Space-Efficient Evaluation of Hypergeometric Series

by

Howard Cheng, Barry Gergel, Ethan Kim, Eugene Zima

Report TR-CS-04-04
September 2004.

Space-Efficient Evaluation of Hypergeometric Series

Howard Cheng* Barry Gergel
Ethan Kim†
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
cheng@cs.uleth.ca, barry.gergel@uleth.ca, ethan.kim@uleth.ca

Eugene Zima
Department of Physics and Computer Science
Wilfrid Laurier University
Waterloo, Ontario, Canada
ezima@wlu.ca

Abstract

Hypergeometric series are used to approximate many important constants, such as e and Apéry's constant $\zeta(3)$. The evaluation of such series to high precision has traditionally been done by binary splitting followed by integer division. However, the numerator and the denominator computed by binary splitting usually contain a very large common factor. In this paper, we apply standard computer algebra techniques including modular computation and rational reconstruction to overcome the shortcomings of the binary splitting method. The space complexity of our algorithm is the same as a bound on the size of the *reduced* numerator and denominator of the series. Moreover, if the predicted bound is small, the time complexity is better than the standard binary splitting approach. Our approach allows a series to be evaluated to a higher precision without additional memory. We show that when our algorithm is applied to compute $\zeta(3)$, the memory requirement is significantly reduced compared to the binary splitting approach.

1 Introduction

We consider the evaluation of the hypergeometric series

$$S(N) = \sum_{n=0}^{N-1} \frac{a(n)}{b(n)} \prod_{i=0}^{n} \frac{p(i)}{q(i)} \tag{1}$$

to high precision, where a, b, p, and q are polynomials with integer coefficients, and $a(n)$, $b(n)$, $p(n)$, $q(n)$ have bit length $O(\log n)$. We also assume that the series is linearly convergent, so

*Supported by Natural Sciences and Engineering Research Council Discovery Grant and Research Tools and Instruments Grant.
†Supported by a Natural Sciences and Engineering Research Council Undergraduate Student Research Award.

that the nth term of (1) is $O(c^{-n})$ with $c > 1$. These series are commonly used in the high precision evaluation of elementary functions and other constants, including the exponential function, logarithms, trigonometric functions, and constants such as the Apéry's constant $\zeta(3)$ [7, 8].

A widely used approach to the computation of (1) is binary splitting [3, 8], which computes the numerator and denominator of the rational number $S(N)$. The decimal representation of $S(N)$ is then computed by fixed-point division of the numerator by the denominator. The binary splitting approach takes advantage of the special form of the series (1) to obtain a denominator that is relatively small (of size $O(N \log N)$). It also takes advantage of fast integer multiplication to obtain a time complexity of $O((\log N)^2 \mathsf{M}(N))$, where $\mathsf{M}(N) = O(N \log N \log\log N)$ is the complexity of integer multiplication of two N-bit integers [11]. The space complexity of the algorithm is $O(N \log N)$, the size of the computed numerator and denominator.

Typically, the numerator and denominator computed by binary splitting have large common factors. For example, in the computation of 640000 digits of $\zeta(3)$, as much as 86% of the size of the computed numerator and denominator can be attributed to their common factor [4]. Empirically, we have observed that the size of the reduced numerator and denominator is $O(N)$ instead of $O(N \log N)$ as computed by binary splitting. The additional digits computed not only slow down the final division but also require more memory to be used during the computation. For computing a large number of decimal digits, either the computation cannot be done at all or some data would have to be swapped to memory, increasing the computation time dramatically.

A different representation for integers was used by Cheng and Zima [4] to help reduce the size of the computed numerator and denominator. The integers were represented in *partially factored form*, so that common factors of small primes are easily removed. Although a completely factored representation would give the reduced numerator and denominator, addition of integers in factored form is too costly. By using a moderate number of primes in this representation, it was shown that the size of the intermediate results can be reduced to about half of those computed by standard binary splitting and computational time is also reduced by more than half. The asymptotic time and space complexities are unchanged from that of standard binary splitting.

