RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe

Exact Solution of Linear Equation Systems over Rational Numbers by Parallel p-Adic Arithmetic

Carla LIMONGELLI, Roberto PIRASTU

(March 22, 1994)

RISC-Linz Report Series No. 94-25

Supported by: Austrian Forschungsförderungsfonds (FWF), project no.S5302-PHY (Parallel Symbolic Computation). CNR (National Research Council of Italy) under the Grant nr. 203.07.23. Commission of the European Communities in the framework of the program "Human Capital and Mobility", contract nr. ERBCHBICT930501.

Exact solution of linear systems over rational numbers by parallel p-adic arithmetic

Carla Limongelli* and Roberto Pirastu**

Research Institute for Symbolic Computation RISC-Linz
Johannes Kepler University, A-4040 Linz, Austria
e-mail: {limongel,rpirastu}@risc.uni-linz.ac.at

Abstract. We describe a parallel implementation of an algorithm for solving systems of linear equations over the field of rational numbers based on Gaussian elimination. The rationals are represented by truncated p-adic expansion. This approach permits us to do error free computations directly over the rationals without converting the system to an equivalent one over the integers. The parallelization is based on a multiple homomorphic image technique and the result is recovered by a parallel version of the Chinese remainder algorithm. Using a MIMD machine, we compare the proposed implementation with the classical modular arithmetic, showing that truncated p-adic arithmetic is a feasible tool for solving systems of linear equations. The proposed implementation leads to a speedup up to seven by ten processors with respect to the sequential implementation.

1 Introduction

For a positive integer n we want to solve a system of n linear equations for the n unknowns $x_1, \ldots, x_n$

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = b_2 \\ \qquad\qquad \ldots \qquad\qquad\qquad \ldots \\ a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n = b_n \end{cases} \tag{1}$$

where $a_{i,j}$ and b_i ($i = 1, \ldots, n$ and $j = 1, \ldots, n$) are rational numbers. We will denote the system (1) by $A\mathbf{x} = \mathbf{b}$.

Gaussian elimination is often used in numerical analysis to find approximations to the solutions of such systems. It is well known that for so-called ill-conditioned systems, small errors in the approximation of the coefficients may lead to large errors in the approximation of the solution. For instance, when a

* Supported by CNR (National Research Council of Italy) under Grant n. 203.07.23
** Supported by the Commission of the European Communities in the framework of the program "Human Capital and Mobility", contract Nr. ERBCHBICT930501

system based on floating point numbers attempts a division of a large dividend by a small divisor, the floating point result could be far from the exact result.

The use of exact arithmetic overcomes this problem. We will apply p-adic arithmetic to perform exact computations and we will compare this approach with the one based on modular arithmetic.

There exist mainly two possibilities to compute exact solutions for (1): either one first transforms the problem to the solution of a system over the integers, or one computes with rational numbers. One can compute a matrix A' and a vector $\mathbf{b}'$ with integer entries, such that the system $A'\mathbf{x} = \mathbf{b}'$ has the same solutions as (1). To $A'\mathbf{x} = \mathbf{b}'$ one can for instance apply Cramer's rule and obtain a solution avoiding rational arithmetic. This approach has the disadvantage that the entries in A' are in general considerably larger than the entries in A. On the other hand, working directly with system (1) needs an error-free representation of rational numbers and algorithms for error-free computations with them.

In this work we present a parallel implementation for solving linear systems, based on Gaussian elimination algorithm and the p-adic representation of rational numbers via truncated power series w.r.t. a prime basis p. The order of truncation r, as well as the number p, is chosen in accordance with an a priori estimation of the magnitude of the solution of the problem. This allows us to do error-free computations directly with rational numbers. For a detailed treatment of p-adic arithmetic in the context of symbolic computation, refer to [7] and [5].

Our goal is to show that p-adic arithmetic provides an efficient tool for solving linear systems over the rational numbers. For this reason, we compared our implementation with one using modular arithmetic and with a sequential implementation in the computer algebra system Maple [2].

