A Unified Root Algorithm in Bignum Arithmetic

Conference Paper · October 2018

DOI: 10.1109/IAEAC.2018.8577784

2 authors, including:

Yiping Cheng
Beijing Jiaotong University
33 PUBLICATIONS **58** CITATIONS
SEE PROFILE

A Unified Root Algorithm in Bignum Arithmetic

Yiping Cheng[1], Guizhi Cheng[2]
1. School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
2. School of Economics and Management, Beijing Information Science and Technology University, Beijing 100192, China
ypcheng@bjtu.edu.cn

***Abstract*—In bignum arithmetic, the cube root and higher-order root operations are conventionally treated as elementary functions, meaning that for them, correct rounding is not required, usually even without an error bound. In this paper we provide a unified algorithm `RootRem` to find the (arbitrary-ordered) root of bignums. This algorithm first finds the reciprocal root by using Newton iteration, and then obtains the root through a reciprocal operation. It has guaranteed correct rounding in all cases. `RootRem(2)`, which is to find the square root, is currently found to be less efficient than the existing algorithm `SqrtRem` proposed by Brent and Zimmermann. However, `RootRem($p$)` with $p \geq 3$ fill an important theoretical gap and are major improvements over the existing algorithms.**

***Keywords*—arbitrary precision; Newton iteration; computer arithmetic; multiprecision computing; correct rounding**

I. Introduction

The area of bignum (or arbitrary precision) arithmetic deals with the implementation of arithmetic operations of numbers whose digits of precision are limited not by CPU word length, but only by the available memory of the computer. It has found applications in cryptography, high accuracy numerical simulation, and financing, to name only a few. An overview of this area is provided in [1].

In this paper we are concerned with the computation of the square and higher-order root of bignums. Our interest in this topic came from the fact that square root is, together with the four commonly recognized fundamental operations, included in the IEEE 754 standard as a fundamental operation, being required to have guaranteed correct rounding. And, although higher-order root operations are not as important as the square root, they are still among the most often used elementary functions. As we shall see later, general root operations can be implemented by a general algorithm in which the square root is a special case.

There are three prevalent methods for the bignum root (especially the square root) computing and the closely related division problems. They are Newton iteration [1], [2], [3], SRT [4], and Goldschmidt [5]. The latter two methods are quite successful in hardware arithmetic, but they have lower asymptotic performances than the Newton method. We care very much about asymptotic performance, so we shall focus on the Newton method in this paper. In this direction, the most relevant references are [1, Sections 1.5 and 3.5], [2, Chapter 29], and [3]. The present paper was motivated by our failure in finding an existing practically used algorithm for cube and higher-order root operations with guaranteed correct rounding. Our approach here to this problem is a combination of the approach of [1, Section 3.5.1], whose main idea is to first find the reciprocal and then find the root itself, and the approach of [3] which emphasizes integer formulation. This combined approach has two advantages:

- In Newton iteration for finding the reciprocal root, each iteration step is division-free and the iterator has a unified form, which is considered helpful for us to develop a unified algorithm for arbitrary-ordered root operations.
- By formulating the problem in the integer domain, as was adopted in [3], we are able to carry out an accurate error analysis, which is pivotal in enabling us to develop a correct rounding algorithm.

We shall report what we have achieved using this approach in the rest of this paper. It is structured as follows. Section II describes the related previous work. Section III derives the `RecRoot` algorithm, which is aimed to find the reciprocal root. In Section IV, the main algorithm `RootRem` is described. In Section V, numerical experiment results are provided to compare `RootRem` with existing algorithms, and a conclusion is finally drawn.

II. Previous Work

A. Representation of Bignums

In computers, bignums (or arbitrary precision numbers) are stored using positional representation as described below.

A big integer A is represented by

$$A = \sum_{i=1}^{m} a_i \beta^{m-i}, \tag{1}$$

where β is the base or radix, and for each i, $0 \leq a_i \leq \beta - 1$. Each a_i is called a *word*, and m is thus the word-sequence *length* of A. Obviously, we have $0 \leq A \leq \beta^m - 1$.

Similarly, a big floating point number is represented by

$$A = \beta^e \sum_{i=1}^{m} a_i \beta^{-i}, \tag{2}$$

where the sequence $(a_1, a_2, \ldots, a_m)$ is the mantissa, and e is the exponent.