In this paper, we study the application of well-known techniques in computer algebra to the evaluation of (1). If a bound on the size of the *reduced* numerator and denominator is known, we can compute the image of $S(N)$ in (1) under an appropriately chosen modulus. Rational number reconstruction can then be applied to recover the reduced numerator and denominator [5, 12, 13]. We show how to apply our techniques to the computation of $\zeta(3)$, including the prediction of the size of the reduced numerator and denominator. In particular, we obtain the desired $O(N)$ bound on the size of reduced numerator and denominator, which is an interesting result by itself. The techniques used in the analysis may be applied to similar hypergeometric series.

We can view our approach as an extension of our work in [4]—we obtain the advantage of using a completely factored form without its drawbacks. The time complexity of our algorithm is no worse than the binary splitting approach, and can be better if the reduced numerator and denominator have size significantly less than $O(N \log N)$. Furthermore, the space complexity of our algorithm is the same as the bound on the size of the reduced numerator and denominator. Our approach is different from that of taken by the PiFast program [7]. PiFast uses "large integers with limited precision" to reduce the space usage to $O(N)$ but the time complexity remains the same [7, see "Algorithms"→"Binary splitting method"]. Our approach is sensitive to the size of the reduced numerator and denominator and can be much faster when the size is predicted to be small.

The paper is organized as follows. Section 2 gives the necessary preliminaries for the rest of the

paper. We give the algorithm in Section 3 and a complexity analysis in Section 4. Section 5 shows the application of our algorithm to the computation of $\zeta(3)$. The relationship of our approach and the partially factored representation introduced in [4] is explained in Section 6. Concluding remarks are given in Section 7.

2 Preliminaries

In this section, we recall known algorithms that are needed in our new algorithm. We also give the relevant hypergeometric series representations of $\zeta(3)$ that will be used to illustrate the techniques discussed in this paper.

2.1 Binary Splitting

We give a brief overview of the binary splitting approach for the evaluation of (1) as described in [8]. Given bounds n_1, n_2 consider the partial sum

$$S = \sum_{n=n_1}^{n_2} \frac{a(n)}{b(n)} \frac{p(n_1) \cdots p(n)}{q(n_1) \cdots q(n)}. \tag{2}$$

The algorithm computes the integers $P = p(n_1) \cdots p(n_2)$, $Q = q(n_1) \cdots q(n_2)$, $B = b(n_1) \cdots b(n_2)$ and $T = BQS$. If $n_1 = n_2$, these values are computed directly. Otherwise, the series is divided into the left and right halves and the corresponding quantities are computed recursively. The results from each half are combined by the formulas:

$$P = P_l P_r, \quad Q = Q_l Q_r, \quad B = B_l B_r, \text{ and } \quad T = B_r Q_r T_l + B_l P_l T_r, \tag{3}$$

where the subscripts indicate whether the results are from the left half or the right half. Application of this algorithm to (1) starts with $n_1 = 0$ and $n_2 = N - 1$. Once these quantities are computed by binary splitting, a final division $S(N) = \frac{T}{BQ}$ is performed to obtain the decimal digits.

The success of the application of binary splitting is due to the fact that at each recursive invocation integers of relatively close sizes are multiplied. This provides a balance of operand sizes to take advantage of asymptotically fast integer multiplication algorithms. It was shown that the size of the computed results are $O(N \log N)$ bits and that the time complexity of binary splitting is $O((\log N)^2 \mathsf{M}(N))$ [8]. As we can see from (3), common factors between the numerator and the denominator are not removed.

