Fast evaluation of iterated multiplication of very large polynomials: An application to chinese remainder theory

D. Laing[1] B. Litow[2]

(Received 30 August 2006; revised 18 December 2007)

Abstract

We consider the problem of exactly computing the number of integers in a Chinese Remainder Representation (CRR) whose pseudorank does not equal the rank. We call this number the census. The rank is key in developing CRR-intrinsic methods for comparing integers in CRR, a problem known to be notoriously difficult. Pseudorank can be computed in highly restrictive computation models. We have developed and implemented a fast, efficient algorithm for computing the census based on using a variant of the FFT to compute iterated products of polynomials of very large degree, and with arbitrary size integer coefficients. Experimental census results are tabulated. This census information makes possible a new approach to exploring the fine structure of CRR.

See http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/article/view/112 for this article, Published December 27, 2007. ISSN 1446-8735

Contents

1 Introduction

This research has its roots in work done between 1990 and 2001 by Davida, Litow and Chiu, the goal of which was to find a small integer approximation to the Chinese Remainder rank in order to construct a nearly logarithmic depth, polynomial, size Boolean circuit for integer division. The first attempt yielded $O(\log(n) \cdot \log\log n)$ depth [4], then Chiu [1] achieved a breakthrough and true $O(\log n)$ depth was achieved for division of n bit integers. This breakthrough was subsequently refined by Hesse [6] and applied to several problems in computational complexity.

Despite all of the applications for the approximation to rank (the pseudorank), little was known about how pseudorank behaved. Late in 2004, Litow

and Laing devised and implemented a polynomial time algorithm to measure pseudorank behavior [9]. Since then, we devised two improved algorithms for this purpose. Here we report on computational details of an implementation of the more efficient of these newer algorithms. We refer to this algorithm as Algorithm A. The work reported in this paper is part of a larger study into connections between arithmetic in Chinese Remainder Representation and certain questions of computational complexity.

The basic scheme of Algorithm A is an r-fold product C of polynomials, $C_1 \cdots C_r$, where $r = \Theta(n/\log n)$, and n is the maximum bit size of integers in Chinese Remainder Representation. Each polynomial C_i has degree $O(r)$, and nonnegative integer coefficients of size $O(n/r) = O(\log n)$. See section 3.1 for details. However, the integer coefficients of the polynomial C can be of the order of 2^n. The coefficients of C are the data we seek, so exact evaluation of C is required. This represents a challenge in applying FFT methods to computation of C which was successfully met by a modification of an implementation due to Hee [5], the details of which are covered in section 3.2. Algorithm A is interesting both as an exact census algorithm for CRR and as the motivation for the modification, refinement and profiling of a variant of the FFT which is suitable for iterated multiplication of polynomials where a loss in precision is unacceptable.

2 Chinese remainder representation

A CRR is specified by a single positive integer $P = p_1 \cdots p_r$, where $p_1, \ldots, p_r$ are consecutive odd primes.

Lemma 1 *Let r be the largest integer such that $p_r < n$, then $r = \Theta(n/\log n)$, and $2^{2n} > P > 2^n$.*

Proof: The claim for r follows from the prime number theorem, and the claim for P is exercise II.9.a in Vinagradov [15]. ♠

We fix $p_1 = 3$. CRR refers to both P and the associated representation of any integer $x < P$. The CRR of an integer x (integer will mean nonnegative integer) is the list $|x|_{p_1}, \ldots, |x|_{p_r}$, where $|x|_z = y$ means that y is the least integer such that $x \equiv y \bmod z$. The weak Chinese Remainder Theorem asserts that each $0 \leq x < P$ is uniquely identified by its CRR. We write $|x|_P$ to indicate that the integer x is given in CRR, rather than in radix notation. On the other hand, $|x|_{p_i}$ indicates that radix notation is used. By Lemma 1, if we want to represent integers up to n bits (we can do somewhat better), then $|x|_{p_i} < \log P = O(n)$, and $r = \Theta(n/\log n)$. Thus radix notation for $|x|_{p_i}$ involves at most $\log\log P = \log n$ bits.

The chief obstruction to using CRR for large scale arithmetic is comparison. While comparison is simple in radix, it is a major headache in CRR, as mentioned by Knuth [7]. Much of our work on the CRR-complexity connection has to do with CRR comparison.

