Draft. Aimed at Math. Comp.

SCALED REMAINDER TREES

DANIEL J. BERNSTEIN

Abstract. It is well known that one can compute $U \bmod p_1, U \bmod p_2, \ldots$ in time $n(\lg n)^{2+o(1)}$ where n is the number of bits in $U, p_1, p_2, \ldots$. Here $U, p_1, p_2, \ldots$ can be integers or polynomials over a fixed finite field. Bostan, Lecerf, and Schost recently introduced an algorithm for the polynomial case that takes time $n(\lg n)^{2+o(1)}$ with a smaller $o(1)$. They did not claim any similar speedup for integers; their algorithm uses polynomial reversal and coefficient-matrix transposition, neither of which applies to integers. This paper presents a simpler algorithm that achieves the same speedup and that works for both polynomials and integers. This paper then points out several redundancies that can be eliminated from the algorithm, saving even more time.

1. Introduction

Let U be an integer. Let p_1, p_2, p_3, p_4 be positive integers. Here is the **remainder tree** of U, p_1, p_2, p_3, p_4:

$$\begin{tikzcd}[column sep=small]
 & & & U \bmod p_1p_2p_3p_4 \arrow[dll] \arrow[drr] & & & \\
 & U \bmod p_1p_2 \arrow[dl] \arrow[dr] & & & & U \bmod p_3p_4 \arrow[dl] \arrow[dr] & \\
U \bmod p_1 & & U \bmod p_2 & & U \bmod p_3 & & U \bmod p_4
\end{tikzcd}$$

Each $U \bmod P$ is represented inside a computer as a bit string in base 2: the string $b_0, b_1, \ldots, b_{e-1}$ represents the integer $b_0 + 2b_1 + \cdots + 2^{e-1}b_{e-1}$.

A well-known method to compute $U \bmod p_1, U \bmod p_2, U \bmod p_3, U \bmod p_4$ is to compute this remainder tree. Compute $U \bmod p_1p_2p_3p_4$; reduce modulo p_1p_2 to obtain $U \bmod p_1p_2$; reduce that modulo p_1 to obtain $U \bmod p_1$; and so on.

This paper introduces the **scaled remainder tree** of U, p_1, p_2, p_3, p_4:

$$\begin{tikzcd}[column sep=small]
 & & & (U/p_1p_2p_3p_4) \bmod 1 \arrow[dll] \arrow[drr] & & & \\
 & (U/p_1p_2) \bmod 1 \arrow[dl] \arrow[dr] & & & & (U/p_3p_4) \bmod 1 \arrow[dl] \arrow[dr] & \\
(U/p_1) \bmod 1 & & (U/p_2) \bmod 1 & & (U/p_3) \bmod 1 & & (U/p_4) \bmod 1
\end{tikzcd}$$

Each scaled-remainder-tree node $(U/P) \bmod 1$ is simply $1/P$ times the remainder-tree node $U \bmod P$. The node is represented inside a computer as a nearby real

Date: 2004.08.20. Permanent ID of this document: `e2b8da026cf72d01d97e20cf2874f278`.

2000 *Mathematics Subject Classification.* Primary 11Y16.

The author was supported by the National Science Foundation under grant DMS–0140542, and by the University of Sydney.

number $2^{-1}b_{-1} + 2^{-2}b_{-2} + \cdots + 2^{-e}b_{-e}$, which in turn is represented as the string $b_{-1}, b_{-2}, \ldots, b_{-e}$. A precise definition of "nearby" is given in Section 2 of this paper.

One can compute the remainders $U \bmod p_1, U \bmod p_2, U \bmod p_3, U \bmod p_4$ by computing this scaled remainder tree. Divide U by $p_1p_2p_3p_4$, and reduce modulo 1, to obtain $(U/p_1p_2p_3p_4) \bmod 1$; multiply by p_3p_4, and reduce modulo 1, to obtain $(U/p_1p_2) \bmod 1$; multiply that by p_2, and reduce modulo 1, to obtain $(U/p_1) \bmod 1$; and so on. Then multiply each $(U/p_j) \bmod 1$ by p_j to recover $U \bmod p_j$.