In practice it is often the case that $b(n) = 1$ and hence $B = 1$. For the remainder of this paper, we will assume that $b(n) = 1$ to simplify the presentation of our algorithm. The algorithm and analysis given can easily be modified for $b(n) \neq 1$. We also note that by defining $\tilde{a}(n) = a(n)$, $\tilde{b}(n) = 1$, $\tilde{p}(0) = p(0)$, $\tilde{p}(n) = p(n)b(n-1)$ for $n > 0$, and $\tilde{q}(n) = q(n)b(n)$, we obtain a hypergeometric series of the desired form. Although the sizes of $\tilde{P}$ and $\tilde{Q}$ computed from the transformed series will be doubled due to additional common factors, the sizes of the reduced $\tilde{T}$ and $\tilde{Q}$ remain the same.

2.2 Rational Number Reconstruction

Given positive integers g and m, the rational number reconstruction problem is to find a and b such that $g \equiv ab^{-1} \bmod m$, $\gcd(b, m) = 1$, $|a| < \sqrt{m}/2$ and $0 < b \leq \sqrt{m}$. An algorithm based

on the Euclidean algorithm was first given by Wang, Guy, and Davenport [12]. It has quadratic time complexity but linear space complexity. Collins and Encarnación provided an algorithm which has the same complexity but is faster in practice [5]. Recently, Pan and Wang gave an algorithm which has time complexity $O((\log\log m)\mathsf{M}(\log m))$ [13]. Most current practical implementations are quadratic.

2.3 Computation of $\zeta(3)$

To illustrate our approach in this paper, we consider the following formula for computing $\zeta(3)$ to high precision [8]

$$\zeta(3) \approx \frac{1}{2}\sum_{n=0}^{N-1} \frac{(-1)^n \left(205\,n^2 + 250\,n + 77\right) \left((n+1)!\right)^5 \left(n!\right)^5}{\left((2\,n+2)!\right)^5}. \tag{4}$$

Here, $a(n) = 205n^2 + 250n + 77$, $b(n) = 1$, $p(0) = 1$, $p(n) = -n^5$ for $n > 0$, and $q(n) = 32(2n+1)^5$. This series gives approximately 3.01 decimal digits of accuracy for each extra term.

We note that another formula obtained by creative telescoping [1, 2] has also been used for the computation of $\zeta(3)$:

$$\zeta(3) \approx \frac{1}{24}\sum_{n=0}^{N-1} \frac{(-1)^n a(n)((2n+1)!(2n)!n!)^3}{(3n+2)!((4n+3)!)^3}, \tag{5}$$

where $a(n) = 126392n^5 + 412708n^4 + 531578n^3 + 336367n^2 + 104000n + 12463$. We will only use (4) because it is simpler to analyze. Although formula (5) converges faster than formula (4), it has been observed that the reduced numerators and denominators computed by the two series are similar for the same number of digits of accuracy [4, Table 1]. It has also been shown that the partially factored form was more successful in removing common factors from the results computed by (4) than from those computed by (5) [4, Table 6].

3 Algorithm

We now give an overview of the algorithm. Let $\hat{T}$ and $\hat{Q}$ be the reduced numerator and denominator of $S(N)$. We assume that a bound κ on the bit lengths (and hence the magnitudes) of $\hat{T}$ and $\hat{Q}$ is given to the algorithm.

Algorithm 1 Computation of the decimal expansion of $S(N)$.

1: Choose a sufficiently large modulus m such that $\gcd(m, \hat{Q}) = 1$
2: Compute the image g such that $g \equiv \hat{T}\hat{Q}^{-1} \bmod m$.
3: Apply rational number reconstruction on g and m to obtain $\hat{T}$ and $\hat{Q}$.
4: Perform fixed-point division on $\hat{T}$ and $\hat{Q}$ to obtain the decimal expansion of $S(N)$.

Step 3 makes use of standard rational number reconstruction algorithms as discussed in Section 2.2. Step 4 is the same as that of the standard binary splitting approach in Section 2.1. In the following subsections we describe the first two steps in more detail.