Observe that knowing $|x|_2$ generally makes possible very efficient integer comparison algorithms. For example, for $x, y < P$, $x < y$ iff $||y - x|_P|_2 \equiv |x|_2 + |y|_2 \bmod 2$, since otherwise $|y - x|_P = y - x + P$. If parity were easy to compute in CRR, comparison would also be easy. The following strong form of the Chinese Remainder Theorem points out the main obstacle to CRR-based comparison.

Theorem 2

$$x + q(x) \cdot P = \sum_{i=1}^{r} |x \cdot (P/p_i)^{p_i - 2}|_{p_i} \cdot P/p_i \,.$$

Discussion and proofs of Theorem 2 are provided by Tanaka and Szabo [13] and Knuth [7]. Note that the integer $q(x)$, known as the rank of x, is less

than r. Observe that from Theorem 2 we have

$$|x|_2 = \Big| \sum_{i=1}^{r} x_i + q(x) \Big|_2 , \tag{1}$$

where

$$x_i = |x \cdot (P/p_i)^{p_i - 2}|_{p_i} ,$$

and we have used $|P|_2 = 1$. The integers x_i are all available via CRR, so to compute $|x|_2$ it suffices to know $|q(x)|_2$. However, there is no known way to efficiently compute $|q(x)|_2$ in a CRR intrinsic manner. This difficulty is the motivation for introducing the pseudorank of x.

Dividing through by P, from Theorem 2, we write

$$x/P + q(x) = \sum_{i=1}^{r} x_i/p_i . \tag{2}$$

Let g be the least integer such that $2^g > 4r$. We define integers $\alpha(x)$ and $\beta(x)$ by $\alpha(x) < 2^g$ such that

$$2^g \cdot \beta(x) + \alpha(x) = \sum_{i=1}^{r} \lfloor 2^g \cdot x_i/p_i \rfloor . \tag{3}$$

Observe that

$$\lfloor 2^g \cdot x_i/p_i \rfloor / 2^g$$

is the binary fraction μ such that $0 \le x_i/p_i - \mu < 1/2^g$. Thus, to g bits precision, $\beta(x) + \alpha(x)/2^g$ mimics $q(x) + x/P$. We call $\beta(x)$ the pseudorank of x.

The next result was proved by Davida and Litow [4] and Tarasov [14].

Theorem 3 *If $x > P/4$, then $\beta(x) = q(x)$. If $\beta(x) \ne q(x)$, then $\beta(x) = q(x) - 1$.*

The integers below $P/4$ comprise what we call the critical region of the CRR. The integer x is said to be good if $q(x) = \beta(x)$, otherwise it is said to be bad. All bad integers are in the critical region. Algorithm A computes the exact number (census) of bad integers. In other work, we derived an estimate for the census, and it is in very good agreement with results obtained from Algorithm A for a range of CRR up to one capable of handling 5000 bit integers [9].

2.1 Weighted finite automata

We use weighted finite automata (WFA) to describe the algorithms of this section. WFA is a huge topic. Examples of directions in WFA research are presented by Culik and Kari [3], Salomaa and Soi [11] and Salomaa and Kui [8]. The first general algebraic treatment of WFA appears to be due to Schützenberger [12]. We deal with only those aspects that are relevant to our algorithms in this section.

We work over CRR P. For our CRR oriented purposes a WFA, F, is a tuple (Σ, U, V, X_σ), where Σ is a finite alphabet, U is a $1 \times g$ matrix, V is a $g \times 1$ matrix, and for each $\sigma \in \Sigma$, X_σ is a $g \times g$ matrix. All matrix entries are rationals. We regard the indices of these matrices as states. The integer g is the number of states. It is useful to think of the nonzero entries of U as 'start' states, and the nonzero entries of V as 'final' states. Σ is partitioned as $\Sigma_1 \cup \cdots \cup \Sigma_r$, where the symbols of Σ_i are in bijective correspondence with $0, 1, \ldots, p_i - 1$. The k-behavior of F, denoted by $\langle F \rangle_k$ is defined to be