A representation (1) or (2) is said to be normalized, if $a_1 \neq 0$. Obviously, if integer A can be represented as a normalized m-word sequence, then $\beta^{m-1} \leq A < \beta^m$.

It is not difficult to conceive that floating point root computation poses no real difficulty beyond integer root computation. Therefore, in this paper, we confine our discussion to integers. Now suppose we have a pn-word representation of integer A:

$$A = \sum_{i=1}^{pn} a_i \beta^{pn-i}, \tag{3}$$

and we are to find its *integer p-th root*

$$X = \sum_{i=1}^{n} x_i \beta^{n-i} = \lfloor \sqrt[p]{A} \rfloor. \tag{4}$$

Note that for any nonnegative real number u, $\lfloor u \rfloor$ denotes the greatest integer not greater than u.

In this paper, we want to ensure that X has normalized representation, thus in the representation of A, it is required that at least one of $a_1, \ldots, a_p$ must be nonzero.

B. The SqrtRem Algorithm

In [1, Section 1.5.1], a square root algorithm `SqrtRem` was proposed. It is based on Newton iteration. For any given integer A, `SqrtRem` computes X, the integer square root of A, and the remainder $R = A - X^2$. This means

$$X \le \sqrt{A} < X + 1 \tag{5}$$

and hence

$$0 \le R = A - X^2 \le 2X. \tag{6}$$

Thus `SqrtRem` has guaranteed correct rounding.

C. Non-Correct-Rounding Root Algorithms

Also in [1], an algorithm `ApproxRecSquareRoot` was proposed for finding the reciprocal square root of big integers. This algorithm is also based on Newton iteration, but unlike `SqrtRem`, it is division-free in the iteration process. As its name suggests, it is an approximate algorithm. Actually, `ApproxRecSquareRoot` inspired the current work. For a detailed description of `ApproxRecSquareRoot` the reader may consult [1, Section 3.5.1]. In addition, in [2, Chapter 29], there are a number of sketchy descriptions of some cube and higer-order root algorithms but they do not seem to have guaranteed correct rounding.

III. Derivation of RecRoot

In this section we formulate the concept of "reciprocal integer p-th root" and derive an algorithm for its computing.

A. Definition of Reciprocal Integer p-th Root

Suppose

$$p \ge 2 \tag{7}$$

and we are given a pn-word integer A :

$$\beta^{pn-p} \le A = \sum_{i=1}^{pn} a_i \beta^{pn-i} < \beta^{pn}, \tag{8}$$

which means that at least one of $a_1, \ldots, a_p$ is nonzero. The *reciprocal integer p-th root* of A, is defined to be the integer Z such that

$$0 \le \beta^{2n} - \sqrt[p]{A} Z < \sqrt[p]{A}. \tag{9}$$

Apparently, such a Z exists and is unique, and we have

$$\beta^n \le Z \le \beta^{n+1}. \tag{10}$$

$Z = \beta^{n+1}$ if and only if $A = \beta^{pn-p}$. Otherwise, Z is an $(n+1)$-word integer which can be expressed as

$$Z = \sum_{i=1}^{n+1} z_i \beta^{n+1-i} \text{ with } z_1 \ne 0. \tag{11}$$

B. Splitting A and Z into High and Low Parts

The `RecRoot` algorithm is incremental in nature, so for convenience of exposition, A (as well as Z) is viewed as the concatenation of a high part and a low part. Let

$$n = h + l \text{ where } h, l > 0. \tag{12}$$

We will describe how to determine h and l later. We then split A and Z as follows.

$$A = \underbrace{\sum_{i=1}^{ph} a_i \beta^{ph-i}}_{A_{\mathcal{H}}} \cdot \beta^{pl} + \underbrace{\sum_{i=1}^{pl} a_{ph+i} \beta^{pl-i}}_{A_{\mathcal{L}}}. \tag{13}$$

If $Z = \beta^{n+1}$, then let $Z_{\mathcal{H}} = \beta^{h+1}$ and $Z_{\mathcal{L}} = 0$. Otherwise,

$$Z = \underbrace{\sum_{i=1}^{h+1} z_i \beta^{h+1-i}}_{Z_{\mathcal{H}}} \cdot \beta^{l} + \underbrace{\sum_{i=1}^{l} z_{h+1+i} \beta^{l-i}}_{Z_{\mathcal{L}}}. \tag{14}$$