3.1 Choice of Modulus

In order to perform the computation successfully, we must ensure that the chosen modulus m is sufficiently large. In particular, we must ensure that

$$2\hat{T}\hat{Q} < m. \tag{6}$$

In other words, the bit length of m should be at least $2\kappa + 3$. Furthermore, we must ensure that m is relatively prime to $\hat{Q}$. We note that any prime larger than $q(n)$ for $0 \leq n < N$ is relatively prime to Q. With our assumption that $q(n)$ has size $O(\log n)$ bits, it suffices to find primes which have size $O(\log N)$ bits. Finding such primes is generally feasible computationally, and we will assume that such a list of primes have been precomputed. The product of sufficiently many such primes serves as a suitable modulus. The product of primes should be computed by a form of binary splitting to take advantage of fast integer multiplication algorithms.

3.2 Computation of Image

We now discuss how to compute the image $g \equiv \hat{T}\hat{Q}^{-1} \bmod m$. First, since $g \equiv TQ^{-1} \bmod m$, we may in fact compute the values of T and Q as computed by standard binary splitting modulo m. By first computing T and Q modulo m and then computing $g \equiv TQ^{-1} \bmod m$, we only need to compute modular inverses once.

Computing T and Q modulo m in a straightforward manner is inefficient (e.g. by adding one term at a time from $n = 0, \ldots, N-1$) because modulo m reductions have to be performed after almost every step. To perform the computation efficiently we must take care to perform modulo m reductions only when necessary. Thus, we will take the following approach.

Algorithm 2 Computation of $g \equiv TQ^{-1} \bmod m$

1: Determine the largest grouping factor G such that the values T, P, and Q for the partial sum in the range $[n_1, n_1 + G)$ satisfy $T, P, Q < m$ for any n_1.
2: Divide the range $[0, N)$ into $\lfloor N/G \rfloor$ groups of size G and possibly one additional group of size $N \bmod G$.
3: For each group, compute the values of T, P, and Q using binary splitting.
4: Combine the values computed above using (3) modulo m.
5: Compute $g \equiv TQ^{-1} \bmod m$.

Steps 3 and 4 can be interleaved by using three variables to accumulate the current values of T, P, and Q as we process each group, so we do not need to store the computed values for each group separately.

We also note that an optimization can be made by combining the values from each group backwards—from the last group to the first group. The value of P is not needed in the final division, and when the groups are combined from right to left by (3) we do not need the value of P_r. Therefore, it is not necessary to compute P and we may eliminate one multiplication. However, if one wishes to extend the results to more terms (in order to compute additional digits), the value of P is required.

One needs to study the particular choices of the polynomials $a(n)$, $p(n)$, and $q(n)$ in order to determine the grouping factor G. Let a_{max}, p_{max}, and q_{max} be the maximum values attained by the three polynomials in the interval $n \in [0, N)$, respectively. Such values can easily be computed

(e.g. using calculus). It is easy to see that the values T, P, and Q computed by binary splitting in the range $[n_1, n_1 + G)$ satisfy

$$\begin{aligned} T &\leq G \cdot a_{max} \cdot \max(p_{max}, q_{max})^G \\ P &\leq p_{max}^G \\ Q &\leq q_{max}^G. \end{aligned}$$

Therefore, G is the largest integer satisfying

$$\begin{aligned} G \cdot \max(p_{max}, q_{max})^G &< m/a_{max} \\ G &< \min(\log_{p_{max}} m, \log_{q_{max}} m). \end{aligned} \tag{7}$$

The appropriate value of G can be found quickly by numerical methods. We also note that the first inequality can be solved using the Lambert W function [6]. In practice, the values of the polynomials $a(n)$, $p(n)$, and $q(n)$ are often smaller when n is small, so it may be possible to use larger groups for smaller values of n.

Finally, we note that the values of $a(n)$, $p(n)$, and $q(n)$ can be computed using chains of recurrences as was done in [4, 14]. Since $a_{max}, p_{max}, q_{max}$ have size $O(\log N)$, it is likely that $a_{max}, p_{max}, q_{max} < m$. Thus, the polynomials can be evaluated efficiently as modular reductions are not required.

