Exact solution of computational problems via parallel truncated p-adic arithmetic *

C. Limongelli

1 Introduction

The aim of the paper is to show the effectiveness of the p-adic arithmetic in scientific computation by selecting and solving problems which manipulates "big" numbers.

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.

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, 1970), Cylindrical Algebraic Decomposition (Collins, 1975), Characteristic Sets (Wang, 1991), Polynomial Remainder Sequences (Buchberger & Loos, 1983), Exponentiation and Gaussian Elimination (Knuth, 1981).

Classical algorithms for rational arithmetic operations (i.e.: arbitrary precision rational arithmetic) described in (Knuth, 1981), are not suitable for big numbers manipulations, due to the cost of GCD algorithm that has to be often applied during computations. Classical modular arithmetic (Knuth, 1981) has also one essential lack which depend on the restriction of the set of the numbers on which it works: the integers. When we reformulate a problem stated over the rationals by means of integers, the size of the numbers blows up, producing very slow computations.

We show how it is possible to handle with rational numbers in an efficient way, starting by p-adics and parallelism. We will also see how has been possible to enhance the speed-up of computations for some classes of mathematical problems, by parallel p-adic approach.

*This work has been partially supported by Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo" of CNR under grant n. 92.01604.69.

Next section presents some definitions necessary to read the paper. The model of p-adic computation will be presented in Section 3 in order to introduce the parallel p-adic schema which is described in Section 4. Section 5 focuses the attention on the parallelization of the Chinese Remainder Algorithm (CRA) which is the most expensive step in this model. Two main parallelization of this algorithm have been proposed and we describe both of them, from a theoretical and a practical point of view. Some experiments are shown in Section 6 and the last Section 7 discuss about the applicability of this approach and the further steps that have to be done to reach a complete parallelization.

2 Working tools

We skip here basic notions about p-adic arithmetic which can be found in (Koblitz, 1977; Gregory & Krishnamurthy, 1984) and (Colagrossi, Limongelli, & Miola, 1995). We just give two main definitions that are necessary for reading the following sections.

DEFINITION 1 (HENSEL CODES) *Given a prime number p, a Hensel code of length r of any 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 left-most r digits and the value e of the related p-adic expansion are called the mantissa and the exponent, respectively. Moreover

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

□

Let $\mathbb{H}_{p,r}$ indicate the set of Hensel codes 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.

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. The result of the computations must belong to the Farey fraction' set which is defined as follows.

DEFINITION 2 (FAREY FRACTION SET) *The Farey fraction set $\mathbb{F}_{p,r}$ is the subset of rational numbers a/b such that:*

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

□

Theorem 1 (Computational complexity of the Forward Mapping) *The forward mapping algorithm given in (Colagrossi et al., 1995) (*Theorem *1), applied to two n-bit numbers, requires $O_B(M(n)\ log_2 n)$ time, where $M(n)$ is the time required to multiply two n-bit numbers.*

Proof: See (Aho, Hopcroft, & Ullman, 1975). □

Theorem 2 (Computational complexity of Backward Mapping) *The sequential backward mapping algorithm given in in (Colagrossi et al., 1995) (*Theorem *3), applied to two n-bit numbers, requires $O_B(M(n)\ log_2 n)$ time.*

Proof: See (Aho et al., 1975). □

Let us consider now the algebra on Hensel codes, and the related algorithms, as given in (Gregory & Krishnamurthy, 1984); see however the corrections in (Dittenberger, 1987). The following theorem holds:

Theorem 3 (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:*

+, - : $O_B(r\ log_2 p)$;

$\cdot$, / : $O_B(r^2\ log_2 p)$.

Proof: See (Limongelli, 1987). □

Moreover, in our computations we use the sequential Karatsuba algorithm to multiply integer numbers, whose complexity in terms of time is given as a function of the size z of the input numbers: $O_B(z^{log_2 3})$.

3 The model of p-adic computation

Before showing the main steps of the algorithm which solves a 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 $\mathbb{F}_{p,r} \subset \mathbb{Q}$;

(ii) let a be the size of $max(a_1, \ldots, a_m, b_1, \ldots, b_m)$, where $q_h = a_h/b_h$, $(h = 1, \ldots, m)$ 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 $I_P = \{q_1, \ldots, q_m\}$ is the set of rational numbers given as the input for solving P and let OP_P be the set of the operators on $\mathbb{Q}$ in assumption (iii): $OP_P = \{op_1, \ldots, op_n\}$. We can define the set C_{P_n} of all the values involved in the computation (including eventually the result of P) as follows:

$$\begin{cases} C_{P_0} = I_P; \\ C_{P_i} = C_{P_{i-1}} \cup q_h \; op_i \; q_{h'}, \quad q_h, q_{h'} \in C_{P_{i-1}}, \;\; h, h' \in \mathbb{N} \quad i \leq n; \end{cases} \tag{1}$$

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 et al., 1995).

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 word-size w of our computer:

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

consequently we determine the approximation $\overline{r}$ as follows (see 2):

$$\overline{r} = \lceil log_2 p(2 \; N^2 + 1) \rceil. \tag{3}$$

The algorithm is described in Figure 1.