Now in view of the above, we have

$$\beta^{ph-p} \le A_{\mathcal{H}} < \beta^{ph} \text{ and } 0 \le A_{\mathcal{L}} < \beta^{pl}. \tag{15}$$

$$\beta^h \le Z_{\mathcal{H}} < \beta^{h+1} \text{ and } 0 \le Z_{\mathcal{L}} < \beta^l, \text{ or } Z_{\mathcal{H}} = \beta^{h+1} \text{ and } Z_{\mathcal{L}} = 0. \tag{16}$$

C. A Key Property of $Z_{\mathcal{H}}$

Let $Z_{\mathcal{H}}$ be the the reciprocal integer p-th root of the ph-word integer $A_{\mathcal{H}}$, which means

$$0 \le \beta^{2h} - \sqrt[p]{A_{\mathcal{H}}} Z_{\mathcal{H}} < \sqrt[p]{A_{\mathcal{H}}}. \tag{17}$$

We first show a lemma.

Lemma 1: Suppose (7,15–17). If $h \ge 3$, or $p \ge 3$ and $h \ge 2$, then

$$\sqrt[p]{A_{\mathcal{H}} + 1}(Z_{\mathcal{H}} - 1) < \sqrt[p]{A_{\mathcal{H}}} Z_{\mathcal{H}}. \tag{18}$$

Proof: Since $p \ge 2$, if $h \ge 3$, then $h + 1 \le ph - p$. If $p \ge 3$ and $h \ge 2$, we also have $h + 1 \le ph - p$.

Let $W = Z_{\mathcal{H}} - 1$. Then $W < \beta^{h+1} \le \beta^{ph-p} \le A_{\mathcal{H}}$. Hence

$$(A_{\mathcal{H}}+1)W^p = A_{\mathcal{H}} W^p + W^p < A_{\mathcal{H}} W^p + A_{\mathcal{H}} W^{p-1} < A_{\mathcal{H}} (W+1)^p.$$

Taking p-th root on the left and right sides of the above inequality, we obtain (18). ∎

Note that inequality (18) generally does not hold for $p = 2$ and $h = 2$. For example, let $\beta = 10$, and $A_{\mathcal{H}} = \beta^{ph-p} = 100$, then $Z_{\mathcal{H}} = \beta^{h+1} = 1000$, but $\sqrt{101} \cdot 999 = 10039.8 \cdots > \sqrt{100} \cdot 1000$. Now, we will use this lemma to derive a key property of $Z_{\mathcal{H}}$ in relation to A.

Proposition 2: Suppose (7,13–17). Suppose further that $h \ge 3$, or $p \ge 3$ and $h \ge 2$. Then

$$0 < \beta^{2h+l} - \sqrt[p]{A}(Z_{\mathcal{H}} - 1) < 2\sqrt[p]{A}. \tag{19}$$

Proof: Multiplying (17) by β^l, we get

$$0 \le \beta^{2h+l} - \sqrt[p]{A_{\mathcal{H}}} \beta^l Z_{\mathcal{H}} < \sqrt[p]{A_{\mathcal{H}}} \beta^l. \tag{20}$$

By (13) and Lemma 1, we have

$$\sqrt[p]{A}(Z_{\mathcal{H}} - 1) < \sqrt[p]{A_{\mathcal{H}} + 1}\beta^l (Z_{\mathcal{H}} - 1) < \sqrt[p]{A_{\mathcal{H}}} \beta^l Z_{\mathcal{H}}.$$

This, combined with the left inequality of (20), gives

$$0 < \beta^{2h+l} - \sqrt[p]{A}(Z_{\mathcal{H}} - 1). \quad (21)$$

Now by (13) we have $\sqrt[p]{A_{\mathcal{H}}}\beta^l \le \sqrt[p]{A}$, which, combined with the right inequality of (20), gives $\beta^{2h+l} - \sqrt[p]{A}Z_{\mathcal{H}} < \sqrt[p]{A}$. This is equivalent to

$$\beta^{2h+l} - \sqrt[p]{A}(Z_{\mathcal{H}} - 1) < 2\sqrt[p]{A}. \quad (22)$$

The combination of (21) and (22) is just (19). ∎

From this proposition, we can see that either

$$0 \le \beta^{2h+l} - \sqrt[p]{A}Z_{\mathcal{H}} < \sqrt[p]{A}, \quad (23)$$

or

$$0 < \beta^{2h+l} - \sqrt[p]{A}(Z_{\mathcal{H}} - 1) < \sqrt[p]{A}. \quad (24)$$

D. Newton Iterator for Reciprocal Root

As `RecRoot` works incrementally, for pedagogic purposes we shall describe it here as a recursive algorithm, although in real implementation, loops are preferred to recursions. In recursive `RecRoot`, there is one Newton iteration after one recursive call. In this subsection we consider the iterator, i.e. the function that maps the initial value to the next.