$$U \cdot \Big(\sum_{\sigma \in \Sigma} X_\sigma \Big)^k \cdot V \,.$$

We write $\langle F \rangle_k$ as

$$\sum_{\sigma_{i_1} \cdots \sigma_{i_k}} U \cdot X_{\sigma_{i_1}} \cdots X_{\sigma_{i_k}} \cdot V \,,$$

and a summand is denoted by $\langle F\rangle_k(\sigma_{i_1}\cdots\sigma_{i_k})$. We note that by design, we enforce $\langle F\rangle_k(\sigma_{i_1}\cdots\sigma_{i_k}) = 0$ if $i_j \geq i_h$ for $j < h$. This means that if $\sigma_{i_1}\cdots\sigma_{i_r}$ does not correspond to a CRR, any WFA that we construct will generate a zero summand for that string.

3 Exact census computation

Prior to the development of Algorithm A, we developed and implemented an algorithm that gave an approximate census [9]. This algorithm also depended on iterated polynomial multiplication. In both cases the coefficients of the polynomials had to be of large precision. The current scheme uses arbitrary precision integers in order to compute the census precisely, whereas the approximation algorithm used arbitrary precision complex rationals in order to guarantee a result within a certain error bound.

3.1 Algorithm A

It will be convenient to let

$$\mu_i(x) = \lfloor 2^g \cdot x_i/p_i \rfloor .$$

We start from equation (3). The only matter requiring comment is computation of $\sum_x \alpha(x)$. This can be done with a WFA, F. The states of F are pairs of integers of the form $(i, \sum_{j=1}^{i} \mu_j(x))$ for $i = 1, \ldots, r$. The start state is $(0, 0)$. We regard $(0, \sum_{j=1}^{0} \mu_j(x))$ as $(0, 0)$. The only nonzero entries of $X_{|x|_{p_i}}$ are indexed by the state transition (row, column) pairs

$$\Big(i-1, \sum_{j=1}^{i-1} \mu_j(x)\Big), \quad \Big(i, \sum_{j=1}^{i} \mu_j(x)\Big) .$$

The common nonzero entry value is 1. The only nonzero entry of matrix $X_{|x|_{p_1}} \cdots X_{|x|_{p_r}}$ will be indexed by $(0,0), (r, \sum_{j=1}^{r} \mu_j(x))$. Now, $\sum_{j=1}^{r} \mu_j(x) = a + 2^g \rho(x)$, where $a < 2^g$ is an integer. V has nonzero entries at states $(r, a + 2^g \cdot b)$, where $b < r$. The entry at such a state is a. We see that the r-output of F is $\alpha(x)$, and hence $\langle F \rangle_r = \sum_x \alpha(x)$.

Rather than use the native WFA just described, we recast the method underlying the WFA in terms of polynomial multiplication. Let C_i be the polynomial $\sum_{j=0}^{p_i - 1} z^{\mu_i(j)}$ for $i = 1, \ldots, r$. Let C be the r-fold multiplication of all C_i polynomials, where the degree of each term z^i arising from this operation is reduced modulo 2^g. Take C to be of the form $\sum_{i=0}^{2^g - 1} a_i \cdot z^i$, and compute from this $\sum_{i=0}^{2^g - 1} a_i \cdot i$.

Lemma 4

$$\sum_{i=0}^{2^g - 1} a_i \cdot i = \sum_x \alpha(x) .$$

Proof: The coefficient a_i is just the number of $x < P$ such that $\alpha(x) = i$. Indeed, this is just a partial summand computed in the r-behavior of the WFA described earlier in this section. The lemma follows from this. ♠

Theorem 5 *Letting $\mathcal{A}(n)$ denote the time to multiply two n bit integers, Algorithm A computes the census of bad integers in $O(n^2 \cdot \mathcal{A}(n))/\log n$ bitwise arithmetic operations.*

Proof: Multiplication of the r polynomials C_i, with reduction of all intermediate polynomial degrees modulo 2^g is doable in time dominated by the product of two degree 2^g polynomials with integer coefficients no larger than P. By Lemma 1, $P < 2^{2n}$, so coefficient multiplications involve at most