4 Time and Space Complexity

In this section, we give both time and space complexity analysis of Algorithm 1. In the first step, we compute a modulus m of size $O(\kappa)$ bits using procedure similar to binary splitting. Using a similar analysis as in [8], one sees that this step has time complexity $O((\log \kappa)\mathsf{M}(\kappa))$ and space complexity $O(\kappa)$. The rational number reconstruction in step 3 also has time complexity $O((\log \kappa)\mathsf{M}(\kappa))$ and space complexity $O(\kappa)$ [13]. The division in the last step can be computed in $O(\mathsf{M}(\kappa + N))$ time and requires space $O(\kappa + N)$.

We now examine the complexity of computing the image g modulo m in Algorithm 2. The computation of the grouping factor G in the first step is fast and negligible compared to the remainder of the computation. In step 3, binary splitting is applied to each group to compute results of size $O(\kappa)$, so that the time complexity is $O((\log G)\mathsf{M}(\kappa))$ and the space complexity is $O(\kappa)$. Note that the size assumption on the polynomials $a(n)$, $p(n)$, $q(n)$ implies that they can be evaluated in $O(\log N)$ time at each point. Thus, the total time complexity due to binary splitting is $O((N/G)(\log G)\mathsf{M}(\kappa) + N \log N)$. Finally, combining the results of the groups in step 4 has time complexity $O((N/G)\mathsf{M}(\kappa))$. From (7) and properties of the Lambert W function [6], we see that $G = \Theta(\kappa/\log N)$ and hence $N/G = \Theta((N \log N)/\kappa)$. Therefore, the total time complexity is

$$O(((N \log N)/\kappa)(\log \kappa - \log\log N)\mathsf{M}(\kappa) + N \log N).$$

Finally, since binary splitting and combination of results from each group can be interleaved, the amount of space required is $O(\kappa)$.

We summarize the complexity result below. We note that above analysis is only valid if $\kappa = O(N)$ because the number of groups N/G would be less than one otherwise.

Theorem 1 *Let $\kappa = O(N)$ be a bound on the bit length of the reduced numerator and denominator of $S(N)$ in (1). Our algorithm has time complexity $O(((N \log N)/\kappa)(\log \kappa - \log\log N)\mathsf{M}(\kappa) + N \log N + \mathsf{M}(\kappa + N))$ and space complexity $O(\kappa)$.* □

If $\kappa = O(N)$, we have a time complexity of $O((\log N)^2 \mathsf{M}(N))$, which is the same as that of binary splitting. If $\kappa = O(\log N)$, then the complexity reduces to $O(N(\log\log N)\mathsf{M}(\log N))$. Again, we emphasize that κ is a bound on the *reduced* numerator and denominator of $S(N)$ and can be significantly smaller than the $O(N \log N)$ numerator and denominator computed by binary splitting.

5 Application to $\zeta(3)$

In this section, we showed how to compute the bound κ on the size of the reduced numerator $\hat{T}$ and denominator $\hat{Q}$ in formula (4) for the computation of $\zeta(3)$.

We note that since $\zeta(3) = 1.202\ldots$, the size of T and Q (and also $\hat{T}$ and $\hat{Q}$) cannot differ by more than 1 decimal digit. As a result, we will concentrate on computing a bound on $\hat{Q}$ only.

We first show how we can obtain the size of Q computed by standard binary splitting. From the formulas $Q = q(0) \cdots q(N-1)$ and $q(n) = 32(2n+1)^5$, we see that

$$Q = 2^{5N} \prod_{i=0}^{N-1} (2i+1)^5 = \frac{(2N)!^5}{N!^5}. \tag{8}$$

Therefore, the size of Q can easily be computed by taking the logarithm of the Gamma function, for example.

