J. Symbolic Computation (1993) **15**, 181–197

On an Efficient Algorithm for Big Rational Number Computations by Parallel p-adics†

CARLA LIMONGELLI

Dipartimento di Informatica e Sistemistica

Università "La Sapienza"

Via Salaria 113, 00198 Roma, Italy

E-mail: limongel@disco1.ing.uniroma1.it

(Received 14 September 1992)

This paper presents an algorithm for evaluating an arithmetic expression over "big" rational numbers. The method exploits p-adic arithmetic and parallelism to achieve efficiency.

Roughly, the algorithm begins by mapping the input rational numbers to the related p-adic codes for several prime bases, then the corresponding expression is evaluated over these codes and finally the rational output is recovered from the resulting codes. All these three steps are parallelized.

An efficient algorithm to compute the recovery step is proposed. Let p be the maximum of the prime bases for the codes, let r be their truncation order and let k be the number of processors. The asymptotic bit-wise complexity of the proposed recovery step is $O(r\ (k\ log\ p)^{log_2 3})$. The previous known bound was $O(r^2 (k\ log\ p)^{log_2 3})$.

Let n be the number of operations in the given expression: the asymptotic bit-wise complexity of the proposed algorithm for evaluating arithmetic expressions is $O(n\ r^2\ log\ p)$.

1. Introduction

In this paper we propose a parallel algorithm for evaluating "arithmetic expressions" over "big" rational numbers. By "arithmetic expression" we mean any kind of expression involving the four basic arithmetic operators $+$, $\cdot$, $-$, $/$, over the rational numbers, for example:

$$\frac{(x+y)\cdot z}{x} + \frac{2}{z}, \quad \text{with } x, y, z \in \mathbf{Q}.$$

By "big" integer number we mean a number which is greater than the maximum integer that can be stored in a word of a given computer. We define a rational number (in reduced form) a/b to be big when a or b (or both) is a big integer.

† This work has been partially supported by MURST and by CNR under the Project "Sistemi Informatici e Calcolo Parallelo", contract n. 92.01604.69.

0747-7171/93/020181+17 $08.00/0

The reason of our interest in big numbers manipulations is in that they often arise as intermediate or final results in various algebraic algorithms such as Gröbner Bases (Buchberger B. (1970)), Cylindrical Algebraic Decomposition (Collins G. E. (1975)), Characteristic Sets (Wang D. (1991)), Polynomial Remainder113 Sequences (Loos R. (1983)), Exponentiation and Gaussian Elimination (Knuth D. (1981)).

The classical algorithms for rational arithmetic operations described in Knuth D. (1981), are not suitable for big numbers, due to *GCD* computation, so we propose to handle them starting by *p*-adics and parallelism. Below we give a brief sketch of the proposed approach.

We assume to have an input arithmetic expression over m rational numbers $q_1, \ldots, q_m$. Our goal is to evaluate it, obtaining a rational result q. We proceed in three main steps (see also Figure 1):

- Each rational number q_h $(h = 1 \ldots m)$ is mapped via a prime bases p_i to its related Hensel codes $\bar{q}_h^{(i)}$ $(i = 1 \ldots k)$, with a certain truncation order r. All the mappings are done in parallel.

  Let us note that the truncation order, as well as the bases p_i, are chosen according to a previous estimate of the problem solution. An apt Farey fraction set $\mathbf{F}_{p,r} \subset \mathbf{Q}$, which the rational solution will belong to, must be identified and the appropriate choice for p and r is made accordingly.

  Such an estimate can be evaluated once a given algorithm is stated for the solution of the problem. This can turn out to be a difficult problem, but such an estimate can be computed on the basis of the number of operations that must be performed in order to reach the rational solution.

  For example, it is well known that the determinant $D(A)$ of a given N-dimensional square matrix $A = \{a_{i,j}\}_{1 \le i,j \le N}$ is bounded by $N! \cdot a^N$, where $a = max\{|\ a_{i,j}\ |\}_{1 \le i,j \le N}$. It is also easy to compute in advance the maximum coefficient which arises during polynomial multiplication: given the polynomials $\sum_{i=0}^{d_a} a_i \cdot x^i$ and $\sum_{j=0}^{d_b} b_j \cdot x^j$, if $a = max\{|\ a_i\ |\}_{1 \le i \le d_a}$, $b = max\{|\ b_j\ |\}_{1 \le j \le d_b}$ and $c = max\{a, b\}$, then the largest coefficient of the resulting polynomial is bounded by $max\{d_a, d_b\} \cdot c^2$.

- The arithmetic expression is evaluated over the $\bar{q}_h^{(i)} \in \mathbf{H}(p_i, r)$ (sets of Hensel codes).

  The image problem is solved in parallel over k finite fields $\mathbf{H}(p_i, r)$ whose elements are the coefficients of the truncated power series with base p_i and approximation r.

- The unique solution $\bar{q} \in \mathbf{H}(p_1 \cdot \ldots \cdot p_k, r)$ is recovered from the k code results $\bar{q}^{(1)} \in \mathbf{H}(p_1, r), \ldots, \bar{q}^{(k)} \in \mathbf{H}(p_k, r)$, by applying the Chinese Remainder Algorithm (*CRA*); then the rational output $\bar{q} \in \mathbf{F}_{p,r}$ is obtained via the backward mapping algorithm. The *CRA* is parallelized.

  Let us note that the correspondence between $\mathbf{F}_{p,r}$ and $\mathbf{H}(p, r)$ is injective, hence if the solution does exist in $\mathbf{F}_{p,r}$, then it is unique.

Actually the crucial step of this approach is the detection of the unique solution via the *CRA*.