Our parallelization consists of applying the well known Gaussian elimination method (see for instance [1]) for different prime bases, and recovering the result by the Chinese Remainder Algorithm (CRA). The implementation was done in PACLIB, a C-language library for parallel symbolic computation [6], on a Sequent parallel machine with a MIMD architecture.

In the following section we give some basics about p-adic arithmetic. In the third section the application of Gaussian elimination algorithm using p-adic arithmetic will be outlined. Fourth section will be devoted to describing the features of the parallel implementation. Concluding remarks are made in the fifth section.

2 Basics of p-adic arithmetic

For any positive integer m, we denote by $\mathbb{Z}_m$ the ring of the integers modulo m and by $|\cdot|_m$ the canonical ring homomorphism from $\mathbb{Z}$ to $\mathbb{Z}_m$. Let $\mathbb{N}$ be the set of natural numbers. For a given prime p, a rational number α can be represented in a unique way as

$$\alpha = (c/d) \cdot p^e \quad , \tag{2}$$

where c, d, and e are integers, c, d, and p pairwise relatively prime and d positive. Furthermore, α can be uniquely expressed in the following form:

$$\alpha = \sum_{i \geq e} a_i p^i \qquad \text{where} \qquad a_i \in \mathbb{Z}_p \ .$$

The infinite sequence $(a_e a_{e+1} \cdots a_{-1}\ a_0 a_1 \cdots)$ is called the *p-adic representation* of α. We use a *truncated representation*, defined as follows.

Definition 1 (Hensel Codes). Let p be prime and $r \in \mathbb{N}$. For any rational number $\alpha = (c/d) \cdot p^e$, where c, d and p are pairwise relatively prime, the *Hensel code* $H_{p,r}(\alpha)$ of length r of α is the pair

$$(mant_\alpha, exp_\alpha) = (a_0 a_1 \cdots a_{r-1}, e) \ ,$$

where the r leftmost digits of the p-adic representation of α and e are called the *mantissa* and the *exponent*, respectively.

One easily verifies that we have

$$|c \cdot d^{-1}|_{p^r} \ = \ \sum_{i=0}^{r-1} a_i \cdot p^i \in \mathbb{Z}_{p^r} \ .$$

Let $\mathbb{H}_{p,r}$ denote the set of all Hensel codes w.r.t. the prime p and the code length r, i.e., $\mathbb{H}_{p,r} := \{H_{p,r}(\alpha) \mid \alpha \in \mathbb{Q}\}$. The forward and the backward mappings between $\mathbb{Q}$ and $\mathbb{H}_{p,r}$ are algorithmically computed by the Extended Euclidean Algorithm (EEA), as we state in the following theorems.

Theorem 2 (Forward Mapping). *Let p be prime and $r \in \mathbb{N}$. Let $\alpha = (c/d)p^e$ be rational, such that c, d, and p are pairwise relatively prime. Then the mantissa $mant_\alpha$ of the code $H_{p,r}(\alpha)$ is computed by the EEA applied to p^r and d as*

$$mant_\alpha \equiv c \cdot y \qquad \pmod{p^r} \ ,$$

where y is the second output of the EEA.

Proof. See [11].

Definition 3 (Farey Fraction Set). Let $N(p, r) = \left\lfloor \sqrt{\frac{p^r - 1}{2}} \right\rfloor$. The *Farey fraction set* $\mathbb{F}_{p,r}$ of order $N(p, r)$ is the subset of rational numbers a/b such that:

$$a, b \in \mathbb{N}, \quad 0 \leq a \leq N(p, r), \quad 0 < b \leq N(p, r) \ .$$

Theorem 4 (Backward Mapping). *Let p be prime, $r \in \mathbb{N}$ and $c/d \in \mathbb{F}_{p,r}$. If m is the value in $\mathbb{Z}_{p^r}$ of the Hensel code mantissa related to c/d, then the EEA applied to p^r and m computes a finite sequence of pairs (x_i, y_i) such that $x_i/y_i = c/d$ for some i.*

Proof. See [11].