The advantage of the scaled remainder tree over the original unscaled remainder tree is that almost all of the divisions are replaced by similar-size multiplications. Multiplications are faster than divisions by a constant factor. Section 3 of this paper discusses scaled-remainder-tree speed in more detail.

Similar comments apply to 2-adic division and to polynomial division. One can also consider scaling remainders in contexts other than remainder trees.

Previous work. Fiduccia in [6, Section 2, Method C] introduced the remainder tree of polynomials $U, p_1, p_2, \ldots, p_m$, when each p_j has degree 1.

Moenck and Borodin in [9] proved that this remainder tree can be computed using $n(\lg n)^{3+o(1)}$ coefficient operations, where n is the total number of coefficients in $U, p_1, p_2, \ldots, p_m$. Moenck and Borodin developed, as a subroutine, a polynomial-division algorithm using $n(\lg n)^{2+o(1)}$ coefficient operations, where n is the number of input coefficients.

Moenck and Borodin in [9] also introduced remainder trees of integers. Moenck and Borodin claimed that if $p_1, p_2, \ldots, p_m$ are "single-precision integers" then the remainder tree of $U, p_1, p_2, \ldots, p_m$ can be computed in time $n(\lg n)^{2+o(1)}$ where n is the total number of bits in $U, p_1, p_2, \ldots, p_m$. It is, however, not clear what "single-precision integers" are supposed to be. I don't know who first proved that the same $n(\lg n)^{2+o(1)}$ bound holds for arbitrary $U, p_1, p_2, \ldots, p_m$.

In [3, Section 6], using a fast Newton-type division algorithm by Cook et al., Borodin and Moenck proved that the remainder tree of polynomials $U, p_1, p_2, \ldots, p_m$ can be computed using $n(\lg n)^{2+o(1)}$ coefficient operations. Borodin and Moenck also mentioned that $p_1, p_2, \ldots, p_m$ did not need to be linear; as above, I don't know who first proved the $n(\lg n)^{2+o(1)}$ bound in this case.

The exponent 2 in $n(\lg n)^{2+o(1)}$ is conjectured to be optimal for both polynomials and integers. There have nevertheless been some remainder-tree speedups:

- There is some redundancy in the usual algorithm for computing the product tree $p_1, p_2, p_3, p_4, p_1p_2, p_3p_4, p_1p_2p_3p_4$. Removing this redundancy speeds up the computation of product trees by a factor of roughly 3/2. This was pointed out recently by Robert Kramer.
- There are redundancies in Newton's method for computing reciprocals, in the usual multiply-by-reciprocal method of computing quotients, and in the multiply-by-quotient-and-subtract method of computing remainders. Brent, Schönhage, Grotefeld, Vetter, Karp, Markstein, Harley, Hanrot, Zimmermann, and I removed these redundancies, saving a factor of roughly 2 in the computation of remainders. See [2] for most of the details.
- One can use the product of approximate reciprocals of p_1 and p_2 as the starting point for the Newton iteration for the reciprocal of p_1p_2.

There have also been various speedups in the underlying multiplication subroutines.

Bostan, Lecerf, and Schost recently introduced a new algorithm for computing $U \bmod p_1, U \bmod p_2, \ldots, U \bmod p_m$, when $U, p_1, p_2, \ldots, p_m$ are polynomials. See [5] for the degree-1 case and [4, Section 3.1] for the general case; there were more authors of [4], but the general case was already mentioned in [5, Section 7]. Bostan, Lecerf, and Schost observed that their algorithm was faster than the Borodin-Moenck algorithm. (On the other hand, they neglected to consider the recent improvements in division.) They did not claim any similar speedups for integers; their algorithm relies on coefficient-matrix transposition and polynomial reversal, neither of which applies to integers.

The scaled-remainder-tree algorithm in this paper (without most of the speedups discussed in Section 3), when applied to polynomials, is as fast as the Bostan-Lecerf-Schost algorithm. In fact, its intermediate results—for various P, the coefficients of $x^{-1}, x^{-2}, \ldots, x^{-\deg P}$ in U/P—are exactly the unidentified intermediate results in the Bostan-Lecerf-Schost algorithm. This paper can be viewed as identifying the ring structure, rather than merely the module structure, behind the Bostan-Lecerf-Schost algorithm; this extra structure makes the algorithm much easier to understand, exposes some redundancies, and allows similar speedups for integers.