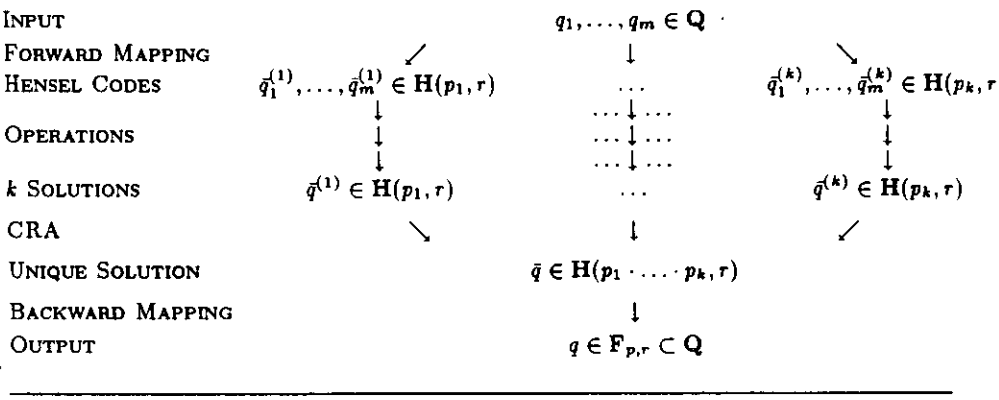

Figure 1: General schema of parallel p-adic computations.

In Colagrossi A., Limongelli C. (1988), an algorithm has been devised to perform the recovery step of the unique p-adic solution, with an asymptotic computational complexity of $O_B(r^2(k\ log\ p)^{log_2 3})$, where O_B indicates, as usual, the order of magnitude under the bitwise computational model described in Aho, Hopcroft, Ullman (1975).

Here we are going to show how to improve this result. The improved algorithm proposed in this paper is obtained essentially by parallelizing some steps which occur in the backward mapping of the former algorithm, i.e. CRA and multiplication. The resulting asymptotical computational complexity is $O_B(r\ (k\ log\ p)^{log_2 3})$. It is also proved, here, that the updating of the whole p-adic parallel approach by this new algorithm offers a considerable speed-up in terms of computing time, especially when problems are considered in which the maximum size of the input data is small w.r.t. the output data.

In order to evaluate the main result of this paper we have to stress that while in Colagrossi A., Limongelli C. (1988), the CRA step could eventually dominate, in terms of time, the rest of the computation, the new one is such that its computational complexity never dominates the asymptotical behaviour of the whole algorithm. Thus, the new algorithm provides a better behaviour when it is applied to problems where the number n of operations needed to reach the solution is at most m^2, where m is the number of input rational numbers involved. This class of problems is actually wide: it contains for example a large set of operations over matrices and over polynomials. Some examples of such problems will be discussed in the last section.

The plan of the paper is as follows: section 2 summarizes the basics of p-adic arithmetic and can be skipped by a reader familiar with it. In section 3 the general schema for p-adic arithmetic computations is presented in order to support the algorithmic descriptions that follow. The schema of parallel p-adic computation is described in section 4. Section 5 presents the enhanced parallel algorithm and the analysis of its improved computational complexity. An evaluation of the obtained computational results is discussed in section 6.

2. Basics of p-adic Arithmetic

In this Section, we summarize some of the basic concepts of p-adic arithmetic together with the theoretical complexity of the algorithms to be used in the following computational analysis. A much more detailed presentation of p-adic arithmetic and of Hensel codes representation can be found in Koblitz N. (1977) and in Gregory R.T., Krishnamurthy E.V. (1984).

A rational number $\alpha = a/b$, can always be expressed in a normalized form as

$$\alpha = \frac{c}{d} \cdot p^e$$

where e is an integer; p is a prime number; c, d and p are pairwise relatively prime integers.

Every rational number α can be uniquely expressed in the following form, called the p-adic expansion of α:

$$\alpha = \sum_{i=e}^{\infty} a_i p^i; \quad a_i \in \mathbf{Z}_p; \quad e \in \mathbf{Z}; \quad \|\alpha\|_p = p^{-e}; \quad a_e \neq 0.$$

A finite length p-adic number representation and its arithmetic has to be defined on the so called Hensel codes, as follows:

DEFINITION 2.1. *Given a prime number p, a Hensel code of length r, of a normalized rational number $\alpha = (c/d) \cdot p^e$ is a pair*

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

where the r leftmost digits and the value e of the related p-adic expansion are called the mantissa and the exponent, respectively.

Let $\mathbf{H}(p, r)$ indicate the Hensel's codes set w.r.t. the prime p and the approximation r and let $H(p, r, \alpha)$ indicate the Hensel code representation of the rational number $\alpha = (a/b) \cdot p^e$ w.r.t. the prime p and the approximation r.

DEFINITION 2.2. *The Farey fraction set $\mathbf{F}_{p,r}$ is the subset of rational numbers a/b such that:*

$$a, b \in \mathbf{N}, \quad 0 \leq a \leq N, \quad 0 < b \leq N, \quad N = \left\lfloor \sqrt{\frac{p^r - 1}{2}} \right\rfloor$$

The forward and the backward mappings between rational numbers and Hensel codes can then be defined on the basis of the following theorems.

THEOREM 2.1. (FORWARD MAPPING) *Given a prime p, an integer r and a rational number $\alpha = (c/d) \cdot p^e$, such that $GCD(c, p) = GCD(d, p) = 1$, the mantissa $mant_\alpha$ of the code related to the rational number α, is computed by the Extended Euclidean Algorithm (EEA) applied to p^r and d as:*

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

where y is the second output of the EEA.

PROOF. See Miola, A. (1984). □

THEOREM 2.2. (COMPUTATIONAL COMPLEXITY OF THE FORWARD MAPPING) *The forward mapping algorithm given in* THEOREM 2.1, *applied to two n-bit numbers, requires* $O_B(M(n) \log n)$ *time, where* $M(n)$ *is the time required to multiply two n-bit numbers.*

PROOF. See Aho, Hopcroft, Ullman (1975). □

THEOREM 2.3. (BACKWARD MAPPING) *Given a prime* p, *an integer* r, *a positive integer* $m \leq p^r$ *and a rational number* c/d *such that* $GCD(c,p) = GCD(d,p) = 1$, $c/d \in \mathbf{F}_{p,r}$, *and* m *is the value in* $\mathbf{Z}_p^r$ *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 there exists an index* i *for which* $x_i/y_i = c/d$.