Arithmetic operations on Hensel codes are carried out, digit by digit, starting from the leftmost digit, as in the usual base-p arithmetic operations [3]. An addition (or a subtraction) can give a result in which some leftmost digits are equal to zero. In this case we say that the addition (or subtraction) produced a *pseudo-Hensel code*.

Definition 5 (Pseudo-Hensel codes). A *pseudo-Hensel code* is a code such that $a_0 = \cdots = a_k = 0$, for some k with $0 < k \leq r - 1$.

This loss of significative digits does not permit one to execute the division, as it has been stated in [5]. In [8] it is shown that it is possible to overcome this problem by introducing a new approach both for division and for the treatment of the pseudo-Hensel codes. These results extend the applicability of p-adic arithmetic to a wide class of computing problems.

In order to reduce the occurrences of pseudo-Hensel codes by choosing an appropriate base p, we remark that the probability of finding a leading zero in a code is equal to $1/(p-1)$. The probability of obtaining a leading zero after an addition of two Hensel codes is given by the probability of finding a (with $1 \leq a \leq p$) as leading digit of the first code and $p - a$ as the leading digit of the second code, that is $1/(p-1)^2$. The same occurs for subtraction. From a computational point of view, the best possible choice for p is hence made by taking p to be the greatest prime number less than the maximum integer representable in a memory word. On the other way we want to avoid overflow during computations, hence we will fix the base p on the ground of the wordsize w of our computer: $p \leq 2^{w/2} + 1$.

The possibility of performing exact arithmetic operations on $\mathbb{H}_{p,r}$ is ensured by the following theorem.

Theorem 6. *Let p be prime, $r \in \mathbb{N}$, $\alpha_1, \alpha_2 \in \mathbb{Q}$, $\Phi \in \{+, -, \cdot, /\}$ an arithmetic operator. If*

$$\alpha_1 \Phi \alpha_2 = \alpha_3, \qquad \text{with } \alpha_3 \in \mathbb{F}_{p,r}$$

then there exists precisely one $H_{p,r}(\alpha_3)$ such that

$$H_{p,r}(\alpha_3) = H_{p,r}(\alpha_1) \Phi' H_{p,r}(\alpha_2)$$

where Φ' is the operator in $\mathbb{H}_{p,r}$ which corresponds to Φ in $\mathbb{Q}$.

Proof. See Theorem 2, Theorem 4 and [5].

A scheme for a general computation consists of mapping on $\mathbb{H}_{p,r}$ the rational input numbers and then of performing the computations over $\mathbb{H}_{p,r}$. However, by Theorem 4, the inverse mapping can be performed only when the expected result belongs to $\mathbb{F}_{p,r}$. This means that we need a bound for the size of the result, in order to make the right choice for p and r.

3 Bounds for the Solutions

Before showing the parallel implementation in the next session, we present the sequential method of Gaussian elimination and an estimate for the size of the result. The problem is stated as follows.

Problem *Given* $A \in \mathbb{Q}^{n\times n}$ *and* $\mathbf{b} \in \mathbb{Q}^n$*, find, if it exists, a vector* $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Q}^n$ *such that*

$$A\mathbf{x} = \mathbf{b} \ . \tag{3}$$

We consider the case $A \in \mathbb{Z}^{n\times n}$ and $\mathbf{b} \in \mathbb{Z}^n$ first. By Cramer's rule we know that

$$x_i = \frac{|A_i|}{|A|} \ , \tag{4}$$

where $|A|$ denotes the determinant of A, and A_i is the matrix obtained from A by substituting the ith column by $\mathbf{b}$. Now let a be a maximal entry in a matrix $M \in \mathbb{Z}^{n\times n}$, then one easily proves by induction on n that $|M| \leq n!a^n$. From this and (4) we obtain that both numerator and denominator of any x_i are bounded by $n!a^n$, where a is now a maximal entry in A and $\mathbf{b}$. From this bound we determine a value for r, such that the result is in $\mathbb{F}_{p,r}$ for a given prime p. From the definition it suffices that

$$n!a^n \leq \left\lfloor \sqrt{\frac{p^r - 1}{2}} \right\rfloor \ . \tag{5}$$

Considering the square of both sides of the inequality we obtain $2(n!a^n)^2 + 1 \leq p^r$. This implies $\log_p(2(n!a^n)^2 + 1) \leq \log_p p^r$ or, equivalently,

$$r \geq \log_p(2(n!a^n)^2 + 1) \ . \tag{6}$$

Hadamard's inequality (see for instance [10] or [9]) gives another bound for the determinant

$$|A|^2 \leq \prod_{i=1}^{n} \left(\sum_{j=1}^{n} a_{i,j}^2 \right)^{1/2} \ . \tag{7}$$