2. Accuracy

Let m be a nonnegative integer. Let $p_1, p_2, \ldots, p_m$ be positive integers. Let U be an integer. The **scaled remainder tree** of $U, p_1, p_2, \ldots, p_m$ is defined as follows. The root of the tree is $(U/p_1p_2\cdots p_m) \bmod 1$. If $m \le 1$ then that's the complete tree. If $m \ge 2$ then the left subtree is the scaled remainder tree of $U, p_1, p_2, \ldots, p_k$, and the right subtree is the scaled remainder tree of $U, p_{k+1}, p_{k+2}, \ldots, p_m$, where $k = \lceil m/2 \rceil$.

At the ith level of an ℓ-level scaled remainder tree, a node $(U/P) \bmod 1$ is represented by a bit string $b_{-1}, b_{-2}, \ldots, b_{-e}$ such that

$$\frac{U}{P} - 2^{-1}b_{-1} - 2^{-2}b_{-2} - \cdots - 2^{-e}b_{-e}$$

has distance smaller than $(\ell + i)/4\ell P$ from an integer.

Consider, for example, the scaled remainder tree of U, p_1, p_2, p_3, p_4. The root $(U/p_1p_2p_3p_4) \bmod 1$ is represented by a bit string $b_{-1}, b_{-2}, \ldots, b_{-e}$ such that

$$\left|\frac{U}{p_1p_2p_3p_4} - 2^{-1}b_{-1} - 2^{-2}b_{-2} - \cdots - 2^{-e}b_{-e} - q\right| < \frac{4}{12p_1p_2p_3p_4}$$

for some integer q. The second-level node $(U/p_1p_2) \bmod 1$ is represented by a bit string $b_{-1}, b_{-2}, \ldots, b_{-e}$ such that

$$\left|\frac{U}{p_1p_2} - 2^{-1}b_{-1} - 2^{-2}b_{-2} - \cdots - 2^{-e}b_{-e} - q\right| < \frac{5}{12p_1p_2}$$

for some integer q. The third-level node $(U/p_1) \bmod 1$ is represented by a bit string $b_{-1}, b_{-2}, \ldots, b_{-e}$ such that

$$\left|\frac{U}{p_1} - 2^{-1}b_{-1} - 2^{-2}b_{-2} - \cdots - 2^{-e}b_{-e} - q\right| < \frac{6}{12p_1}$$

for some integer q.

These distance bounds balance three desirable features:

- The bounds—most importantly, the bounds at the bottom level—are tight enough. The approximation to $(U/P) \bmod 1$ determines $(U/P) \bmod 1$: multiplying the approximation by P, and rounding the result to an integer, produces $U \bmod P$.
- The bounds at each level are slightly tighter than the bounds at the next level. The $1/4\ell P$ gap means that a simple multiply-and-round is sufficient to move from each level to the next. See Theorem 2.1.
- The bounds are not much tighter than necessary. Computations can thus be carried out in reasonably low precision.

I do not claim that the bounds $(\ell+1)/4\ell P, (\ell+2)/4\ell P, \ldots, (2\ell)/4\ell P$ are optimal. It might be a bit better to use $1/2\ell P, 2/2\ell P, \ldots, \ell/2\ell P$, for example.

Theorem 2.1. *Let P and Q be positive integers. Let ℓ be a positive integer. Let f be a positive integer. Let S be an integer. Let e be a positive integer such that $2^e \ge 2\ell P$. Let Y be an integer such that $Y \equiv QS \pmod{2^f}$. Let Z be an integer within $1/2$ of $Y/2^{f-e}$. Define $R = Z \bmod 2^e$. Then $R/2^e - QS/2^f$ has distance at most $1/4\ell P$ from an integer.*

In particular, if $S/2^f - U/PQ$ has distance smaller than $(\ell+i)/4\ell PQ$ from an integer, then $R/2^e - U/P$ has distance smaller than $(\ell+i+1)/4\ell P$ from an integer. In other words, if the f bits of $S \in \{0, 1, \ldots, 2^f - 1\}$ represent a node $(U/PQ) \bmod 1$ at the ith level of an ℓ-level scaled remainder tree, then the e bits of R represent a child node $(U/P) \bmod 1$ at the $(i+1)$st level.