Our approach to determine a bound on the size of $\hat{Q}$ is to determine a lower bound on the number of times each prime p divides into T and Q as computed by binary splitting. The minimum of the two quantities gives a lower bound on the size of the common factor, and removing this from the size of T and Q gives an upper bound on the size of $\hat{T}$ and $\hat{Q}$. In our analysis, it will be convenient to write T as

$$T = \sum_{n=0}^{N-1} a(n) p(0) \cdots p(n) q(n+1) \cdots q(N-1), \tag{9}$$

where it is understood that the term contains no $q(k)$ part when $n = N-1$. We will also make use of the well-known fact [9] that the number of times a prime p divides into $n!$ is

$$\sum_{i=1}^{\lfloor \log_p n \rfloor} \left\lfloor \frac{n}{p^i} \right\rfloor. \tag{10}$$

For $p = 2$, we see from (8) that p divides into Q exactly $5N$ times. Now, each term in T can be written as

$$a(n) n!^5 2^{5(N-n-1)} \prod_{i=n+1}^{N-1} (2i+1)^5 \tag{11}$$

Ignoring the factors of 2 in $a(n)$, the number of times 2 divides into each term of T is bounded below by

$$5\left(\sum_{i=1}^{\lfloor \log_2 n\rfloor}\left\lfloor\frac{n}{2^i}\right\rfloor + N - n - 1\right) \geq 5\left(N - n - 1 + \sum_{i=1}^{\lfloor \log_2 n\rfloor}\left(\frac{n}{2^i} - 1\right)\right)$$
$$\geq 5\left(N - n - 1 + \left(n - \frac{n}{2^{\lfloor \log_2 n\rfloor}}\right) - \lfloor \log_2 n\rfloor\right) \geq 5(N - 3 - \lfloor \log_2 n\rfloor).$$

The minimum is obtained when $n = N - 1$, and hence 2 divides into T at least $5(N - 3 - \lfloor \log_2(N-1)\rfloor)$ times. Thus, 2 divides into $\hat{Q}$ at most $15 + 5\lfloor \log_2(N-1)\rfloor$ times.

For all other primes p, a similar technique can be used to obtain a lower bound for the number of times p divides into T and Q. We can see from (8) that p divides into Q

$$5\sum_{i=1}^{\lfloor \log_p 2N\rfloor}\left\lfloor\frac{2N}{p^i}\right\rfloor - \left\lfloor\frac{N}{p^i}\right\rfloor \tag{12}$$

times. From (9), the number of times p divides into each term of T is at least

$$5\left(\sum_{i=1}^{\lfloor \log_p 2N\rfloor}\left\lfloor\frac{n}{p^i}\right\rfloor + \sum_{i=1}^{\lfloor \log_p 2N\rfloor}\left(\left\lfloor\frac{2N}{p^i}\right\rfloor + \left\lfloor\frac{N}{p^i}\right\rfloor\right) - \sum_{i=1}^{\lfloor \log_p 2N\rfloor}\left(\left\lfloor\frac{2n}{p^i}\right\rfloor - \left\lfloor\frac{n}{p^i}\right\rfloor\right)\right)$$
$$\geq 5\left(\sum_{i=1}^{\lfloor \log_p 2N\rfloor}\left(\left\lfloor\frac{2N}{p^i}\right\rfloor - \left\lfloor\frac{N}{p^i}\right\rfloor\right) - \log_p 2n\right).$$

Again, the minimum is obtained when $n = N - 1$, so that each prime up to $2N$ divides into $\hat{T}$ at most $5\log_p(2N-2)$ times. Therefore,

$$\hat{Q} \leq 2^{15+5\lfloor \log_2(N-1)\rfloor} \cdot \prod_{p\leq 2N} p^{5\log_p(2N-2)}. \tag{13}$$

From the prime number theorem [9], we know that the number of prime numbers up to n, $\pi(n)$, is $O(n/\log n)$. Thus,

$$\begin{aligned}\log_2 \hat{Q} &\leq 15 + 5\lfloor \log_2(N-1)\rfloor + \sum_{p\leq 2N} 5\log_2(2N-2)\\ &= O\left(15 + 5\log(N-1) + 5\log(2N-2)\cdot\frac{2N}{\log 2N}\right)\\ &= O(N).\end{aligned}$$

Thus we have the following result.