PROOF. See Miola A. (1984). □

THEOREM 2.4. (COMPUTATIONAL COMPLEXITY OF BACKWARD MAPPING) *The sequential backward mapping algorithm given in* THEOREM 2.3, *applied to two n-bit numbers, requires* $O_B(M(n) \log n)$ *time.*

PROOF. See Aho, Hopcroft, Ullman (1975). □

Let us consider now the algebra on Hensel codes, and the related algorithms, as given in Gregory R.T., Krishnamurthy E.V. (1984); see howewer the corrections in Dittemberger K. (1985). The following theorem holds:

THEOREM 2.5. (COMPUTATIONAL COMPLEXITY OF ARITHMETIC OPERATIONS) *Given two rational numbers* α, β *and given a prime number* p *and an approximation* r, *the computational complexity of the operations* $+, -, \cdot, /$, *under the bitwise computational model, between* $H(p,r,\alpha)$ *and* $H(p,r,\beta)$ *is:*
$+, -$: $O_B(r \log p)$;
$\cdot, /$: $O_B(r^2 \log p)$.

PROOF. See Limongelli C. (1987). □

3. Computing with p-adic Arithmetic

Let us consider the problem P involving the computation of a sequence of n arithmetic operations over a given set of m big rational numbers a_i/b_i, $i = 1, \ldots, m$: the p-adic arithmetic can be used to solve this problem once the size of the result is estimated.

Before showing the main steps of the algorithm which solves the problem P, let us state the following assumptions:

(i) on the basis of the estimated size of the result for P, the values p and r have to be chosen such that this rational result belongs to $\mathbf{F}_{p,r} \subset \mathbf{Q}$;

(ii) let s be the size of $max(a_1, \ldots, a_m, b_1, \ldots, b_m)$, where $\alpha_h = a_h/b_h$, $(h = 1, \ldots, m)$ are the m rational input numbers of the given problem P;

(iii) the solution of the problem P requires n arithmetic operations on big rational numbers.

A trivial model for representing the work needed in solving a general problem P, can be sketched in the following few lines. Let us assume that $I_P = \{\alpha_1, \ldots, \alpha_m\}$ is the set of rational numbers given as the input for solving P (where $\alpha_h = a_h/b_h$, $h = 1, \ldots, m$); let OP_P be the set of the operators on $\mathbf{Q}$ in assumption (iii): $OP_P = \{op_1, \ldots, op_n\}$.

We can define the set C_P of all the values involved in the computations (including also the result of P) as $C_P = I_P \cup O_P$, where the set O_P collects all the intermediate values occurring during the execution of the operations in OP_P and eventually the solution of the problem. So the general schema for the solution of P consists of the execution of the operations in OP_P, over elements of C_P, while adjoining in turn each new values γ to C_P.

$$\gamma = \alpha \; op_l \; \beta, \quad \alpha, \beta \in C_P, \quad l = 1, \ldots, n.$$

We can distinguish four main classes of problems, depending on the size of the input data and rational result: the first class is constituted by those problems which have big input numbers and a big output result. Problems in the second class have small input numbers but a big output; problems in the third class start from big input numbers giving small output; problems in the last class have both small input and output data.

We are interested in the first two classes of problems because the others classes can be easily faced by the use of the classic p-adic arithmetic described in Colagrossi A., Limongelli C., Miola A. (1990).

As far as the first two classes are concerned, we must appropriately choose p, according to the assumption (i). In this case the term "appropriately" means that we want to avoid overflow during the computations. Hence we fix the base p on the ground of the wordsize w of our computer:

$$p \leq 2^{\frac{w}{4}} + 1; \tag{3.1}$$

consequently we determine the approximation $\bar{r}$ as follows (see DEFINITION 2.2):

$$\bar{r} = \lceil log_2 p(2\ N^2 + 1) \rceil. \tag{3.2}$$

The algorithm is described in Figure 2. In this algorithm, $OP_p^{\mathbf{H}(p,\bar{r})}$ denotes the set of the image operations in $\mathbf{H}(p, \bar{r})$ of the operations in OP_P, and $C_p^{\mathbf{H}(p,\bar{r})}$ is the image set of C_P.

In order to analyze the computational complexity of the steps of the algorithm and in order to distinguish between the previous classes of problems, let us state further assumptions:

(iv.1) $s = \bar{r}\ log_2 p$ (the input data are big rational numbers, i.e. P is in the first class);

(iv.2) $s < log_2 p$ (the input data are not big rational numbers, while the intermediate computations involve big numbers and also the output is a big rational number, i.e. P is in the second class);

(v) each arithmetic operation in the p-adic domain has maximal computational complexity $O_B(\bar{r}^2\ log\ p)$, according to the THEOREM 2.5.

Moreover, in our computations we use the sequential Karatsuba algorithm to multiply

SEQUENTIAL ALGORITHM

Input: p: prime number,
$\overline{r}$: code length, found by relation (3.2),
$I_P = \{\alpha_h = a_h/b_h,\ h = 1,\ldots,m\}$: m rational input numbers;
Output: the rational solution $a/b \in \mathbf{F}_{p,\overline{r}}$;

S.1 Each rational number $\alpha_h \in I_P$, is mapped into the related Hensel code. Let the related set of input data be $I_P^{\mathbf{H}(p,\overline{r})} = \{(mant_{\alpha_h}, exp_{\alpha_h}),\ h = 1,\ldots,m\}$, where $(mant_{\alpha_h}, exp_{\alpha_h})$ is the image of the i-th number α_h in the Hensel code set $\mathbf{H}(p,\overline{r})$;

S.2 n operations are carried out in $\mathbf{H}(p,\overline{r})$, in order to obtain the Hensel code result:

$$(mant_\gamma, exp_\gamma) = (mant_\alpha, exp_\alpha) \quad op_l' \quad (mant_\beta, exp_\beta),$$

where

$$(mant_\alpha, exp_\alpha), (mant_\beta, exp_\beta), (mant_\gamma, exp_\gamma) \in C_P^{\mathbf{H}(p,\overline{r})},$$

$$op_l' \in OP_P^{\mathbf{H}(p,\overline{r})}, \quad l = 1\ldots n.$$

S.3 The unique rational result is recovered by applying the EEA to the Hensel code result.

Figure 2: Algorithm for sequential p-adic computations.

integer numbers, whose complexity in terms of time is given as a function of the size z of the input numbers: $O_B(z^{log_2 3})$.