Look at the defining inequality (9) of the reciprocal integer p-th root, we find that it is the integer truncation of the solution of the equation

$$\beta^{2pn}Z^{-p} - A = 0. \quad (25)$$

So the Newton iterator for finding the solution of (25) is

$$\begin{aligned} f(Z_0) &:= Z_0 - \frac{\beta^{2pn}Z_0^{-p} - A}{-p \cdot \beta^{2pn} \cdot Z_0^{-p-1}} \\ &= Z_0 + \frac{Z_0(\beta^{2pn} - AZ_0^p)}{p \cdot \beta^{2pn}}. \end{aligned} \quad (26)$$

This iterator f is not directly usable since f generally does not map to an integer. So instead, the following iterator is used in the real algorithm:

$$f^{\downarrow}(Z_0) := Z_0 + \lfloor \frac{Z_0(\beta^{2pn} - AZ_0^p)}{p \cdot \beta^{2pn}} \rfloor. \quad (27)$$

We then show a nice property of the iterator f.

Proposition 3: Let $Z_1 = f(Z_0)$ where f is defined by (26). Let

$$\epsilon_0 = 1 - \frac{\sqrt[p]{A}Z_0}{\beta^{2n}}, \quad (28)$$

$$\epsilon_1 = 1 - \frac{\sqrt[p]{A}Z_1}{\beta^{2n}}. \quad (29)$$

If

$$0 \le \epsilon_0 \le \frac{3}{p-1}, \quad (30)$$

then

$$0 \le \epsilon_1 \le \frac{p+1}{2}\epsilon_0^2. \quad (31)$$

Proof: It follows from (26) that

$$\begin{aligned} 1 - \frac{\sqrt[p]{A}Z_1}{\beta^{2n}} &= 1 - \frac{\sqrt[p]{A}Z_0}{\beta^{2n}} - \frac{\sqrt[p]{A}}{\beta^{2n}}\frac{Z_0(\beta^{2pn} - AZ_0^p)}{p \cdot \beta^{2pn}} \\ &= 1 - \frac{\sqrt[p]{A}Z_0}{\beta^{2n}} - \frac{\sqrt[p]{A}Z_0}{p \cdot \beta^{2n}}(1 - \frac{AZ_0^p}{\beta^{2pn}}). \end{aligned}$$

Hence by (28,29), this yields

$$\begin{aligned} \epsilon_1 &= \epsilon_0 - \frac{1-\epsilon_0}{p}[1 - (1-\epsilon_0)^p] \\ &= \epsilon_0 + \frac{1-\epsilon_0}{p} \cdot \sum_{k=1}^{p}\binom{p}{k}(-\epsilon_0)^k \\ &= \epsilon_0 + \frac{1}{p}[\sum_{k=1}^{p}\binom{p}{k}(-\epsilon_0)^k + \sum_{k=2}^{p+1}\binom{p}{k-1}(-\epsilon_0)^k] \\ &= \frac{1}{p} \cdot \sum_{k=2}^{p+1}\binom{p+1}{k}(-\epsilon_0)^k. \end{aligned} \quad (32)$$

This can be reexpressed as

$$\epsilon_1 = \begin{cases} \frac{1}{p}\sum_{s=1}^{\frac{p}{2}} \epsilon_0^{2s}[\binom{p+1}{2s} - \binom{p+1}{2s+1}\epsilon_0], & \text{for } p \text{ even,} \\ \frac{1}{p}\{\sum_{s=1}^{\frac{p-1}{2}} \epsilon_0^{2s}[\binom{p+1}{2s} - \binom{p+1}{2s+1}\epsilon_0] + \epsilon_0^{p+1}\}, & \text{for } p \text{ odd.} \end{cases} \quad (33)$$