Theorem 2 *The size of the reduced numerator $\hat{T}$ and denominator $\hat{Q}$ computed by formula (4) is $O(N)$.* □

Digits	Terms (N)	Bound on $\hat{Q}$ (digits)	Size of Q (digits)
1000	333	1189	4481
5000	1661	6493	28140
10000	3322	12722	61279
50000	16610	67242	364436
100000	33220	130729	778872
500000	166100	679761	4474848
1000000	332200	1327237	9449705
10000000	3322000	13401351	111107033

Table 1: Bounds on the size of $\hat{Q}$ (in decimal digits) for various digits of $\zeta(3)$ in (4). Also shown is the size of Q as computed by binary splitting.

Table 1 shows the size of the bound on $\hat{Q}$ as computed by (13). We observed experimentally that the bounds are less than 10% of the size $\hat{T}$ and $\hat{Q}$ (up to 500000 digits). If the values of $\pi(n)$ are precomputed up to $n = 2N$, the computation of the bound can be done in constant time. A slightly less accurate bound can also be computed in constant time using the bound $\pi(n) < 1.25506n/\log n$ [10].

We remark the analysis above can be applied to other hypergeometric series which have a similar form as (4). For example, it is easy to analyze the prime divisors of the numerator and denominator of series in which each term can be expressed in factorials, binomial coefficients, or integer powers. Examples of such series can be found in [8].

6 Relationship to Partially Factored Representation

The partially factored representation of integers were used previously in order to reduce the size of the intermediate results in binary splitting [4]. Let $p_1, \ldots, p_m$ be the first m primes. An integer X is represented as

$$X = \left(\prod_{i=1}^{m} p_i^{\alpha_i} \right) x, \tag{14}$$

where $\alpha_i \geq 0$, and x, called the *standard component*, is in standard base-b representation. In this representation, it is easy to multiply and remove common small prime factors. However, addition and subtraction can be costly because any small prime factor that is not common to both operands must be multiplied into the standard component. No trial division or factoring is performed after addition to ensure that $\gcd(p_i, x) = 1$, so that only the small common prime factors remain in the exponent part of the representation (14). The values of $a(n)$, $p(n)$, and $q(n)$ are converted into partially factored representation such that $\gcd(p_i, x) = 1$.

It was shown that the partially factored representation was successful in the computation of $\zeta(3)$ because the numerator and the denominator have many small prime factors in common, so that many of these factors are preserved in the exponent part [4]. By using a moderate number of primes ($m \approx 500$), it was shown that binary splitting using partially factored representation was about 2.65 times faster than binary splitting, and the size of the final numerator and denominator computed have size slightly less than half of those obtained by standard binary splitting. Although

one may increase the number of primes used in order to reduce the size of the final results, the cost of additions and subtractions dominates and the resulting algorithm becomes slower. It was shown that for computing 1 million digits of $\zeta(3)$, approximately 60% of the computation time was spent multiplying prime factors into the standard component during additions [4].

In Section 5, our analysis of the size of $\hat{T}$ and $\hat{Q}$ for $\zeta(3)$ was done by examining the number of times each prime p divides into T and Q as computed by binary splitting. Although our analysis is similar to the idea of partially factored representation, our analysis in fact produces a better bound than the actual size of the numerator and denominator computed by binary splitting using partially factored representation. The reason is that our analysis was performed on the entire series, while binary splitting with partially factored integers are performed only on a portion of the series at any recursive invocation. It is possible that some prime factors in one portion is not a common factor until a large enough portion is considered. For example, consider the case when only two terms i and $i+1$ are combined, so that the computation of T is performed by (3) as

$$T = q(i+1)p(i)a(i) + p(i)p(i+1)a(i+1). \tag{15}$$

Small prime factors in $q(i+1)$ may not occur in $p(i+1)$, but may occur at $p(k)$ or $q(k)$ for some other k. These small prime factors will be multiplied into the standard component and never be removed. Since our analysis in Section 5 consider the whole series, each term of the final value of T has the form

$$a(i)p(0)\cdots p(i)q(i+1)\cdots q(N-1). \tag{16}$$

Thus, there is more opportunity to detect common factors.

