	22.51
2^{10}	356.2	20.93
2^{11}	323.7	19.78
2^{12}	299.6	18.69
2^{13}	280.8	17.68
2^{14}	268.0	16.69
2^{15}	261.7	15.73
2^{16}	258.4	14.71
2^{17}	257.6	13.63
2^{18}	257.1	12.65
2^{19}	257.0	11.69
2^{20}	257.0	10.63
2^{21}	257.0	9.77

Fig. 5. Comparing $2n/w$ to number of iterations as $\|\mathcal{Q}_0\|$ varies ($w = 32$, $n = 2^{12}$).

and vary the number of moduli. It shows that the number of iterations asymptotically approaches $2n/w$ as $\|\mathcal{Q}_0\|$ grows.

Figure 6 shows the ratio of the number of iterations required by the accelerated algorithm [1,2] to the number of iterations required by the modular algorithm for the tests summarized in Fig. 3. This ratio is a useful measure of the relative performance of the modular algorithm, since the accelerated algorithm is very efficient in the number of iterations of the main loop; it uses the k-ary reduction [5] when U and V are of similar sizes,



Input size n	Modular iterations	Accelerated iterations
2^8	17.0	14.0
2^9	33.0	28.0
2^{10}	65.2	58.5
2^{11}	129.2	119.1
2^{12}	257.6	237.8
2^{13}	513.9	475.6
2^{14}	1026.2	949.4
2^{15}	2051.9	1824.7

Fig. 6. Ratio of accelerated iterations to modular iterations.

and the dmod reduction [1,2] when the sizes differ greatly. Note that the modular algorithm compares favorably; for input sizes ranging from 2,048 bits to 16,384 bits the ratio is over 0.92.

Table 1 provides evidence that the values $|U/V \bmod p|$ are uniformly distributed over their respective ranges $[0, (p-1)/2]$. For each run of the algorithm registered in the table in Fig. 3, we examined the distribution, at each iteration, of the size of b, by counting the number of times the size is i bits. Table 1 shows the percentage of these values for $i = 1, \dots, 17$ for one particular (randomly chosen) iteration. Nearly 50% of the b values lie between 2^{30} and 2^{31}, 25% lie between 2^{29} and 2^{30}, 12.5% between 2^{28} and 2^{29}, and so on. This is as one would expect for a uniform distribution of the values.

Table 1
Distribution of size of b for one iteration, for varying n

Size of b	Values of n								Expected for uniform dist.
	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	
1–15	0.001	0.000	0.002	0.002	0.000	0.001	0.002	0.004	0.001
16	0.003	0.003	0.000	0.002	0.002	0.003	0.003	0.002	0.002
17	0.002	0.002	0.001	0.229	0.003	0.001	0.000	0.005	0.003
18	0.005	0.012	0.005	0.006	0.003	0.005	0.005	0.005	0.006
19	0.012	0.019	0.014	0.013	0.009	0.010	0.013	0.017	0.012
20	0.026	0.018	0.027	0.019	0.023	0.029	0.020	0.022	0.024
21	0.050	0.054	0.052	0.047	0.053	0.047	0.048	0.044	0.049
22	0.108	0.105	0.816	0.110	0.092	0.093	0.086	0.105	0.098
23	0.191	0.217	0.208	0.221	0.202	0.201	0.192	0.218	0.195
24	0.355	0.385	0.373	0.414	0.392	0.396	0.394	0.397	0.391
25	0.744	0.791	0.761	0.804	0.787	0.737	0.785	0.779	0.781
26	1.609	1.632	1.578	1.616	1.557	1.547	1.525	1.540	1.563
27	3.138	3.166	3.126	3.190	3.153	3.153	3.151	3.088	3.125
28	6.357	6.316	6.216	6.241	6.237	6.252	6.217	6.144	6.25
29	12.471	12.553	12.479	12.411	12.698	12.492	12.492	12.613	12.5
30	24.940	24.769	25.121	24.862	24.915	24.965	24.917	25.013	25
31	49.989	49.951	49.933	49.976	49.822	50.038	50.095	49.829	50

Remarks and future work

We close with some miscellaneous remarks and ideas for future work relating to the new algorithm.

The same techniques used to extend Sorenson's GCD algorithm to also compute A and B such that $G = AU + BV$ (see [5, Section 7]) can be used to extend the new modular algorithm, with G, A, and B all expressed in modular representation before conversion back to standard representation is done. More moduli, and hence, more processors, are needed to assure recovery of A and B, since they grow in size as the algorithm progresses. Note also, that although the algorithm presented above restricts moduli to primes, a set of relatively prime moduli should also suffice with only minor complications to the algorithm (such as the test to see whether a modulus is relatively prime to V).