$\mathcal{A}(2n)$ bitwise arithmetic operations. Since multiplication is no worse than quadratic time, we can express this cost as $O(\mathcal{A}(n))$. Using FFT, or similar convolution transform methods, the $O(r)$ pairwise polynomial products can each be done using $2^g \cdot g \cdot \mathcal{A}(n)$ operations. By Lemma 1 this cost is bounded above by $O(n \cdot \mathcal{A}(n))$, and by the same lemma, $O(r) = O(n/\log n)$, yielding the time bound. ♠

3.2 Implementation of the polynomial multiplication

There are problems with using an FFT based on complex roots of unity with arbitrary precision. If a complex root of unity is approximated using a truncated Taylor series for the exponential function then the approximations are too large to be practically computable. The Nussbaumer convolution algorithm runs in $O(n \log n)$ and uses integer arithmetic throughout, making it suitable for use as a replacement for the FFT. Nussbaumer [10] provides an in depth discussion of the algorithm which covers the algorithm and convolution in general in great depth. Hee [5] details the variant which we are using, which is used for sequences of length $N = 2^b$.

Our implementation is based on the implementation by Hee [5]. The algorithm computes polynomial multiplication modulo $z^N - 1$. We decompose this multiplication using the Chinese Remainder Theorem into a combination of polynomial multiplications modulo $z^{N/2} - 1$ and $z^{N/2} + 1$. Clearly polynomial multiplication modulo $z^{N/2} - 1$ can be dealt with via recursion. Polynomial multiplication modulo $z^{N/2} + 1$ is more involved.

We perform the circular convolution modulo $P(z)$ of two polynomials by using the polynomial transform, which is similar to the DFT except that it operates over the field of polynomials rather than the field of complex numbers.

When multiplying polynomials modulo $z^{N/2} + 1$, we map the one dimen-

sional sequence of length M to a two dimensional sequence of length L_1 and width L_2. As we are dealing with sequences of length 2^b, L_1 will equal either L_2 or $2 \cdot L_2$. We now have L_1 circular convolutions modulo $z^{L_2}+1$ to perform, using z^{L_2/L_1} as the root of unity. Each of these circular convolutions is going to require a polynomial multiplication modulo $z^{L_2}+1$, and so we achieve polynomial multiplication modulo $z^{N/2}+1$ through recursion.

The speed of the implementation is greatly affected by how the recursion ends. Typically this will be with non-recursive routines for the convolution of short sequences. Techniques for discovering these are again provided by Nussbaumer [10]. The implementation by Hee mentioned earlier used a routine for the convolution of sequences length two to end the recursion, whereas we used routines for both sequence lengths two and four. A comparison on the differences in running times and number of arbitrary precision operations required are in the results section.

An earlier version of the census algorithm required much larger coefficients, to the point that memory was more of an issue than speed. This was dealt with by manually managing the swapping process. Polynomials were saved to disc in 2^b sections such that the width of each section was beneath a given threshold, and the polynomials were multiplied from these pieces via Toom–Cook decomposition [2]. This greatly extended the reach of the algorithm. Algorithm A was free from memory problems for the problem sizes we were interested in.

4 Results

All computations were performed on a Intel Xeon CPU 2.66 GHz with 512 Kb cache and 4096 Mb RAM. We made use of the GNU Multiple Precision Math Library (GMP), a fast arbitrary precision math library available under the GNU Lesser General Public License from http://gmplib.org/.