Hence to have $\epsilon_1 \ge 0$, it is sufficient that $\binom{p+1}{2s} - \binom{p+1}{2s+1}\epsilon_0 \ge 0$ for all $s = 1, \ldots, \lfloor\frac{p}{2}\rfloor$, which is guaranteed by the condition

$$\epsilon_0 \le \frac{3}{p-1}. \quad (34)$$

The equation (32) can also be reexpressed as

$$\epsilon_1 = \frac{p+1}{2}\epsilon_0^2 \quad -$$

$$\begin{cases} \frac{1}{p}\{\sum_{s=1}^{\frac{p-2}{2}} \epsilon_0^{2s+1}[\binom{p+1}{2s+1} - \binom{p+1}{2s+2}\epsilon_0] + \epsilon_0^{p+1}\}, & \text{for } p \text{ even,} \\ \frac{1}{p}\sum_{s=1}^{\frac{p-1}{2}} \epsilon_0^{2s+1}[\binom{p+1}{2s+1} - \binom{p+1}{2s+2}\epsilon_0], & \text{for } p \text{ odd.} \end{cases} \quad (35)$$

Hence to have $\epsilon_1 \le \frac{p+1}{2}\epsilon_0^2$, it is sufficient that $\epsilon_0 \ge 0$ and $\binom{p+1}{2s+1} - \binom{p+1}{2s+2}\epsilon_0 \ge 0$ for all $s = 1, \ldots, \lfloor\frac{p-1}{2}\rfloor$, which is guaranteed by the condition

$$0 \le \epsilon_0 \le \frac{4}{p-2}. \quad (36)$$

The conjunction of (34) and (36) is (30), which is therefore a sufficient condition for (31). ∎

E. Newton Iteration for Reciprocal Root

It is now time for completing the description of our Newton iteration for reciprocal root. The initial value of the iteration is chosen to be an integer obtained from the result of the recursive call:

$$Z_0 = \tilde{Z}_{\mathcal{H}}\beta^l, \quad (37)$$

where $\tilde{Z}_{\mathcal{H}} = Z_{\mathcal{H}}$ or $Z_{\mathcal{H}} - 1$ and exactly which is chosen will be described later. Now,

$$\begin{aligned} f^{\downarrow}(\tilde{Z}_{\mathcal{H}}\beta^l) &= \tilde{Z}_{\mathcal{H}}\beta^l + \lfloor \frac{\tilde{Z}_{\mathcal{H}}\beta^l[\beta^{2pn} - A(\tilde{Z}_{\mathcal{H}}\beta^l)^p]}{p \cdot \beta^{2pn}} \rfloor \\ &= \tilde{Z}_{\mathcal{H}}\beta^l + \lfloor \frac{\tilde{Z}_{\mathcal{H}}(\beta^{2pn} - A\tilde{Z}_{\mathcal{H}}^p\beta^{pl})}{p \cdot \beta^{2pn-l}} \rfloor \\ &= \tilde{Z}_{\mathcal{H}}\beta^l + \lfloor \frac{\tilde{Z}_{\mathcal{H}}(\beta^{2ph+pl} - A\tilde{Z}_{\mathcal{H}}^p)}{p \cdot \beta^{2ph+pl-l}} \rfloor. \end{aligned} \quad (38)$$

Now we are able to show a key property of $f^{\downarrow}(\tilde{Z}_{\mathcal{H}}\beta^l)$.

Proposition 4: Suppose (7,8,12,13). Let $f^{\downarrow}$ be defined by (27). Suppose further that

$$2p + 2 \leq \beta, \quad (39)$$

$$h > l. \quad (40)$$

If

$$0 \leq \beta^{2h+l} - \sqrt[p]{A}\tilde{Z}_{\mathcal{H}} < 2\sqrt[p]{A}, \quad (41)$$

then

$$0 \leq \beta^{2n} - \sqrt[p]{A}f^{\downarrow}(\tilde{Z}_{\mathcal{H}}\beta^l) < 2\sqrt[p]{A}. \quad (42)$$

Proof: Let

$$\epsilon_0 = 1 - \frac{\sqrt[p]{A}\tilde{Z}_{\mathcal{H}}\beta^l}{\beta^{2n}}, \quad (43)$$

$$\epsilon_1 = 1 - \frac{\sqrt[p]{A}f(\tilde{Z}_{\mathcal{H}}\beta^l)}{\beta^{2n}}. \quad (44)$$