From this bound the following condition is derived in [5]

$$p^r \leq \sum_{i=1}^{n} |b_i| \prod_{i=1}^{n} \left(\sum_{j=1}^{n} a_{i,j}^2 \right) \ . \tag{8}$$

It should be remarked that both bounds are still quite conservative, since a smaller choice of p and r is often sufficient. In the general case $A \in \mathbb{Q}^{n\times n}$ and $\mathbf{b} \in \mathbb{Q}^n$ the bound for the numerator and denominator of the x_i's becomes $n!a^{n(n+1)}$. This follows again from Cramer's rule by considering the equivalent system obtained from A by multiplying each row by the common denominator of all entries in that row and of the ith entry in $\mathbf{b}$, i.e., multiplying by a number of magnitude at most a^{n+1}.

4 Parallel Implementation

We describe the parallelization on, say, k concurrent processors. We first compute k prime numbers $p_1, \ldots, p_k$ at random and the corresponding code length r such that the entries of the solution $\mathbf{x}$ are contained in $\mathbb{F}_{g,r}$, where $g = p_1 \cdots p_k$.

At this point k parallel tasks are started. Each of them computes the image of the problem w.r.t. one prime in p-adic representation of the rational entries, i.e., $H_{p_i,r}(A)$ and $H_{p_i,r}(\mathbf{b})$. By a certain abuse of notation we denote by $H_{p_i,r}(A)$ the matrix $(\tilde{a}_{i,j})$ with $\tilde{a}_{i,j} = H_{p_i,r}(a_{i,j})$, and analogously for $H_{p_i,r}(\mathbf{b})$.

Then for each processor a sequential implementation of Gaussian elimination is executed via p-adic arithmetic. The main steps of this algorithm are given in Fig. 1.

Gaussian Elimination for p-adic Computations

Input: n: degree of the linear system;
$A = (a_{i,j}) \in \mathbb{Q}^{n\times n}$: n-dimensional square matrix;
$\mathbf{b} = (a_{1,n+1}, \ldots, a_{n,n+1}) \in \mathbb{Q}^n$: column vector;
p: prime base;

Output: $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Q}^n$: solution of $A\mathbf{x} = \mathbf{b}$, if it exists;

Begin

1.1. Compute the maximum integer number a among the numerators and denominators of the rational entries in A and $\mathbf{b}$;
Compute the truncation order r, as shown in (6);

1.2. Apply the mapping $H_{p,r}$ to all the entries of A and $\mathbf{b}$.

1.3. $l := 0$;
for $i := 1$ **to** n **do**
  $l := l + 1$;
  for $j := i$ **to** $n + 1$ **do**
    Divide $a_{i,j}$ by $a_{i,i}$
  od;
  for $h := l + 1$ **to** n **do**
    Multiply the i-th row by $a_{h,l}$;
    Subtract the new i-th row from the h-th row of the system obtained at last iteration. This is the new h-th row;
  od;
od;

1.4. Recover $H_{p,r}(x_1), \ldots, H_{p,r}(x_n)$ from the triangular matrix obtained;

End

Fig. 1. Gaussian elimination Algorithm for p-adic Computations

Gaussian elimination computes solutions $\mathbf{x}^{(i)} \in \mathbb{H}^n_{p_i,r}$ for $i = 1, \ldots, k$. After collecting all of the $\mathbf{x}^{(i)}$ we execute k concurrent calls of CRA. We apply the parallel version of CRA described in [8] on each sequence of components $x_j^{(1)}, \ldots, x_j^{(k)}$, obtaining the component $x_j \in \mathbb{H}_{p_1 \cdots p_k, r}$ of the solution vector $\mathbf{x}$.