See Theorem 2.2 for a faster method of finding a representation for $(U/P) \bmod 1$.

Proof. $R/2^e$ has integer distance from $Z/2^e$, which has distance at most $1/2^{e+1} \le 1/4\ell P$ from $Y/2^f$, which has integer distance from $QS/2^f$. $\square$

Theorem 2.2. *Let P and Q be integers larger than 1. Let ℓ be a positive integer. Let f be a positive integer such that $2^f \ge 8\ell PQ$. Let S be an integer such that $0 \le S < 2^f$. Let e be a positive integer such that $2^e \ge 8\ell P$. Let Y be an integer such that $-2^f + 1 \le Y \le 2^f - 1$ and $Y \equiv QS \pmod{2^f - 1}$. Let Z be an integer within $1/2$ of $Y/2^{f-e}$. Define $R = Z \bmod 2^e$. Then $R/2^e - QS/2^f$ has distance at most $1/4\ell P$ from an integer.*

In particular, if the f bits of S represent a node $(U/PQ) \bmod 1$ at the ith level of an ℓ-level scaled remainder tree, then the e bits of R represent a child node $(U/P) \bmod 1$ at the $(i+1)$st level, as in Theorem 2.1.

The advantage of Theorem 2.2 over Theorem 2.1 is that multiplication modulo $2^f - 1$ is, for large f, about twice as fast as multiplication modulo 2^f. See Section 3 for further discussion of speed. Theorem 2.2 requires two bits more precision (which one could reduce by shifting the S interval and tightening the Y interval), but this disadvantage is unnoticeable when f is large.

Proof. $R/2^e$ has integer distance from $Z/2^e$, which has distance at most $1/2^{e+1} \le 1/16\ell P$ from $Y/2^f$, which has distance at most $|Y|/2^f(2^f-1) \le 1/2^f \le 1/8\ell PQ \le 1/16\ell P$ from $Y/(2^f - 1)$, which has integer distance from $QS/(2^f - 1)$, which has distance at most $|QS|/2^f(2^f - 1) \le Q/2^f \le 1/8\ell P$ from $QS/2^f$. Finally $1/16\ell P + 1/16\ell P + 1/8\ell P = 1/4\ell P$. $\square$

Here is a concrete example of a scaled-remainder-tree calculation. Take $U = 314159265358979323$, $\ell = 3$, $m = 4$, $p_1 = 977$, $p_2 = 983$, $p_3 = 991$, and $p_4 = 997$.

Compute $p_1p_2 = 960391$, $p_3p_4 = 988027$, and $p_1p_2p_3p_4 = 948892238557$. Select exponents $45, 25, 15$ satisfying $2^{45} \geq 8\ell p_1p_2p_3p_4$, $2^{25} \geq 8\ell \max\{p_1p_2, p_3p_4\}$, and $2^{15} \geq 8\ell \max\{p_1, p_2, p_3, p_4\}$; also select an exponent 42 satisfying $2^{42}(\ell + 1) \geq 4\ell p_1p_2p_3p_4$. Perform a high-precision division to see that $U/p_1p_2p_3p_4$ has distance below 2^{-42} from $331080 + 853476536609/2^{45}$. Discard the integer part 331080. Multiply 853476536609 by 988027 modulo $2^{45}-1$, obtaining 29200555256697; divide the result by 2^{45-25} and round to an integer, obtaining 27847820. Multiply by 983 modulo $2^{25} - 1$, obtaining 27545795; divide the result by 2^{25-15} and round to an integer, obtaining 26900. Continue in the same way, using Theorem 2.2 repeatedly, to see that the scaled remainder tree of U, p_1, p_2, p_3, p_4 is approximately

$$\begin{tikzcd}[column sep=small]
 & & & 853476536609/2^{45} \arrow[dll] \arrow[drr] & & & \\
 & 27847820/2^{25} \arrow[dl] \arrow[dr] & & & & 15308656/2^{25} \arrow[dl] \arrow[dr] & \\
26900/2^{15} & & 27569/2^{15} & & 28338/2^{15} & & 4175/2^{15}
\end{tikzcd}$$

—i.e., that the scaled remainder tree is represented by the 45 bits of 853476536609, the 25 bits of 27847820, the 25 bits of 15308656, etc. Then round $977 \cdot 26900/2^{15}$, $983 \cdot 27569/2^{15}$, $991 \cdot 28338/2^{15}$, $997 \cdot 4175/2^{15}$ to integers, obtaining $(802, 827, 857, 127) = (U \bmod p_1, U \bmod p_2, U \bmod p_3, U \bmod p_4)$.