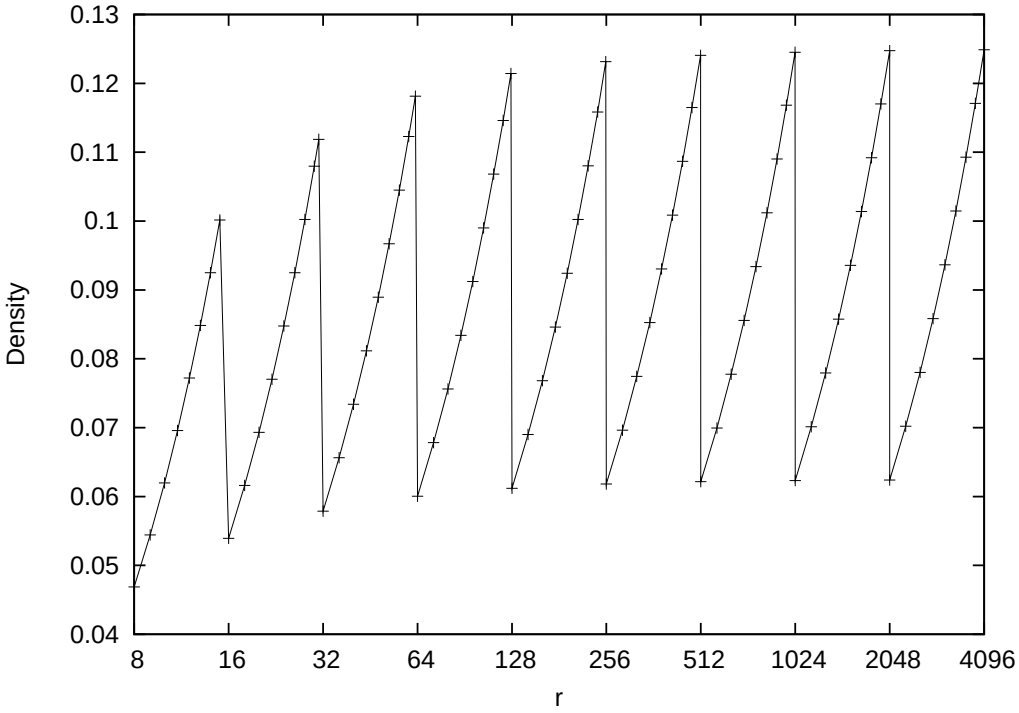

FIGURE 1: Bad census densities versus increasing values of r.

4.1 Results for census

The bad census for a given r divided by the corresponding value of P is referred to as the bad density, shown for various values of r in Figure 1. Clearly the upper bound on the density varies with the upper bound on the largest bad integer, which is controlled by the relationship between r and 2^g. When $r = 2^b$ the upper bound on the largest bad integer is $\frac{P}{8}$ and when $r = 2^{2 \cdot b - 1}$ the upper bound is $\frac{P}{4}$. As can be seen from the data, the bad census approaches the value of half the largest bad integer from below, giving densities that vary from $\frac{1}{16}$ to $\frac{1}{8}$ over each octave.

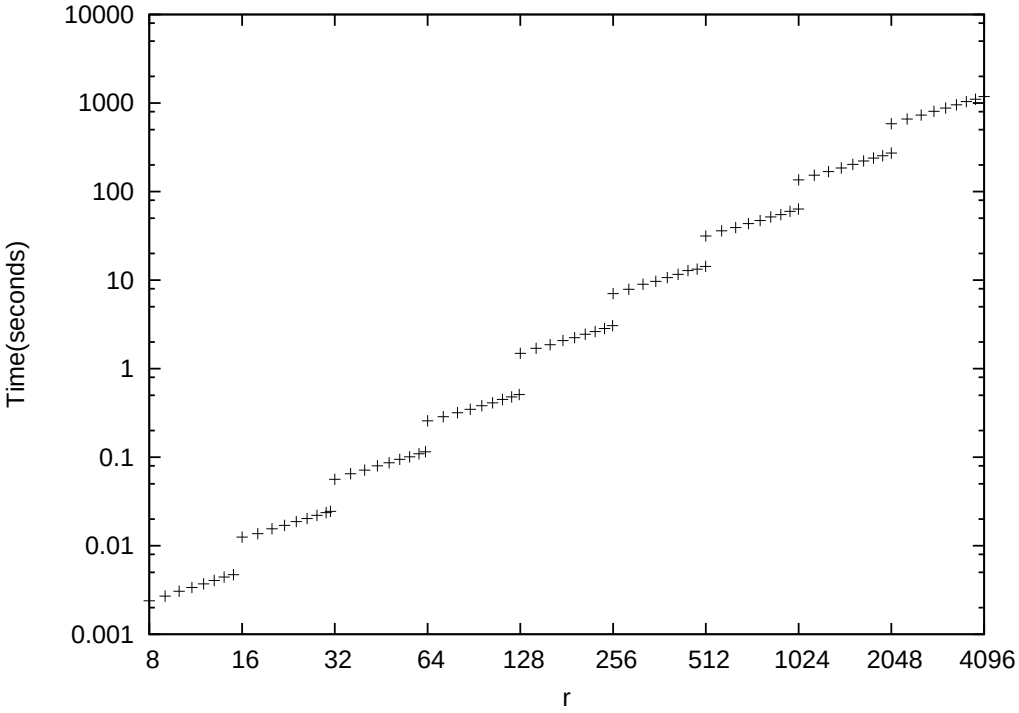

FIGURE 2: Log of running time of the census algorithm versus increasing values of r.