From the assumptions made on r, the list $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\}$ of results obtained in this way can be mapped back to a vector $\mathbf{x} \in \mathbb{F}_{g,r}^n$ (vector over the Farey fraction set) by the CRA. From Theorem 6 we know that if the solution exists in $\mathbb{F}_{g,r}$, then it is unique.

After this, the result $\mathbf{x}$ only needs to be converted from the p-adic to the usual representation by the backward mapping, applied in parallel on each component.

A scheme of the parallel computation is given in Fig. 2.

INPUT		$A, \mathbf{b}$	
PRIME COMPUTATION		$\downarrow$	
		$p_1, p_2, \ldots, p_k$	
FORWARD MAPPING	$\swarrow$	$\downarrow$	$\searrow$
HENSEL CODES	$H_{p_1,r}(A), H_{p_1,r}(\mathbf{b})$	$\ldots$	$H_{p_k,r}(A), H_{p_k,r}(\mathbf{b})$
	$\downarrow$	$\downarrow$	$\downarrow$
GAUSS ALGORITHM	$\downarrow$	$\downarrow$	$\downarrow$
	$\downarrow$	$\downarrow$	$\downarrow$
SOLUTIONS	$\mathbf{x}^{(1)} = H_{p_1,r}(\mathbf{x})$	$\ldots$	$\mathbf{x}^{(k)} = H_{p_k,r}(\mathbf{x})$
SYNCHRONIZATION	$\searrow$	$\downarrow$	$\swarrow$
		$\downarrow$	
		$\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}$	
	$\swarrow$	$\downarrow$	$\searrow$
	$\swarrow$	$\downarrow$	$\searrow$
	$\mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_1^{(k)}$	$\ldots$	$\mathbf{x}_n^{(1)}, \ldots, \mathbf{x}_n^{(k)}$
CRA	$\downarrow$	$\downarrow$	$\downarrow$
UNIQUE SOLUTION	$\mathbf{x}_1 \in \mathbb{H}_{p_1 \cdots p_k, r}$	$\ldots$	$\mathbf{x}_n \in \mathbb{H}_{p_1 \cdots p_k, r}$
INVERSE MAPPING	$\searrow$	$\downarrow$	$\swarrow$
		$\downarrow$	
OUTPUT		$\mathbf{x}$	

Fig. 2. Parallel Computation Scheme

In the case of dense matrices with large dimension and size w.r.t. the number of processors, also standard parallelization techniques for dense linear systems could be applied.

We implemented this algorithm in PACLIB (see [6]), a system developed at RISC-Linz for computer algebra purposes in particular. PACLIB is based on the SACLIB library (see [4]), which provides several computer algebra algorithms written in C. It should be noticed that several other symbolic computation systems provide a parallel implementation of a linear system solver. For instance the parallel computer algebra system ||MAPLE|| (see [13]), a parallel version of Maple developed at RISC-Linz, provides a solver based on the Gauss-Jordan algorithm that can also handle symbolic entries [12].

5 Experimental Results and Comparisons

We tested the implementation on several randomly generated linear systems on a Sequent Symmetry machine with 20 processors, a MIMD computer with shared memory.

The parallel implementation is compared with a sequential implementation, where we apply sequentially the same mapping onto $\mathbb{H}_{p_i,r}$ for the same primes p_i as in the corresponding parallel execution.

In Table 1 the execution times of both the sequential and the parallel implementation are reported in milliseconds. The input size is the maximum bit length of the numbers. If the entries are rational numbers the numerator and the denominator have a bit length bounded by this input size. Parallelizing the algorithm over 10 processors we achieve a speedup up to 7.5, with respect to the sequential algorithm.