THEOREM 3.1. (COMPUTATIONAL COMPLEXITY OF THE SEQUENTIAL ALGORITHM) *The total time required to solve P using the Sequential Algorithm, under the assumptions (i),...,(v), is:*

$$O_B(max(m\ (\overline{r}\ log\ p)^{log_2 3}\ log\ (\overline{r}\ log\ p),\ n\ \overline{r}^2\ log\ p)). \tag{3.3}$$

Moreover if $n = m^u$, with $u \in \mathbf{N}$, the Sequential Algorithm takes

$$\begin{cases} O_B(m(\overline{r}\ log\ p)^{log_2 3}\ log(\overline{r}\ log\ p)) & if\ u = 1 \\ O_B(m^u(\overline{r}^2\ log\ p)) & if\ u > 1 \end{cases} \tag{3.4}$$

PROOF. Considering each of the steps of the Sequential Algorithm, we compute the time necessary to solve P.

In the step **S.1** the application of the forward mapping to m rational numbers of size $s = \overline{r} log_2 p$ (by assumption (iv)), requires (by THEOREM 1.2)

$$O_B(m\ (\overline{r}\ log\ p)^{log_2\ 3}\ log(\overline{r}\ log\ p)) \tag{3.5}$$

time. In fact in this case we apply the EEA to the denominator of the input number and to $p^{\overline{r}}$ whose size is always $\overline{r}\ log_2 p$ because it does not depend on the size of the input.

The computations of n multiplications in step **S.2** (by assumption (v)) in $\mathbf{H}(p, \overline{r})$, require (by THEOREM 2.5)

$$O_B(n\ \overline{r}^2\ log\ p) \tag{3.6}$$

time and the size of the resulting code is the same.

In the step **S.3** the application of the backward mapping algorithm requires (by THEOREM 2.4 and by assumption (i))

$$O_B((\overline{r}\ log\ p)^{log_2 3}\ log(\overline{r}\ log\ p)) \tag{3.7}$$

time.

The algorithm used to solve steps **S.1** and **S.3** is the same, with the difference that during the step **S.1** it is applied m times. So (3.7) does not influence the asymptotical behaviour of the Sequential Algorithm. From the estimates (3.5) and (3.6) relation (3.3) follows and hence (3.4). □

Let us note that the THEOREM 3.1 holds both under the assumption (iv.1) and under the assumption (iv.2).

4. Parallelization of p-adic Arithmetic

In order to develop a parallelization strategy we will refer to the model of shared memory MIMD machine. In practice, only a moderate number of processors (about 20 physical processors) can be realized.

Looking at formulae (3.3) and (3.4), we observe that the cost in computing time depends essentially on $\overline{r}^2$, hence a large $\overline{r}$ comes to be time wasteful. The only way to decrease the approximation $\overline{r}$ should be by increasing the base p. So the design of more efficient algorithms for big rational numbers arithmetic, has to deal with balancing the trade-off between a large base p and a reasonable approximation $\overline{r}$. Here the idea is to reduce the approximation $\overline{r}$ to an approximation $r < \overline{r}$, without increasing the base p (which, in fact, we have limited by the above relation (3.1)).

In order to exploit the possibilities of parallelization over k processors, we choose several prime numbers p_i, $i = 1, \ldots, k$, such that $g = p_1 \cdot p_2 \cdot \ldots \cdot p_k < p^k$. On this ground we state the further assumption:

(vi) $size(g) \leq size(p^k) = k\ log_2 p$, where $p = max(p_1, p_2, \ldots, p_k)$.

In such a way we carry out the computations in parallel, over the k distinct Hensel code domains $\mathbf{H}(p_1, r), \ldots, \mathbf{H}(p_k, r)$. So the approximation r needed for computing the result of P in $\mathbf{F}_{p,\overline{r}}$ can be obtained by the definition of the Farey fraction set from the relation:

$$(p^k)^r = p^{\overline{r}},$$

which implies

$$r = \frac{\overline{r}}{k}. \tag{4.1}$$

Actually no p_i (base of the computation on the i-th processor) exceeds the limit we

imposed above, while the effectively used approximation r is sensibly reduced, making the computations faster.

In order to analyze and compare the following algorithms with the Sequential Algorithm we can assume w.l.o.g.:

(vii) $r = log_2 p$. This is a reasonable assumption in order to balance the size of p and r in the definition of the Farey fraction set including the result of P. In fact, if $r > log_2 p$ then, from THEOREM 2.5 and from assumption (v), we note that such a larger r would be time wasteful. On the other hand we have to choose a large base p, so as to guarantee the result of the problem P in $\mathbf{F}_{p,r}$.

Given such a trade off we can also establish a convenient value for the number k of the processors to be used. Let us note that if $k < r$ then the advantages of parallelization would decrease. Moreover, we should actually limit the size of k by r because $k > r$ would not increase speed. Hence, we state the following assumption:

(viii) $k = r$.

Moreover, we rewrite the estimate (3.4) as follows:

$$\begin{cases} O_B(m(r^{3\ log_2 3}\ log\ r)) & \text{if } u = 1 \\ O_B(m^u(r^5)) & \text{if } u > 1 \end{cases} \tag{4.2}$$

On the ground of the previous assumptions ((i) ... (viii)) the parallel schema of computation is described in the following Figure 3.

THEOREM 4.1. (COMPUTATIONAL COMPLEXITY OF THE PARALLEL ALGORITHM) *The Parallel Algorithm takes*

$$O_B(max(m\ r^{3\ log_2 3}\ log\ r,\ \ n\ r^3,\ \ r^{2(1+log_2 3)})) \tag{4.3}$$

time, under the assumption (iv.1), and takes

$$O_B(max(m\ r^{2\ log_2 3}\ log\ r,\ \ n\ r^3,\ \ r^{2(1+log_2 3)})) \tag{4.4}$$

time, under the assumption (iv.2). Moreover, if $n = m^u$, with $u \in \mathbf{N}$, the following relations hold:

$$\begin{cases} O_B(m\ r^{2\ log_2 3}\ log\ r) & \text{if } u \leq 2 \\ O_B(m^u\ r^3) & \text{if } u > 2 \end{cases} \tag{4.5}$$

$$\begin{cases} O_B(\ r^{2(1+log_2 3)}) & \text{if } u = 1 \\ O_B(m^u\ r^3) & \text{if } u > 1 \end{cases} \tag{4.6}$$