Figure 2 is a graph of the running time for computing the census for various values of r, which shows that the running time approximately doubles over the range of each octave and also doubles on the transition from one octave to another, as this is where the value of 2^g changes which causes the degree of the polynomials to double.

TABLE 1: Number of additions and multiplications with running times for convolutions of length n, with and without a four element convolution in the basis.

	Without 4 element convolution			With 4 element convolution		
n	adds	muls	time(s)	adds	muls	time(s)
4	$2.10 \cdot 10^{1}$	$6.00 \cdot 10^{0}$	$4.65 \cdot 10^{-6}$	$2.10 \cdot 10^{1}$	$6.00 \cdot 10^{0}$	$4.60 \cdot 10^{-6}$
8	$1.17 \cdot 10^{2}$	$1.80 \cdot 10^{1}$	$1.71 \cdot 10^{-5}$	$7.00 \cdot 10^{1}$	$1.50 \cdot 10^{1}$	$1.06 \cdot 10^{-5}$
16	$5.57 \cdot 10^{3}$	$6.60 \cdot 10^{2}$	$6.90 \cdot 10^{-5}$	$3.22 \cdot 10^{3}$	$5.10 \cdot 10^{2}$	$4.22 \cdot 10^{-5}$
32	$1.53 \cdot 10^{3}$	$1.62 \cdot 10^{2}$	$1.72 \cdot 10^{-4}$	$9.22 \cdot 10^{2}$	$1.23 \cdot 10^{2}$	$1.02 \cdot 10^{-4}$
64	$5.47 \cdot 10^{3}$	$5.46 \cdot 10^{2}$	$5.65 \cdot 10^{-4}$	$3.35 \cdot 10^{3}$	$4.11 \cdot 10^{2}$	$3.28 \cdot 10^{-4}$
128	$1.37 \cdot 10^{4}$	$1.31 \cdot 10^{3}$	$1.38 \cdot 10^{-3}$	$8.60 \cdot 10^{3}$	$9.87 \cdot 10^{2}$	$7.93 \cdot 10^{-4}$
256	$3.18 \cdot 10^{4}$	$2.85 \cdot 10^{3}$	$3.05 \cdot 10^{-3}$	$2.06 \cdot 10^{4}$	$2.14 \cdot 10^{3}$	$1.79 \cdot 10^{-3}$
512	$6.94 \cdot 10^{4}$	$5.92 \cdot 10^{3}$	$6.52 \cdot 10^{-3}$	$4.62 \cdot 10^{4}$	$4.44 \cdot 10^{3}$	$3.88 \cdot 10^{-3}$
1024	$2.08 \cdot 10^{5}$	$1.82 \cdot 10^{4}$	$1.95 \cdot 10^{-2}$	$1.37 \cdot 10^{5}$	$1.37 \cdot 10^{4}$	$1.15 \cdot 10^{-2}$
2048	$4.92 \cdot 10^{5}$	$4.28 \cdot 10^{4}$	$4.60 \cdot 10^{-2}$	$3.24 \cdot 10^{5}$	$3.21 \cdot 10^{4}$	$2.70 \cdot 10^{-2}$
4096	$1.08 \cdot 10^{6}$	$9.19 \cdot 10^{4}$	$1.00 \cdot 10^{-1}$	$7.24 \cdot 10^{5}$	$6.90 \cdot 10^{4}$	$5.98 \cdot 10^{-2}$
8192	$2.29 \cdot 10^{6}$	$1.90 \cdot 10^{5}$	$2.10 \cdot 10^{-1}$	$1.55 \cdot 10^{6}$	$1.43 \cdot 10^{5}$	$1.28 \cdot 10^{-1}$
16384	$4.91 \cdot 10^{6}$	$3.87 \cdot 10^{5}$	$4.43 \cdot 10^{-1}$	$3.39 \cdot 10^{6}$	$2.90 \cdot 10^{5}$	$2.74 \cdot 10^{-1}$
32768	$1.02 \cdot 10^{7}$	$7.80 \cdot 10^{5}$	$9.16 \cdot 10^{-1}$	$7.17 \cdot 10^{6}$	$5.85 \cdot 10^{5}$	$5.77 \cdot 10^{-1}$
65536	$2.13 \cdot 10^{7}$	$1.57 \cdot 10^{6}$	$1.89 \cdot 10^{0}$	$1.51 \cdot 10^{7}$	$1.17 \cdot 10^{6}$	$1.21 \cdot 10^{0}$