Dimension	Input Size	Sequential	Parallel	Speedup
10	10	3064	692	4.4
15	10	9388	1792	5.2
20	10	27707	4421	6.2
20	20	57014	7563	7.5
20	30	69481	11196	6.2
25	10	61528	8751	7.0
25	20	108695	15813	6.8
30	10	119890	16893	7.0

Table 1. Comparison of sequential and parallel algorithm

We compared our implementation with an efficient modular parallel implementation for systems over integers which makes use of a mixed method (Gauss and Kramer), implemented in PACLIB. We consider two cases:

1. The input data are integers;
2. The input data are rationals. We present the case where only the vector **b** has rational entries.

In the first case, p-adic arithmetic is essentially reduced to modular arithmetic and the backward mapping becomes almost trivial. From Cramer's rule we know that the denominator of any component x_i of the solution $\mathbf{x}$ divides $|A|$. So

$H_{g,r}(x_i) \cdot H_{g,r}(|A|)$ is an integer and no backward mapping is needed, since we have

$$x_i = \frac{/H_{g,r}(x_i)H_{g,r}(|A|)/_g}{|A|} \quad , \tag{9}$$

where $/\cdot/_g$ is the *signed* modulo mapping to $\{-\frac{g-1}{2}, \ldots, \frac{g-1}{2}\}$. Remark that the determinant $|A|$ is computed as a by-product of Gaussian elimination.

In the second case, in order to do a fair comparison with the modular algorithm, which accepts only integers entries, we have to study the size of equivalent inputs for both algorithms. Let $A \in \mathbb{Q}^{n\times n}$ be a system over the rational numbers and let s be the maximal size among all denominators of the entries in A and $\mathbf{b}$. We must transform $A\mathbf{x} = \mathbf{b}$ to an equivalent system $A'\mathbf{x} = \mathbf{b}'$ with integer entries. This means to multiply each row/equation by an appropriate integer. It is easy to see that the smallest integer m_j such that $m_j\mathbf{a}_j, m_jb_j$ are all integers, is equal to the l.c.m. over all denominators arising in the jth row $\mathbf{a}_j$ (resp. b_j) of A (resp. $\mathbf{b}$). In the worst case, m_j will be the product of all denominators (i.e., $m \leq (n+1)s$). In the comparison, one must take into account this fact, although the average size of m will be probably smaller than this.

Here we are considering rational entries only in $\mathbf{b}$, so if s is the size of the entries for the p-adic algorithm, the entries of the modular algorithm will be of size $2s$.

In Table 2 we show the behaviour of the algorithms for fixed input length. For the considerations stated above a length of 10 for integer entries means a length of 5 for the rational case.

Dimension	Modular	P-adic	Rational
5	321	218	278
10	692	619	718
15	1422	1719	1315
20	3210	3315	2995
25	5972	5756	5579
30	14309	12299	11107

Table 2. Comparison of modular, p-adic, and rational p-adic when length=10.

In Table 3 the timings of some executions are shown, where the dimension of the system is 20.

We also compare our sequential algorithm with the implementation available in MapleV (see [2]). MapleV implements a fraction free Gaussian elimination, so the equations are converted to have integer coefficients and after each elimination step, the greatest common divisor of the coefficients is divided out to minimize growth. The timings in Fig. 3 show the behaviour of the algorithms.

Input length	Modular	P-adic	Rational
5	1908	2024	1722
10	3007	2966	2903
15	3519	3957	3416
20	5760	5441	4421
25	7502	8785	7067
30	9845	11151	9037
35	11222	11734	10023
40	22406	21372	11023

Table 3. Comparison of modular, p-adic, and rational p-adic when dim. =20.

6 Conclusions

We presented a parallel implementation for solving systems of linear equations over the rational numbers. For the computations, the rational coefficients are represented by Hensel codes, i.e., by a truncated p-adic representation. This approach permits us to do error-free computations directly over the field of rational numbers, without converting the system to an equivalent problem over the integers. The parallelization is done by applying the sequential Gaussian