The 2-adic case. At this point I should give a 2-adic example, probably using the same inputs as above.

The x^{-1}-adic case. Let m be a nonnegative integer. Let $p_1, p_2, \ldots, p_m$ be monic polynomials in one variable x over a commutative ring. Let U be a polynomial in x over the same ring. The **scaled remainder tree** of $U, p_1, p_2, \ldots, p_m$ is defined exactly as above.

Each node $(U/P) \bmod 1$ is represented as follows. Expand U/P as a Laurent series in the variable x^{-1}. Then the representation is the sequence of coefficients $b_{-1}, b_{-2}, \ldots, b_{-\deg P}$ of $x^{-1}, x^{-2}, \ldots, x^{-\deg P}$ respectively. There is no need to keep more than $\deg P$ coefficients: roundoff errors do not accumulate for polynomials.

The analogue of Theorem 2.2 is easy. Take the coefficients $b_{-1}, b_{-2}, \ldots, b_{-f}$ of $x^{-1}, x^{-2}, \ldots, x^{-f}$ in U/PQ, where $f = \deg PQ$. Multiply $b_{-1}x^{f-1} + \cdots + b_{-f}x^0$ by Q modulo $x^f - 1$. Extract the coefficients of $x^{f-1}, x^{f-2}, \ldots, x^{f-\deg P}$ in the product. The result is the representation of $(U/P) \bmod 1$.

Here is a concrete example. Take $U = 3x^3 + 1x^2 + 4x + 1$, $m = 4$, $p_1 = x - 1$, $p_2 = x - 2$, $p_3 = x - 3$, and $p_4 = x - 4$. Compute $p_1p_2 = x^2 - 3x + 2$, $p_3p_4 = x^2 - 7x + 12$, and $p_1p_2p_3p_4 = x^4 - 10x^3 + 35x^2 - 50x + 24$. Multiply U by $1/p_1p_2p_3p_4 = x^{-4} + 10x^{-5} + 65x^{-6} + 350x^{-7} + \cdots$, obtaining $3x^{-1} + 31x^{-2} + 209x^{-3} + 1156x^{-4} + \cdots$. Multiply $3x^3 + 31x^2 + 209x + 1156$ by $x^2 - 7x + 12$ modulo $x^4 - 1$, obtaining $28x^3 + 65x^2 - 5581x + 13882$; also multiply $3x^3 + 31x^2 + 209x + 1156$ by $x^2 - 3x + 2$ modulo $x^4 - 1$, obtaining $122x^3 + 591x^2 - 3047x + 2334$. Multiply $28x + 65$ by $x - 2$ modulo $x^2 - 1$, obtaining $9x - 102$; also multiply $28x + 65$ by $x - 1$ modulo $x^2 - 1$, obtaining $37x - 37$. Multiply $122x + 591$ by $x - 4$ modulo $x^2 - 1$, obtaining $103x - 2242$; also multiply $122x + 591$ by $x - 3$ modulo $x^2 - 1$, obtaining $225x - 1651$.

The scaled remainder tree of U, p_1, p_2, p_3, p_4 is

$$\begin{tikzcd}[column sep=tiny]
 & & & 3x^{-1}+31x^{-2}+209x^{-3}+1156x^{-4}+\cdots \arrow[dll] \arrow[drr] & & & \\
 & 28x^{-1}+65x^{-2}+\cdots \arrow[dl] \arrow[dr] & & & & 122x^{-1}+591x^{-2}+\cdots \arrow[dl] \arrow[dr] & \\
9x^{-1}+\cdots & & 37x^{-1}+\cdots & & 103x^{-1}+\cdots & & 225x^{-1}+\cdots
\end{tikzcd}$$

represented by the coefficient sequences

$$(3, 31, 209, 1156), (28, 65), (122, 591), (9), (37), (103), (225).$$

The remainders $U \bmod p_1, U \bmod p_2, U \bmod p_3, U \bmod p_4$ are $9, 37, 103, 225$.