PROOF. In the step **P.1**, following the assumptions (iv.1), (iv.2) and (vi), the application of the forward mapping to the m input rational numbers, requires

$$O_B(m\ (r^2\ log\ p)^{log_2 3}\ log(r^2\ log\ p)) = O_B(m\ r^{3\ log_2 3}\ log\ r) \tag{4.7}$$

time, if we consider the assumption (iv.1), and takes

$$O_B(m\ (r\ log\ p)^{log_2 3}\ log(r\ log\ p)) = O_B(m\ r^{log_2 3}\ log\ r) \tag{4.8}$$

PARALLEL ALGORITHM

Input: p_i, $i = 1, \ldots, k$, number of processors;
r, code length ($r = \bar{r}/k$);
$I_P = \{\alpha_h = a_h/b_h,\ h = 1, \ldots, m\}$;
Output: the rational solution $a/b \in \mathbf{F}_{p_1 \cdots p_k, r}$ $(= \mathbf{F}_{p,\bar{r}})$;

P.1 Each rational number $\alpha_h \in I_P$ is mapped into k Hensel codes, each belonging respectively to $\mathbf{H}(p_1, r), \ldots, \mathbf{H}(p_k, r)$. Let $(mant_{\alpha_{hi}}, exp_{\alpha_{hi}})$ be the image of the h-th number α_h in the i-th Hensel code set.

P.2 The computations are carried out in parallel by the k processors, the i-th processor performs the n operations needed for the solution of P, over $\mathbf{H}(p_i, r)$, where $i = 1, \ldots, k$:

$$(mant_{\gamma_i}, exp_{\gamma_i}) = (mant_{\alpha_i}, exp_{\alpha_i})\ op'_l\ (mant_{\beta_i}, exp_{\beta_i}),$$

where

$$(mant_{\alpha_i}, exp_{\alpha_i}),\ (mant_{\beta_i}, exp_{\beta_i}),\ (mant_{\gamma_i}, exp_{\gamma_i}) \in C_P^{\mathbf{H}(p_i,r)}$$

and

$$op'_l \in OP_P^{\mathbf{H}(p_i,r)},\quad l = 1, \ldots, n.$$

Here $OP_P^{\mathbf{H}(p_i,r)}$ denotes the set of image operators in $\mathbf{H}(p_i, r)$ of the operators in OP_P, and $C_P^{\mathbf{H}(p_i,r)}$ are the image sets of C_P, while $i = 1, \ldots, k$.

P.3.1 From the final k results, obtained in the previous step, the unique p-adic result is reconstructed by using the Chinese Remainder Algorithm (CRA);

P.3.2 The EEA is applied to the previous result (like step **S.3**).

Figure 3: Parallel Algorithm for p-adic computations.

time, if we consider the assumption (iv.2).
In the step **P.2**, by the assumptions (iii), (v), (vi), (vii), and by THEOREM 2.5,

$$O_B(n\ r^2\ log\ p) = O_B(n\ r^3) \tag{4.9}$$

time is required.
In order to perform step **P.3.1**, a specific backward mapping algorithm has been devised in Colagrossi A., Limongelli C. (1988). This backward mapping algorithm starts from the k given Hensel codes

$$(mant_{\gamma_i}, exp_{\gamma_i}) = (\gamma_{0,i} \ldots \gamma_{j,i} \ldots \gamma_{r-1,i}, exp_{\gamma_i})$$

where $\gamma_{j,i} \in \mathbf{Z}_{p_i}$ represents the $j-th$ digits of the mantissa of the $i-th$ Hensel code. It is based on a "Hensel like" lifting approach and essentially it uses the first j-1 ($j = 1 \ldots r$) digits of each Hensel code mantissa $mant_{\gamma_i}$ to obtain the j-th digit δ_j of the g-adic

expansion of the final code $d \in \mathbf{Z}_g$ (also called the g-adic result), where $g = p_1 \cdot p_2 \cdot, \ldots, \cdot p_k$:

$$d = \delta_0 + \delta_1 g + \cdots + \delta_{r-1} g^{r-1}. \tag{4.10}$$

Up to lower factors, the result of the computational analysis of this step is:

$$O_B(r^2\ (k\ log\ p)^{log_2 3}) = O_B(r^{2(1+log_2 3)}). \tag{4.11}$$

In step **P.3.2** the application of the backward mapping requires in any case (by THEOREM 2.4 and by assumption (vi))

$$O_B((r^2\ log\ p)^{log_2 3}\ log(r^2\ log\ p)) = O_B(r^{3\ log_2 3}\ log\ r^3) \tag{4.12}$$

time.

Now the theorem follows by (4.7), (4.8), (4.9), (4.11) and (4.12). □

If we analyze the detailed algorithm described in Colagrossi A., Limongelli C. (1988), we note that the most time consuming operations are the *CRA* and the computations of the g^j, for $j = 2, \ldots, r-1$. We also note that these steps are performed by the use of only one processor. In the next section we describe how to parallelize these operations (*CRA* and multiplications).

5. An improved parallel algorithm to recover the unique Hensel code result

In this section we define an improvement of the Parallel Algorithm, obtained by modifying the step **P.3.1** by means of further parallelizations.

We will call **(P.3.1)'** such an improved step and Improved Parallel Algorithm the algorithm resulting by substituting in the Parallel Algorithm the step **(P.3.1)'** in place of the step **P.3.1**.

In order to present the improved step **(P.3.1)'**, let us explain each further parallelization.

5.1. *CRA* PARALLELIZATION

The parallelization of *CRA* can be carried out on the basis of the following formula given in Krishnamurthy E. V. (1985):

$$\delta_j = \sum_{i=1}^{k} g_i \cdot \gamma_{j,i} \cdot \overline{g}_i \ \ mod\ g \tag{5.1}$$

where $g_i = g/p_i$, and $\overline{g}_i$ is the solution of

$$(\frac{g}{p_i}) \cdot \overline{g}_i \equiv 1 \ \ mod\ p_i.$$

Let us note that the p_i are relatively prime, so $\overline{g}_i$ can be uniquely determined as $g_i^{-1}\ mod\ p_i$, by applying the *EEA*. Since the terms of the summation (5.1) are independent of each other, the computation of δ_j is parallelized in time

$$O_B(max((k\ log\ p)^{log_2 3},\ \ k^2\ log\ p) = O_B((k\ log\ p)^{log_2 3}), \tag{5.2}$$

that is the time (by assumptions (vii) and (viii)) necessary to compute the terms $g_i\ \gamma_{j,i}$, $\overline{g}_i$ in parallel, and to add them together.