Then by (12),

$$\epsilon_0 = \frac{\beta^{2h+l} - \sqrt[p]{A}\tilde{Z}_{\mathcal{H}}}{\beta^{2h+l}}, \quad (45)$$

which, combined with (41,8,12,39), yields

$$0 \leq \epsilon_0 < \frac{2\sqrt[p]{A}}{\beta^{2h+l}} < \frac{2\beta^n}{\beta^{2h+l}} = \frac{2}{\beta^h} \leq \frac{2}{\beta} < \frac{3}{p-1}. \quad (46)$$

Hence by Proposition 3, we have $0 \leq \epsilon_1 \leq \frac{p+1}{2}\epsilon_0^2$, i.e.

$$0 \leq 1 - \frac{\sqrt[p]{A}f(\tilde{Z}_{\mathcal{H}}\beta^l)}{\beta^{2h+2l}} \leq \frac{p+1}{2}[1 - \frac{\sqrt[p]{A}\tilde{Z}_{\mathcal{H}}\beta^l}{\beta^{2h+2l}}]^2,$$

which is equivalent to

$$0 \leq \beta^{2h+2l} - \sqrt[p]{A}f(\tilde{Z}_{\mathcal{H}}\beta^l) \leq \frac{p+1}{2}\frac{(\beta^{2h+l} - \sqrt[p]{A}\tilde{Z}_{\mathcal{H}})^2}{\beta^{2h}}. \quad (47)$$

Combining (47) and (41), we arrive at

$$0 \leq \beta^{2h+2l} - \sqrt[p]{A}f(\tilde{Z}_{\mathcal{H}}\beta^l) < \frac{p+1}{2}\frac{4\sqrt[p]{A}}{\beta^{2h}}\sqrt[p]{A}. \quad (48)$$

By (8,12), and (39,40), we have

$$0 < \frac{p+1}{2}\frac{4\sqrt[p]{A}}{\beta^{2h}} < \frac{2(p+1)\beta^{h+l}}{\beta^{2h}} = \frac{2p+2}{\beta^{h-l}} \leq \frac{2p+2}{\beta} \leq 1. \quad (49)$$

Combining (48) and (49), we have

$$0 \leq \beta^{2n} - \sqrt[p]{A}f(\tilde{Z}_{\mathcal{H}}\beta^l) < \sqrt[p]{A}. \quad (50)$$

Now since

$$0 \leq f(\tilde{Z}_{\mathcal{H}}\beta^l) - f^{\downarrow}(\tilde{Z}_{\mathcal{H}}\beta^l) < 1, \quad (51)$$

we can finally obtain (42). ■

F. The Full Algorithm Description

Based on all the previous preparation, a full description of the RecRoot algorithm can now be given.

Algorithm 1 RecRoot (assumes $2p + 2 \leq \beta$)

```
Input: $A = \sum_{i=1}^{pn} a_i\beta^{pn-i}$, at least one of $a_1, \ldots, a_p$ nonzero
Output: $\hat{Z} = \sum_{i=1}^{n+1} \hat{z}_i\beta^{n+1-i}$, with $\hat{Z} \geq \beta^n$ and $0 \leq \beta^{2n} - \sqrt[p]{A}\hat{Z} < 2\sqrt[p]{A}$
 1: if $n \leq 3$ then
 2:     return result obtained using base algorithm
 3: end if
 4: $l = \lfloor \frac{n-1}{2} \rfloor$, $h = n - l$
 5: $A_{\mathcal{H}} = \sum_{i=1}^{ph} a_i\beta^{ph-i}$
 6: $\hat{Z}_{\mathcal{H}} = $ RecRoot$(A_{\mathcal{H}})$
 7: $(V_0, V_1, \ldots, V_p) = (A, A\hat{Z}_{\mathcal{H}}, \ldots, A\hat{Z}_{\mathcal{H}}^p)$
 8: if $V_p \leq \beta^{2ph+pl}$ then
 9:     $\tilde{Z}_{\mathcal{H}} = \hat{Z}_{\mathcal{H}}$
10: else
11:     $\tilde{Z}_{\mathcal{H}} = \hat{Z}_{\mathcal{H}} - 1$
12:     $V_p = \sum_{k=0}^{p}(-1)^k\binom{p}{k}V_{p-k}$
13: end if
14: $T = \beta^{2ph+pl} - V_p$
15: $\hat{Z} = \tilde{Z}_{\mathcal{H}}\beta^l + \lfloor \frac{\tilde{Z}_{\mathcal{H}}T}{p \cdot \beta^{2ph+pl-l}} \rfloor$
```