3. Speed

Say m is large, and say U is not much larger than $p_1p_2 \cdots p_m$. How long does it take to compute $U \bmod p_1, U \bmod p_2, \ldots, U \bmod p_m$?

Assume that $p_1, p_2, \ldots, p_m$ are bounded by 2^c. Assume for simplicity that m is a power of 2, namely $2^{\ell-1}$. Assume that computing a product modulo $2^n - 1$ takes time about $3n \lg n$, at least for various integers n with ratio converging to 1. Then computing an n-bit product of two integers takes time about $3n \lg n$.

To compute the product tree $p_1p_2, p_3p_4, \ldots, p_1p_2p_3p_4, \ldots,$ one multiplies $m/2$ pairs of c-bit numbers, then $m/4$ pairs of $2c$-bit numbers, and so on through 1 pair of $(m/2)c$-bit numbers. These multiplications take time about $3(m/2)2c \lg 2c + 3(m/4)4c \lg 4c + \cdots + 3(1)mc \lg mc \approx 3(\ell - 1)mc \lg(\sqrt{2mc})$.

Computing the root of the scaled remainder tree takes time $O(mc \lg mc)$ since U is not much larger than $p_1p_2 \cdots p_m$; this time becomes negligible as ℓ grows. Computing each ith-level node, for $i \geq 2$, takes one multiplication modulo $2^f - 1$ and some easy rounding; here f has to be at least $(m/2^{i-2})c + \lceil \lg 8\ell \rceil$ for Theorem 2.2. These multiplications take time about $3(2)(mc + \lceil \lg 8\ell \rceil) \lg(mc + \lceil \lg 8\ell \rceil) + \cdots + 3(m)(2c + \lceil \lg 8\ell \rceil) \lg(2c + \lceil \lg 8\ell \rceil) \approx 6(\ell - 1)mc \lg(\sqrt{2mc})$. The final multiplications by $p_1, \ldots, p_m$ take negligible time.

The 6-to-3 ratio here is unsurprising. The f-bit product involved in computing a scaled-remainder-tree node is twice as large as the corresponding $(f/2)$-bit product-tree node.

For comparison, computing the corresponding node in an unscaled remainder tree means reducing an f-bit number modulo an $(f/2)$-bit number. The fastest known method to do this takes more than twice as long as computing an f-bit product, when f is large.

The rest of this section points out various redundancies that can be eliminated from the product-tree-and-scaled-remainder-tree computation, reducing the total time from about $9(\ell - 1)mc \lg(\sqrt{2mc})$ to about $5(\ell - 1)mc \lg(\sqrt{2mc})$. See [13] for additional speedups in the unusual case that $p_1, p_2, \ldots, p_m$ vary wildly in size.

FFT doubling. There are four steps in multiplying p_1 by p_2. The first step is to compute a size-n Schönhage-Strassen transform of $p_1 \bmod 2^n - 1$, for an appropriate n. The second step is to compute a size-n transform of $p_2 \bmod 2^n - 1$. The third step is to multiply the transforms. The fourth step is to un-transform the product, obtaining $p_1p_2 \bmod 2^n - 1$. The first, second, and fourth steps each take time about $n \lg n$; the third step takes negligible time.

Suppose that p_1p_2 is then multiplied by p_3p_4. The first step is to compute a size-$2n$ transform of $p_1p_2 \bmod 2^{2n}-1$. But the first half of the size-$2n$ transform of $p_1p_2 \bmod 2^{2n}-1$ is exactly the size-n transform of $p_1p_2 \bmod 2^n-1$, which is already known. This redundancy was pointed out (in the polynomial case) by Robert Kramer in 2004.

Eliminating this redundancy—reusing the first half of each size-$2n$ transform and computing merely the second half—saves two halves of every three product-tree transforms, reducing the product-tree time from about $3(\ell-1)mc\lg(\sqrt{2mc})$ to about $2(\ell-1)mc\lg(\sqrt{2mc})$.