5.2. Parallelization of multiplications

Referring to the step **P.3** of the Parallel Algorithm described in the previous section, we have to note that the g-adic result d (see relation (4.10)) is computed through intermediate computations of terms $d_l = \sum_{j=0}^{l-1} \delta_j \; g^j$ and g^{j+1} which are performed through one processor. In order to improve the complexity of this algorithm, we have to parallelize these steps. We must parallelize the following multiplications:

$$g^{j-1} \cdot \delta_{j-1}, \text{ and } g^{j-1} \cdot g.$$

In both of these multiplications the first and the second operand has size respectively $((j-1)\ k\ log_2 p)$ and $(k\ log_2 p)$. We chop the first operand in $j-1$ blocks $A_1, A_2, \ldots, A_{j-1}$ of size $(k\ log_2 p)$. Then we compute $j-1$ parallel multiplications between each of these numbers and the second operand Let us note that now the $j-1$ multiplications are computed in parallel between numbers of the same size; after that we suitably draw up these numbers and carry out the additions in order to recover the result. The sequential Karatsuba algorithm used to multiply these two numbers requires in this case

$$O_B((j\ k\ log\ p)^{log_2 3})$$

time, while the parallelized version of the classical algorithm essentially requires the time to multiply two numbers of size $(k\ log_2 p)$ and, using the parallel addition, the time required to computing j additions, so the parallel algorithm for multiplications, requires

$$O_B(max((k\ log\ p)^{log_2 3}, k\ j\ log\ p)) = O_B((k\ log\ p)^{log_2 3}). \tag{5.3}$$

5.3. Improvement of the Step P.3.1 of the Parallel Algorithm

In Figure 4 we present the details of the step **(P.3.1)'**:

Let us note that O_{14} is the g-adic solution of the problem P and it will be the input of the step **P.3.2** of the Parallel Algorithm.

Moreover let us note that the steps St-4, St-6, St-12 and St-14 do not affect the whole parallel complexity, since they involve only additions (see Lemma 5.1).

The cost of the step **(P.3.1)'** is estimated by the following lemma.

Lemma 5.1. (Computational complexity of step **(P.3.1)'**) *Step* **(P.3.1)'** *takes*

$$O_B(r^{1+2log_2 3}) \tag{5.4}$$

Proof. **St-0** doesn't influence the computational analysis, because it is an initialization step.

Steps **St-1**, **St-2** and **St-7** consist of multiplications and/or divisions between elements of size $(k\ log_2 p)$ and elements of size $(log_2 p)$. Their computation through Karatsuba algorithm requires $O_B((k\ log\ p)^{log_2 3})$ time.

Steps **St-8** and **St-9** require $O_B((log\ p)^{log_2 3})$ time to perform multiplications between elements of size $(log_2 p)$.

Steps **St-3**, St-4 and **St-11**, St-12, perform the CRA computation as described in subsection 5.1. Their computational complexity is represented by (5.2);

IMPROVED STEP (P.3.1)'

k *processors*, $i = 1 \ldots k$

Input: $p_1, p_2, \ldots, p_k$: k modules;;
$g = p_1 \cdot p_2 \cdot \ldots \cdot p_k$;
r: code length;
$(mant_{\gamma_i}, exp_{\gamma_i}) = (\gamma_{0,i} \ldots \gamma_{j,i} \ldots \gamma_{r-1,i}, exp_{\gamma_i})$, $i = 1, \ldots, k$;
Output: $d = \sum_{j=0}^{r-1} \delta_j \cdot g^j$.

St-0	$d_{-1} = 0$; $\gamma_{0,i} = s_{0,i}$;	
St-1	I_1 : g, p_i;	O_1 : $g_i = g/p_i$;
St-2	I_2 : g_i, p_i;	O_2 : $\overline{g}_i = \lvert g_i^{-1} \rvert_{p_i}$;
St-3	I_3 : $\overline{g}_i$, $p_i, \gamma_{0,i}$;	O_3 : $g_i \cdot \gamma_{0,i} \cdot \overline{g}_i$;
St-4	I_4 : $g_i \cdot \gamma_{0,i} \cdot \overline{g}_i$, for $i = 1 \ldots k$;	$O_4 : \delta_0 = \sum_{i=1}^{k} O_{3,i}$;

for $j = 2$ **to** $r - 1$
do

St-5	I_5 : δ_{j-1}, g^{j-1};	O_5 : $\delta_{j-1} \cdot g^{j-1}$;
St-6	$I_6 : O_5, d_{j-2}$;	$O_6 : d_{j-2} + \delta_{j-1} \cdot g^{j-1} = d_{j-1}$;
St-7	$I_7 : \overline{s}_{j,i-1}, \delta_{j-1}, g_i^{j-1}$;	$O_7 : \overline{s}_{j,i} = \lvert \frac{(\overline{s}_{j,i-1} - \delta_{j-1} g_i^{j-1})}{p_i} + \gamma_{j,i} \rvert_{p_i}$;
St-8	$I_8 : \overline{s}_{j,i}, \overline{g}_i^j$ (O_3 *if* $j = 1$, O_{10} *if* $j > 1$);	$O_8 : \overline{\gamma}_{j,i} = \lvert \overline{s}_{j,i} \cdot \overline{g}_i^j \rvert_{p_i}$;
St-9	$I_9 : \overline{g}_i^j$, (O_3 *if* $j = 1$, O_{10} *if* $j > 1$), $\overline{g}_i$ (O_3);	$O_9 : \lvert \overline{g}_i^j \cdot \overline{g}_i \rvert_{p_i} = \overline{g}_i^{j+1}$;
St-10	$I_{10} : g, g^{j-1}$;	$O_{10} : g^j = g \cdot g^{j-1}$;
St-11	$I_{11} : O_1, O_3, O_8$;	$O_{11} : g_i \cdot \overline{\gamma}_{j,i} \cdot \overline{g}_i$;
St-12	$I_{12} : O_{11}$;	$O_{12} : \delta_{j-1} = \sum_{i=1}^{k} O_{11,i}$;

od

St-13	$I_{13} : O_{10}, O_{12}$;	$O_{13} : \delta_{r-1} \cdot g^{r-1}$;
St-14	$I_{14} : O_{13}, O_6$;	$O_{14} : d = d_{r-2} + \delta_{r-1} \cdot g^{r-1} = \sum_{j=0}^{r-1} \delta_j \cdot g^j$.