This algorithm needs some explanations:

- Different from what we imagined, the result of RecRoot is not necessarily the exact reciprocal integer p-th root Z, but may be $Z - 1$. If we want the assured exact reciprocal root, then a further check is needed. But we opt not to do that here because our true objective is the root, not the reciprocal root.
- After the recursive call, $\hat{Z}_{\mathcal{H}}$ is obtained, which is either $Z_{\mathcal{H}}$ or $Z_{\mathcal{H}} - 1$. Since $h \geq 3$ (see lines 1–4), by Proposition 2 either (23) or (24) holds. If $\hat{Z}_{\mathcal{H}} = Z_{\mathcal{H}}$, according to line 8, if (23) holds, then $\tilde{Z}_{\mathcal{H}} = Z_{\mathcal{H}}$, otherwise $\tilde{Z}_{\mathcal{H}} = Z_{\mathcal{H}} - 1$. If $\hat{Z}_{\mathcal{H}} = Z_{\mathcal{H}} - 1$, then because of (19), it must be that $\tilde{Z}_{\mathcal{H}} = \hat{Z}_{\mathcal{H}} = Z_{\mathcal{H}} - 1$. In all the three cases, $\tilde{Z}_{\mathcal{H}}$ satisfies (41), and finally (42) holds by Proposition 4.
- RecRoot needs a base algorithm to deal with the case $n \leq 3$. When $p \geq 3$, this can be relaxed so that the base algorithm only needs to deal with the case $n \leq 2$ (see Proposition 2). We have written a base algorithm named RecRootBase using lumped (non-incremental) Newton iteration. The details of RecRootBase are omitted here due to space limitation.
- The result of RecRoot is guaranteed to be an $(n + 1)$-word, i.e. it is always smaller than β^{n+1}. To see this, we note that we can add a statement to Proposition 3: If $0 < \epsilon_0 < \frac{3}{p-1}$, then $0 < \epsilon_1$. The detailed proof is omitted due to space limitation.
- Ideally, RecRoot is to be coded as a single algorithm dealing with arbitrary values of p. But in practice it can be used as a mere algorithm template, with different p's having different codes. The present author has finished

coding of RecRoot for $p = 2$ and $p = 3$. Coding the general algorithm needs a bit of additional work because the binomial coefficients $\binom{p}{k}$ may be big integers, which is currently not yet tackled by the present author.

IV. The RootRem Algorithm

We now try to find $X = \lfloor \sqrt[p]{A} \rfloor$. Suppose that by calling RecRoot, we have obtained an $(n+1)$-word Z satisfying

$$0 \le \beta^{2n} - \sqrt[p]{A}Z < 2\sqrt[p]{A}. \tag{52}$$

The algorithm D1Reciprocal proposed in [3] is able to find a reciprocal integer n-word Y of Z satisfying

$$0 \le \beta^{2n} - YZ < Z. \tag{53}$$

It can be derived from (52,53) that

$$\sqrt[p]{A} - 1 < Y < \frac{\beta^{2n}}{\frac{\beta^{2n}}{\sqrt[p]{A}} - 2} = \sqrt[p]{A} + \frac{2\sqrt[p]{A}}{\frac{\beta^{2n}}{\sqrt[p]{A}} - 2}. \tag{54}$$

If $\sqrt[p]{A} > \beta^n - 1$, then $X = \beta^n - 1$, and because $Y \le \beta^n$, we have $Y \le X + 1$. If $\sqrt[p]{A} \le \beta^n - 1$, then it is easy to see $\frac{2\sqrt[p]{A}}{\frac{\beta^{2n}}{\sqrt[p]{A}} - 2} < 2$. So in all cases, we have

$$\sqrt[p]{A} - 1 < Y < \sqrt[p]{A} + 2. \tag{55}$$

And the following algorithm is thus correct.