4.2 Results for general polynomial multiplication

Table 1 contains the number of arbitrary precision additions and multiplications used for polynomial multiplications modulo $z^n - 1$ for various values of n and their running times averaged over 100 samples. The coefficients are randomly generated eight bit integers. The running time is mainly influenced by n rather than the size of the coefficients, both in this example and in the census. The data is presented for the case where the recursion ends with a two element convolution and the case where the recursion ends with either a two element and a four element convolution.

5 Conclusion

The computation of the exact pseudorank census is a new result in Chinese Remainder Theorem, which gives some insight into the manner in which fixed precision approximations to the rank of a number in CRR fail. The algorithm underlying the theoretical result gives rise to a practical result when dealing with the implementation in that it presents a case in which the standard FFT fails to meet requirements for precisions and demonstrates the use of the Nussbaumer convolution as an effective alternative.

References

[1] G. Davida A. Chiu and B. Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications*, 35:259–275, 2001. doi:10.1051/ita:2001119 C710

[2] S. A. Cook and S. O. Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, Aug 1969. C718

[3] K. Culik and J. Kari. *Handbook of Formal Languages*, chapter 10, pages 599–616. Springer, 1997. C714

[4] G. Davida and B. Litow. Fast parallel arithmetic via modular representation. *SIAM J. Comp.*, 20,4:756–765, 1991. doi:10.1137/0220048 C710, C713

[5] J. Hee. Fast convolution using polynomial transforms. http://jenshee.dk/signalprocessing/polytrans.pdf, 2004. C711, C717

[6] W. Hesse. Division is in uniform tc0. In *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, pages 104–114, London, UK, 2001. Springer–Verlag. Also available as http://people.clarkson.edu/~whesse/div.ps. C710

[7] D. Knuth. *Seminumerical Algorithms*, volume 1 of *The Art of Computer Programming*, section 4.3.2. Addison–Wesley, third edition, 1997. C712

[8] W. Kuich and A. Salomaa. *Semirings,Automata,Languages*. Springer-Verlag, 1986. C714

[9] B. Litow and D. Laing. A census algorithm for chinese remainder pseudorank with experimental results. James Cook University, School of IT Tech Report http://www.cs.jcu.edu.au/ftp/pub/techreports/2005-3.pdf, 2005. C711, C714, C715

[10] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*. Springer, 1982. 2nd ed. C717, C718

[11] A. Salomaa and S. Soittola. *Automata Theoretic Aspects of Formal Power Series*. Springer–Verlag, 1978. C714

[12] M. P. Schützenberger. On a theorem of Jungen. *Proc. Am. Math. Soc.*, 13:885–890, 1962. C714

[13] R. Tanaka and N. Szabo. *Residue Arithmetic and its Application to Computer Technology*. McGraw–Hill, 1968. C712

[14] S. P. Tarasov and M. N. Vyalyi. Semidefinite programming and arithmetic circuit evaluation. Technical report, 2005. Available as http://arxiv.org/abs/cs/0512035v1. C713

[15] I. M. Vinagradov. *Elements of Number Theory*. Dover, 1954. C712

Author addresses

1. **D. Laing**, School of Maths, Physics and Information Technology, James Cook University, Townsville, Queensland 4811, Australia.
mailto:david.laing1@jcu.edu.au

2. **B. Litow**, School of Maths, Physics and Information Technology, James Cook University, Townsville, Queensland 4811, Australia.
mailto:bruce.litow@jcu.edu.au