Figure 4: Improved Step **(P.3.1)'**.The steps parallelized over k processors are outlined in **boldface**, while the sequential ones are in plain style.

Steps **St-5**, **St-10** and **St-13** perform the parallel multiplications as described in subsection 5.2. Their computational complexity is given in (5.3).
Steps St-6 and St-14 are not parallelized. They require $O_B(j\ k\ log\ p)$ time (addition between elements of size $(j\ k\ log_2 p)$).
The main loop is iterated $r-1$ times; hence the analysis of its heaviest steps (**St-11** and St-12) allows to represent the computational trend of the algorithm as follows:

$$O_B(max(\sum_{j=2}^{r-1} j\ k\ log\ p,\ \sum_{j=2}^{r-1}(k\ log\ p)^{log_2 3})) =$$

$$O_B(max(r^2\ k\ log\ p,\ r\ (k\ log\ p)^{log_2 3})) =$$

$$O_B(r\ (k\ log\ p)^{log_2 3}))$$

which, by assumptions (vii) and (viii), gives exactly (5.4). □

THEOREM 5.1. (COMPUTATIONAL COMPLEXITY OF THE IMPROVED PARALLEL ALGORITHM) *The Improved Parallel Algorithm takes*

$$O_B(max(m\ r^{3\ log_2 3}\ log\ r,\ n\ r^3,\ r^{1+2\ log_2 3})) \tag{5.5}$$

time, under the assumption (iv.1), and takes

$$O_B(max(m\ r^{2\ log_2 3}\ log\ r,\ n\ r^3,\ r^{1+2\ log_2 3})) \tag{5.6}$$

time, under the assumption (iv.2). Moreover if $n = m^u$, *with* $u \in \mathbf{N}$, *the following relations hold:*

$$\begin{cases} O_B(m\ r^{3\ log_2 3}\ log\ r) & if\ u \leq 2 \\ O_B(m^u\ r^3) & if\ u > 2 \end{cases} \tag{5.7}$$

$$\begin{cases} O_B(r^{1+2\ log_2 3}) & if\ u = 1 \\ O_B(m^u\ (r^3) & if\ u > 1 \end{cases} \tag{5.8}$$

PROOF. Taking into consideration that in the Improved Parallel Algorithm the step **(P.3.1)'** is performed in place of the **P.3.1** of the Parallel Algorithm, the estimate (4.12) given in THEOREM 4.1 (case of the assumption (iv.1)), becomes

$$O_B(max(m(r^2\ log\ p)^{log_2 3}\ log(r^2\ log\ p),\ n\ r^2\ log\ p,\ r(k\ log\ p)^{log_2 3})) \tag{5.9}$$

where (5.4) has been inserted in place of the third operand of (4.3). The estimate (4.4) given in THEOREM 5.1 (case of the assumption (iv.2)) becomes

$$O_B(max(m(r\ log\ p)^{log_2 3}\ log(r\ log\ p),\ n\ r^2\ log\ p,\ r(k\ log\ p)^{log_2 3})) \tag{5.10}$$

where (5.4) has been inserted in place of the third operand of (4.4).
Following the assumptions (vii)and (viii), (5.5) and (5.6) follow by (5.9) and (5.10) respectively. □

6. Comparisons

The following theorem allows to appreciate how better efficiency is reached by the Improved Parallel Algorithm w.r.t. the former Parallel Algorithm. Moreover it supports a final evaluation of the general parallel approach w.r.t. the sequential one.

THEOREM 6.1. (COMPARISONS) *The Improved Parallel Algorithm is less time consuming, or at most equal time consuming as the Sequential and Parallel Algorithm.*

PROOF. The following tables can be derived by THEOREM 6., THEOREM 7. and THEOREM 8.:

	Seq. Alg.	Par. Alg.	Imp. Par. Alg.
case $u = 1$			
assumption (iv.1)	$O_B(m(r^{3\ log_2 3}\ log\ r))$	$O_B(m(r^{3\ log_2 3}\ log\ r))$	$O_B(m(r^{3\ log_2 3}\ log\ r))$
assumption (iv.2)	$O_B(m(r^{3\ log_2 3}\ log\ r))$	$O_B((r^{2(1+log_2 3)})$	$O_B(m(r^{2\ log_2 3}\ log\ r))$
case $u = 2$			
assumption (iv.1)	$O_B(m^u\ r^5)$	$O_B(m(r^{3\ log_2 3}\ log\ r))$	$O_B(m(r^{3\ log_2 3}\ log\ r))$
assumption (iv.2)	$O_B(m^u\ r^5)$	$O_B((r^{2(1+log_2 3)})$	$O_B(m^u\ r^3)$
case $u > 2$			
assumption (iv.1)	$O_B(m^u\ r^5)$	$O_B(m^u\ r^3)$	$O_B(m^u\ r^3))$
assumption (iv.2)	$O_B(m^u\ r^5)$	$O_B(m^u\ r^3)$	$O_B(m^u\ r^3))$

Finally, in the following table the related inequalities are summarized (with 1.59 used in place of $log_2 3$):

Seq. Alg.	Par. Alg.	Imp. Par. Alg.	u
$O_B(m(r^{4.77}\ log\ r))$	$= O_B(m(r^{4.77}\ lo\ r))$	$= O_B(m(r^{4.77}\ log\ r))$	$u = 1$
$O_B(m^u\ r^5)$	$> O_B(m(r^{4.77}\ log\ r))$	$= O_B(m(r^{4.77}\ log\ r))$	$u = 2$
$O_B(m^u\ r^5)$	$> O_B(m^u\ r^3)$	$= O_B(m^u\ r^3)$	$u > 2$

under the assumption (iv.1),

and

$O_B(m(r^{4.77}\ log\ r))$	$> O_B(r^{5.18})$	$> O_B(m\ r^{3.18}\ log\ r)$	$u = 1$
$O_B(m^u\ r^5)$	$> O_B(r^{5.18})$	$> O_B(m^u\ r^3)$	$u = 2$
$O_B(m^u\ r^5)$	$> O_B(m^u\ r^3)$	$= O_B(m^u\ r^3)$	$u > 2$

under the assumption (iv.2).