FFT caching. Later in the computation, starting from a scaled-remainder-tree node $(U/p_1p_2p_3p_4) \bmod 1$, one multiplies by p_1p_2 modulo 2^f-1, where f is about $2n$. One can choose f and n so that f is exactly $2n$; then the size-$2n$ transform of p_1p_2 is already known from the product-tree computation.

Eliminating this redundancy—caching the transform of p_1p_2 for future use—saves one of every three scaled-remainder-tree transforms, reducing the scaled-remainder-tree time from about $6(\ell-1)mc\lg(\sqrt{2mc})$ to about $4(\ell-1)mc\lg(\sqrt{2mc})$.

Similarly, the scaled-remainder-tree node $(U/p_1p_2p_3p_4) \bmod 1$ is multiplied by both p_1p_2 and p_3p_4, so it is transformed twice. Eliminating this redundancy reduces the scaled-remainder-tree time to about $3(\ell-1)mc\lg(\sqrt{2mc})$.

References

[1] Daniel J. Bernstein, *Fast multiplication and its applications*, to appear in Buhler-Stevenhagen *Algorithmic number theory* book. URL: `http://6 k2a 02 jr .roads-uae.com/papers.html#multapps`.

[2] Daniel J. Bernstein, *Removing redundancy in high-precision Newton iteration*, draft. URL: `http://6 k2a 02 jr .roads-uae.com/papers.html#fastnewton`. ID `def7f1e35fb654671c6f767b16b93d50`.

[3] Allan Borodin, Robert T. Moenck, *Fast modular transforms*, Journal of Computer and System Sciences **8** (1974), 366–386; older version, not a subset, in [9]. ISSN 0022–0000. MR 51:7365. URL: `http://6 k2a 02 jr .roads-uae.com/bib/entries.html#1974/borodin`.

[4] Alin Bostan, Grégoire Lecerf, Bruno Salvy, Éric Schost, Bernd Wiebelt, *Complexity issues in bivariate polynomial factorization*, in [11] (2004), 42–49.

[5] Alin Bostan, Grégoire Lecerf, Éric Schost, *Tellegen's principle into practice*, in [7] (2003), 37–44.

[6] Charles M. Fiduccia, *Polynomial evaluation via the division algorithm: the fast Fourier transform revisited*, in [10] (1972), 88–93. URL: `http://6 k2a 02 jr .roads-uae.com/bib/entries.html#1972/fiduccia-fft`.

[7] Hoon Hong (editor), *Proceedings of the 2003 international symposium on symbolic and algebraic computation*, Association for Computing Machinery, New York, 2003. ISBN 1–58113–641–2.

[8] Richard M. Karp (chairman), *13th annual symposium on switching and automata theory*, IEEE Computer Society, Northridge, 1972.

[9] Robert T. Moenck, Allan Borodin, *Fast modular transforms via division*, in [8] (1972), 90–96; newer version, not a superset, in [3]. URL: `http://6 k2a 02 jr .roads-uae.com/bib/entries.html#1972/moenck`.

[10] Arnold L. Rosenberg (chairman), *Fourth annual ACM symposium on theory of computing*, Association for Computing Machinery, New York, 1972. MR 50:1553.

[11] Josef Schicho (editor), *Proceedings of the 2004 international symposium on symbolic and algebraic computation*, Association for Computing Machinery, New York, 2004. ISBN 1–58113–827–X.

[12] Volker Strassen, *The computational complexity of continued fractions*, in [14] (1981), 51–67; see also newer version [13]. URL: `http://6 k2a 02 jr .roads-uae.com/bib/entries.html#1981/strassen`.

[13] Volker Strassen, *The computational complexity of continued fractions*, SIAM Journal on Computing **12** (1983), 1–27; see also older version [12]. ISSN 0097–5397. MR 84b:12004. URL: `http://6 k2a 02 jr .roads-uae.com/bib/entries.html#1983/strassen`.

[14] Paul S. Wang (editor), *SYM-SAC '81: proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, Snowbird, Utah, August 5–7, 1981*, Association for Computing Machinery, New York, 1981. ISBN 0–89791–047–8.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045, USA

E-mail address: `djb@cr.yp.to`