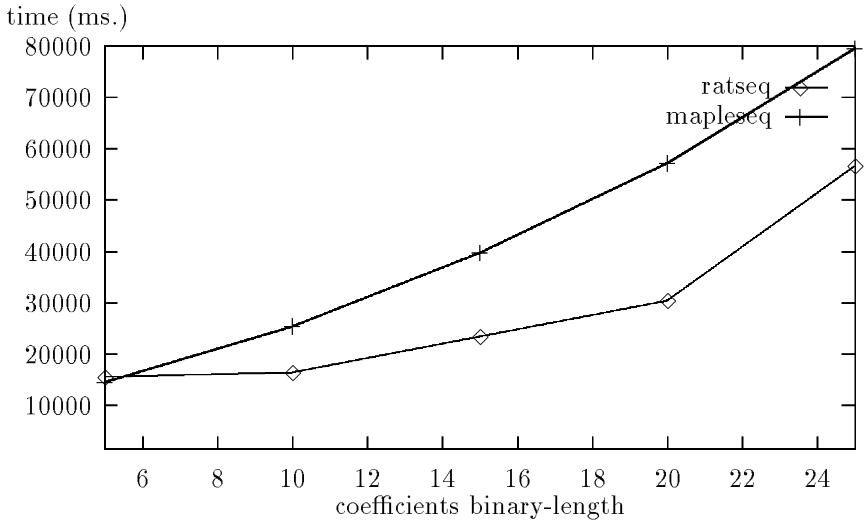

Fig. 3. Comparison of MapleV and the sequential p-adic algorithm with dim. =20.

elimination to different p-adic images of the problem w.r.t. several prime bases, using the p-adic arithmetic described in [7]. The result is recovered by a parallel Chinese remainder algorithm. Using 10 processors the parallel implementation achieves a speedup up to 7.5 with respect to the sequential one.

We compared our implementation with a parallel modular approach over the integers and with the built-in sequential implementation in MapleV. As expected, the p-adic representation is at least as efficient as the modular one for the case of integer coefficients and more efficient for the case of rational coefficients.

These experimental data confirm the expected behaviour of linear algebra algorithms implemented via p-adic arithmetic as regards the heavy computational complexity of CRA. In [8] it is shown that for problems with many large input data the asymptotic running time of the p-adic algorithm is never dominated by the cost of the recovering step.

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley Publishing Company, 1975.
2. Bruce W. Char, Gregory J. Fee, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt. A tutorial introduction to MAPLE. *Journal of Symbolic Computation*, Vol.2, n.2, 1986.
3. Attilio Colagrossi, Carla Limongelli, and Alfonso Miola. Scientific computation by error-free arithmetics. *Journal of information Science and Technology*, July-October 1993.
4. George E. Collins et al. A SACLIB 1.1 user's guide. Technical report, RISC-Linz, Johannes Kepler University, Austria, 1993. RISC-Linz Report Series n. 93-19.
5. Robert T. Gregory and Edayathumangalam V. Krishnamurthy. *Methods and Applications of Error-Free Computation.* Springer Verlag, 1984.
6. Hoon Hong et al. A PACLIB user manual. Technical report, RISC-Linz, Johannes Kepler University, Austria, 1992. RISC-Linz Report Series n. 92-32.
7. Carla Limongelli. *The Integration of Symbolic and Numeric Computation by p-adic Construction Methods.* PhD thesis, Universitá degli Studi di Roma "La Sapienza", Italy, 1993.
8. Carla Limongelli. On an efficient algorithm for big rational number computations by parallel p-adics. *Journal of Symbolic Computation*, Vol.15, n.2, 1993.
9. Maurice Mignotte. Some useful bounds. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra Symbolic and Algebraic Computation*, pages 259–263. Springer Verlag, 1983.
10. Henryk Minc and Marvin Marcus. *Introduction to Linear Algebra.* Macmillan, New York, 1965.
11. Alfonso Miola. Algebraic approach to p-adic conversion of rational numbers. *Information Processing Letters*, 18:167–171, 1984.
12. Roberto Pirastu and Kurt Siegl. Parallel computation and indefinite summation: A ||MAPLE|| application for the rational case. submitted to Journal of Symbolic Computation, 1994.

13. Kurt Siegl. Parallelizing algorithms for symbolic computation using ‖MAPLE‖. In *Fourth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, San Diego*, pages 179–186, 1993.