□

Let us outline the benefit of the proposed Improved Parallel Algorithm in the frame of the parallel approach itself. If we compare the orders (5.4) and (4.11) and look at the inequalities stated above in THEOREM 9., we note that the further parallelizations carried out in step **(P.3.1)'**, have improved the computational complexity of the step **P.3.1** (when $u \leq 2$ under assumption (iv.2)).

As we have previewed in the introduction, the new algorithm provides a better behaviour when it is exploited for the solutions of problems in which $u \leq 2$. Actually there

exists a wide class of such problems. Let us consider, for example, the problems related to the computation over matrices. We know that given two N-dimensional square matrices ($2 \cdot N^2$ input data), the product is computed by N^3 multiplications. So the problem belongs to this class, since the number of operations is at most $4 \cdot N^4$, i.e. $n = N^3 < 4 \cdot N^4 = (2 \cdot N^2)^2 = m^2$. The computation of determinant, the matrix inversion and the LUP decomposition are also examples of problems which belong to this class: these problems are reducible to matrix multiplication as Aho, Hopcroft, Ullman (1975) shows. Other problems in this class are the polynomial multiplication, division and reciprocal, see Aho, Hopcroft, Ullman (1975). Let us consider, for example, the case of polynomial multiplication. Let $\sum_{i=0}^{d_a} a_i \cdot x^i$ and $\sum_{i=0}^{d_a} b_i \cdot x^i$ be two polynomials having the same degree d_a. The number of operations involved in their multiplication is at most $n = 3(d_a + 1)^2 < 4(d_a + 1)^2 = (2(d_a + 1))^2 = m^2$. Another example is the evaluation of a given univariate polynomial. The inputs for the computation of $\sum_{i=0}^{d_a} a_i \cdot x^i$ are the d_a+1 coefficients and the value $\overline{x}$. If the evaluation is computed by following the Horner's rule as it is shown in Aho, Hopcroft, Ullman (1975), we need d_a multiplications and d_a additions i.e.: $n = 2d_a < (d_a + 2)^2 = m^2$. We can also cite the wide class of statistical applications in which, in most cases, few operations are performed over a very large set of input data.

But the main result of such an improvement is that the computational complexity of the Improved Parallel Algorithm is never dominated by the cost of the g-adic result recovering step, while in the Parallel Algorithm it was. In the Parallel Algorithm, under assumption (iv.2), the computational complexity is actually the order due to step **P.3.1.** In the Improved Parallel Algorithm, the overall complexity is no more due to step **(P.3.1)'**.

If we want to compare the performance of the parallel algorithm w.r.t. the sequential one, the table above supports us showing that, under both the assumptions (iv.1) and (iv.2), when $u > 2$, the sequential algorithm and the parallel algorithms have respectively the complexity

$$O_B(m^u\ r^5) \quad \text{and} \quad O_B(m^u\ r^3),$$

where m^u is the number of operations necessary to solve the given expression, and r is the length of the Hensel codes. If we recall the assumptions (vii) and (viii) (i.e., $k = r = log_2 p$) we note that the complexity of the parallel algorithm decreases by a factor $k^2 = r^2$. In fact we have obtained, in (4.1), that the code length r in the Parallel Algorithm is k times smaller than the code length $\overline{r}$ in the Sequential Algorithm.

References

Aho, Hopcroft, Ullman (1975). *The Design and Analysis of Computer Algorithms*, Addison Wesley–Publishing Company.

Bachman, G. (1964). *Introduction to p-adic Numbers and Valutation Theory*, Academic Press, New York.

Buchberger B. (1970). An Algorithm for Finding the Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal (German).*Ph.D. Thesis*, Math. Inst. Univ. of Innsbruck, Austria 1965, and *Aequationes Math. 4/3, 374-383*, 1970.

Colagrossi A., Limongelli C. (1988). Big Numbers p-adic Arithmetic: a Parallel Approach, *Proc. AAECC-6*, Springer Verlag, LNCS Series n. 357.

Colagrossi A., Miola A. (1987). A Normalization Algorithm for Truncated p-adic Arithmetic, *Proc. IEEE 8-th Symposium on Computation Arithmetic*, Como, May 1987.

Colagrossi A., Limongelli C., Miola A. (1990). Scientific Computation by Error-free Arithmetics, *Proc. ICC'90, Computer Trends and Applications, Indian Computing Congress Series*, pagg.63-72, E. Balagurusami, B. Sushila, eds., TaTa-McGraw-Hill.

Collins G. E. (1975). Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition, *Proc. Second Conference on Automata Theory and Formal Languages*, Springer Verlag, LNCS Series *n.* 33.

Dittenberger K. (1985). An Efficient Method for Exact Numerical Computation. *Diploma Thesis*, University of Linz, RISC-Linz Institute, Austria.

Gregory R.T., Krishnamurthy E.V. (1984). *Methods and Applications of Error-Free Computation*, Springer Verlag.

Hensel, K. (1908). *Theorie der Algebraischen Zehlen*, Teubner, Leipzig-Stuttgart.

Knuth D. (1981). *The Art of Computer Programming, Vol II, Seminumerical Algorithms*, Addison Wesley Publishing Company.

Koblitz N. (1977). *p-adic Numbers, p-adic Analysis and Zeta Functions*, Springer-Verlag, 1977.

Krishnamurthy E. V. (1985). *Error-Free Polynomial Matrix Computations*, Springer-Verlag.

Limongelli C. (1987). Aritmetiche non-standard: implementazione e confronti, *Diploma Thesis*, University of Rome "La Sapienza".

Loos R. (1983). Generalized Polynomial Remainder Sequences, in *Computer Algebra - Symbolic and Algebraic Computation*, Buchberger B., Collins E.G., Loos R., eds., Springer Verlag.

Mahler K. (1973). *Introduction to p-adic Numbers and their Functions*, Cambridge University Press.

Miola, A. (1984). Algebraic Approach to pp-adic Conversion of Rational Numbers, *Information Processing Letters n. 18, pagg. 167-171* , Elsevier Science Publisher B.V. (North-Holland).

Wang D. (1991). Characteristics Set and Zero Structure of Polynomial Sets, *Lecture Notes, RISC-Linz*, Johannes Kepler University, Austria.