Because we are performing the analysis on $p(n)$ and $q(n)$ symbolically only once at the beginning of computation, we do not incur any penalty on additions and subtractions as we did with the partially factored representation. Thus, it is feasible to examine all possible prime factors of T and Q in order to obtain a smaller bound on the size of $\hat{T}$ and $\hat{Q}$. Although our analysis can also be used to compute a large common factor at each step of binary splitting, it was shown that removing the common factor at each step in standard binary splitting does not provide significant improvement even if the common factor is given to the algorithm by an oracle at no cost [4, Table 4]. The improvement obtained from the reduced operands was offset by the cost of divisions of large integers.

7 Concluding Remarks

In this paper, we gave an algorithm that requires the same amount (up to a constant factor) of space as the bound on the size of the *reduced* numerator and denominator. When the bound is $O(N)$, the algorithm has the same time complexity as binary splitting but the space complexity is reduced. We showed how our techniques can be applied to the computation of $\zeta(3)$, including a derivation of an $O(N)$ bound on the size of the reduced numerator and denominator. Our algorithm makes it possible to evaluate $\zeta(3)$ and other similar hypergeometric series to a high precision with a reasonable amount of memory.

The use of modular computation offers some opportunity to parallelize the computation. Instead of multiplying the appropriate primes to obtain the modulus m and computing the image g modulo m as we have done, we may in fact compute the images under the different primes in parallel and apply Chinese remaindering to give the image modulo m. The latter can also be parallelized to

some extent. Chinese remaindering can also be used for "checkpointing"—if we wish to compute additional digits based on previously computed results g_1 and m_1, we may compute the new image modulo m_1 by adding more terms to g_1. In addition, an image g_2 may be computed under a modulus m_2, such that $m = m_1 m_2$ is the required new modulus. Chinese remaindering can then be used to combine the two results to obtain the image g modulo m.

References

[1] T. Amdeberhan. Faster and faster convergent series for $\zeta(3)$. *Electronic Journal of Combinatorics*, 3, 1996.

[2] T. Amdeberhan and D. Zeilberger. Hypergeometric series acceleration via the WZ method. *Electronic Journal of Combinatorics*, 4, 1997.

[3] J. Borwein and P. Borwein. *Pi and the AGM.* John Wiley and Sons, 1987.

[4] H. Cheng and E. V. Zima. On accelerated methods to evaluate sums of products of rational numbers. In *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation*, pages 54–61, 2000.

[5] G. E. Collins and M. J. Encarnación. Efficient rational number reconstruction. *Journal of Symbolic Computation*, 20(3):287–297, 1995.

[6] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[7] X. Gourdon and P. Sebah. Numbers, constants and computation. http://numbers.computation.free.fr/Constants/constants.html.

[8] B. Haible and T. Papanikolaou. Fast multiprecision evaluation of series of rational numbers. Technical Report TI–97/7, University of Darmstadt, 1997.

[9] K. H. Rosen. *Elementary Number Theory and Its Applications.* Addison-Wesley, 1992.

[10] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[11] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.

[12] P. S. Wang, M. J. T. Guy, and J. H. Davenport. p-adic reconstruction of rational numbers. *SIGSAM Bulletin*, 16(2):2–3, 1982.

[13] X. Wang and V. Y. Pan. Acceleration of euclidean algorithm and rational number reconstruction. *SIAM Journal on Computing*, 32(2):548–556, 2003.

[14] E. V. Zima. Simplification and optimization transformations of chains of recurrences. In *Proceedings of the 1995 International Symposium on Algebraic Computation*, pages 42–50, 1995.