Attempts to implement a parallel version of the accelerated algorithm [17] on a shared memory multiprocessor have been disappointing, mainly because the algorithm requires fine-grain parallelism. For the same reason it seems likely that it would also be difficult to provide good implementations of the three algorithms mentioned above on a shared memory multiprocessor. The new modular algorithm would most likely suffer the same problem; however, it seems particularly well suited to large SIMD or data-parallel systems that can perform arithmetic on 32-bit words or larger, and better suited to such a system than the other three. Although SIMD architectures have suffered recently from a lack of popularity, they are still popular in certain problem domains, and at least one such architecture is commercially available at the present time [18]. Recent research in micro-architectures [15] indicates that future architectures may very well be descendants of today's SIMD designs.

The major task remaining is to provide a more sophisticated analysis of the worst-case and/or average case behavior; we believe that it can be proven that the algorithm can be shown to execute in $O(n/w + 2^{w/2})$ time in all cases by showing that the minimum $|U/V \bmod p|$ is always small with respect to 2^w. Another possibility for future work is to compare the performance of the modular algorithm and the other three algorithms on a data-parallel system. Finally, it may be possible to apply modular techniques to Schönhage's algorithm; since it takes $O_B(M(n)\log n)$ time [19, Section 8.10], where $M(n)$ is the time it takes to compute n-bit integer multiplication, it may be possible to use modular representation to reduce the complexity contributed by multiplication to the overall problem's complexity.

Acknowledgments

The authors thank Universidade Federal do Rio Grande do Sul, Baldwin–Wallace College, Kent State University, and Mount Union College for the use of their computers for the experimental part of this research. In addition, we thank the referees and A.L. Hentges for their valuable comments.

References

[1] T. Jebelean, A generalization of the binary GCD algorithm, in: ISSAC '93: International Symposium on Symbolic and Algebraic Computation, Kiev, Ukraine, ACM, 1993, pp. 111–116.
[2] K. Weber, The accelerated integer GCD algorithm, ACM Trans. Math. Softw. 21 (1995) 111–122.
[3] D.E. Knuth, The Art of Computer Programming, vol. 1: Fundamental Algorithms, in: Addison–Wesley Ser. Comput. Sci. and Inform. Process., Addison–Wesley, Reading, MA, 1975.
[4] K. Weber, Parallel integer GCD algorithms and their application to polynomial GCD, PhD thesis, Kent State University, 1994.
[5] J. Sorenson, Two fast GCD algorithms, J. Algorithms 16 (1) (1994) 110–144.
[6] T. Jebelean, An algorithm for exact division, J. Symbolic Comput. 15 (2) (1993) 169–180.
[7] J. Sorenson, I. Parberry, Two fast parallel prime number sieves, Inform. and Comput. 114 (1) (1994) 115–130.
[8] B. Chor, O. Goldreich, An improved parallel algorithm for integer GCD, Algorithmica 5 (1990) 1–10.
[9] S.M. Sedjelmaci, On a parallel Lehmer–Euclid GCD algorithm, in: ISSAC 2001: International Symposium on Symbolic and Algebraic Computation, Ontario, Canada, ACM, 2001, pp. 303–308.
[10] D.E. Knuth, The Art of Computer Programming, vol. 2: Seminumerical Algorithms, 2nd ed., in: Addison–Wesley Ser. Comput. Sci. and Inform. Process., Addison–Wesley, Reading, MA, 1981.
[11] J.B. Rosser, L. Schoenfeld, Approximate formulas for some functions of prime numbers, Illinois J. Math. 6 (1962) 64–94.
[12] A.M. Mood, F.A. Graybill, D.C. Boes, Introduction to the Theory of Statistics, 3rd ed., McGraw-Hill, New York, 1974.
[13] L. Panaitopol, Inequalities concerning the function $\pi(x)$: applications, Acta Arithmetica 94 (2000) 373–381.
[14] J.B. Rosser, L. Schoenfeld, Abstract of scientific communications, in: Internat. Congr. Math., Moscow, 1966, Section 3: Theory of Numbers.
[15] P. Beckett, A. Jennings, Towards nanocomputer architecture, in: F. Lai, J. Morris (Eds.), Seventh Asia-Pacific Computer Systems Architecture Conference (ASAC 2002), Australian Computer Society, Melbourne, Australia, 2002, pp. 141–150.

[16] R. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. van Leeuwen (Ed.), Algorithms and Complexity, Handbook of Theoretical Computer Science, vol. A, Elsevier and MIT Press, 1990.
[17] K. Weber, Parallel implementation of the accelerated integer GCD algorithm, in: Special Issue on Parallel Symbolic Computation, J. Symbolic Comput. 21 (1996) 457–466.
[18] M. Meißner, S. Grimm, W. Straßer, J. Packer, D. Latimer, Parallel volume rendering on a single-chip SIMD architecture, in: Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, San Diego, CA, USA, IEEE Press, 2001, pp. 107–113.
[19] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison–Wesley, Reading, MA, 1974.

Post-print standardized by MSL Academic Endeavors, the imprint of the Michael Schwartz Library at Cleveland State University, 2017