Algorithm 2 RootRem (assumes $2p + 2 \le \beta$)

```
Input: $A = \sum_{i=1}^{pn} a_i \beta^{pn-i}$, at least one of $a_1, \dots, a_p$ nonzero
Output: $X = \sum_{i=1}^{n} x_i \beta^{n-i}$ and $R = A - X^p$ which satisfy
    $0 \le R < (X+1)^p - X^p$
 1: $Z = \mathtt{RecRoot}(A)$
 2: $(X, R') = \mathtt{D1Reciprocal}(Z)$
 3: $(V_0, V_1, \dots, V_p) = (1, X, \dots, X^p)$
 4: if $V_p \le A$ then
 5:     $R = A - V_p$
 6: else
 7:     $X = X - 1$
 8:     $V_p = \sum_{k=0}^{p} (-1)^k \binom{p}{k} V_{p-k}$
 9:     if $V_p \le A$ then
10:         $R = A - V_p$
11:     else
12:         $X = X - 1$
13:         $V_p = \sum_{k=0}^{p} (-1)^k \binom{p}{k} V_{p-k}$
14:         $R = A - V_p$
15:     end if
16: end if
```

V. Numerical Results and Conclusion

Currently, we have implemented RootRem(2) and RootRem(3) in C. For comparison, we also implemented SqrtRem. All the three implementations have been thoroughly tested to ensure they are strictly correct. Then we carried out a numerical experiment to compare their performances. The computer running the experiment has an Intel i3-2120 @3.30GHz CPU, 4GB memory, and a Windows 7 32-bit operating system. The bignums whose square root and cube root we want to solve are randomly generated and they are exactly the same for all the three algorithms. Two plots are drawn. The first plot is more about fixed-point computation. It shows the average running time varying with pn which is the length of A, for algorithms SqrtRem, RootRem(2) and RootRem(3). The second plot is more about floating-point computation. It shows the average running time varying with n, where $A = A_1\beta^{2n-10}$, and the A_1's are randomly generated 10-word integers, for SqrtRem and RootRem(2).

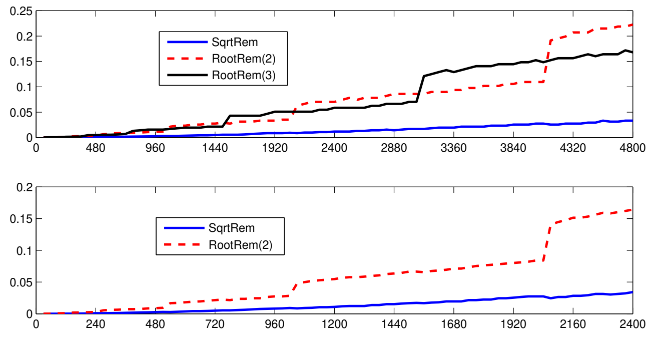

Fig. 1. Average Running Times (in seconds) of SqrtRem, RootRem(2), and RootRem(3)

We have previously hoped RootRem(2) to be more efficient than SqrtRem, as we considered that RootRem(2)'s division-free iteration might be a performance-boosting factor. However, to our disappointment, Fig. 1 clearly shows that RootRem(2) is less efficient. This may be due to the nature of the algorithm, but may also result from possible insufficient optimization of RecRoot(2), which needs further research.

As to RootRem(p) with $p \ge 3$, it is perhaps safe to say that they fill an important theoretical gap and are major improvements over the existing algorithms. We have two reasons: (1) They are provided as implementations of a unified algorithm for arbitrary values of p, which makes them elegant. (2) Being practically efficient as shown by the above plot, they have guaranteed correct rounding, and in the existing literature there is no algorithm known to the present authors that both has guaranteed correct rounding and is practically efficient.

References

[1] R. Brent and P. Zimmermann, *Modern Computer Arithmetic*. Cambridge University Press, 2010, online: http://www.loria.fr/zimmerma/mca/mca-cup-0.5.9.pdf.
[2] J. Arndt, *Matters Computational – Ideas, Algorithms, Source Code*. Springer, 2011, online: https://www.jjj.de/fxt/fxtbook.pdf.
[3] Y. Cheng and Z. Liu, "Refinement of a newton reciprocal algorithm for arbitrary precision numbers," in *Proceedings of 2016 International Conference on Progress in Informatics and Computing (PIC'16)*, Shanghai, China, 2016.
[4] D. Harris, S. Oberman, and M. Horowitz, "SRT division: Architectures, models, and implementations," Computer Systems Library, Standard University, Tech. Rep., 1998.
[5] P. Markstein, "Software division and square root using Goldschmidt's algorithms," in *Proceedings of 6th Conference on Real Numbers and Computers*, Dagstuhl, Germany, 2004.