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) $s = \overline{r} \; log_2 p$ (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 $O_B(\overline{r}^2 \; log_2 p)$, according to the THEOREM 2.5.

Sequential Algorithm

Input: p: prime number,
$\overline{r}$: code length, found by relation (3),
$I_P = \{q_h,\ h = 1, \ldots, m\}$: m rational input numbers;

Output: the rational solution $q \in \mathbb{F}_{p,\overline{r}}$;

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

1.2 Following the model of computation given in (1), n operations are carried out in $\mathbb{H}_{p,\overline{r}}$, in order to obtain the Hensel code result:

$$(mant_q, exp_q)$$

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

Figure 1: Algorithm for sequential p-adic computations.

THEOREM 4 (COMPUTATIONAL COMPLEXITY OF THE SEQUENTIAL ALGORITHM) *The total time required to solve P using the Sequential Algorithm, under the assumptions (i),...,(v), is:*

$$O_B(max(m\ (\overline{r}\ log_2 p)^{log_2 3}\ log_2(\overline{r}\ log_2 p),\ n\ \overline{r}^2\ log_2 p)). \tag{4}$$

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

$$\begin{cases} O_B(m(\overline{r}\ log_2 p)^{log_2 3}\ log(\overline{r}\ log_2 p)) & if\ u = 1 \\ O_B(m^u(\overline{r}^2\ log_2 p)) & if\ u > 1 \end{cases} \tag{5}$$

PROOF: See (Limongelli, 1993). □

However, it is worth to have a look at THEOREM 6 enunciated in (Colagrossi et al., 1995) which states that the inverse mapping can be performed only when the expected result belongs to $\mathbb{F}_{p,r}$.

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

Once a given algorithm is stated for the solution of the problem, the related estimate of the solution can be evaluated. This can come out to be a not simple 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 is well known that the determinant $D(A)$ of a given n-dimensional square matrix A with integer coefficients is bounded by $n! \cdot a^n$, where $a = max\{\mid a_{i,j} \mid\}$, $1 \leq i,j \leq n$. It is also trivial to estimate the maximum coefficient which raises by polynomial multiplication: given the polynomials $\sum_{i=0}^{n} a_i \cdot x^i$ and $\sum_{j=0}^{m} b_j \cdot x^j$, if $a = max\{\mid a_i \mid\}_{1 \leq i \leq n}$, $b = max\{\mid b_j \mid\}_{1 \leq j \leq m}$ and $c = max\{a, b\}$, then the larger coefficient of the resulting polynomial is bounded by $max\{n, m\} \cdot c^2$. Also the problem of rational exponentiation is immediately estimate: if we have to compute a/b^c the size of the result will be given by $c \cdot log_2(max\{a, b\})$.

4 Parallelization of p-adic computations

In order to develop a parallelization strategy we will refer to a model of shared memory MIMD machine, with a moderate number of processors (about 20 physical processors).

Looking at formulae (4) and (5), 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 (2)).

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 $\mathbb{H}_{p_1,r}, \ldots, \mathbb{H}_{p_k,r}$. So the approximation r needed for computing the result of P in $\mathbb{F}_{p,\overline{r}}$ can be obtained by the definition of the Farey fraction set from the relation:

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

which implies

$$r = \frac{\overline{r}}{k}. \tag{6}$$

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.

We assume to have an input a problem P stated over m rational numbers $q_1, \ldots, q_m$. Our goal is to solve it, obtaining a rational result q. We proceed in three main steps (see also Figure 2):

- 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.

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

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

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

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

INPUT		$q_1, \ldots, q_m \in \mathbb{Q}$	
FORWARD MAPPING	$\swarrow$	$\downarrow$	$\searrow$
HENSEL CODES	$\bar{q}_1^{(1)}, .., \bar{q}_m^{(1)} \in \mathbb{H}_{p_1,r}$	$\ldots$	$\bar{q}_1^{(k)}, .., \bar{q}_m^{(k)} \in \mathbb{H}_{p_k,r}$
	$\downarrow$	$\ldots \downarrow \ldots$	$\downarrow$
OPERATIONS	$\downarrow$	$\ldots \downarrow \ldots$	$\downarrow$
	$\downarrow$	$\ldots \downarrow \ldots$	$\downarrow$
k SOLUTIONS	$\bar{q}^{(1)} \in \mathbb{H}_{p_1,r}$	$\ldots$	$\bar{q}^{(k)} \in \mathbb{H}_{p_k,r}$
CRA	$\searrow$	$\downarrow$	$\swarrow$
UNIQUE SOLUTION		$\bar{q} \in \mathbb{H}_{p_1 \cdot \ldots \cdot p_k,r}$	
BACKWARD MAPPING		$\downarrow$	
OUTPUT		$q \in \mathbb{F}_{p,r} \subset \mathbb{Q}$	

Figure 2: General schema of parallel p-adic computations.

In order to analyze and compare the parallel algorithms that we are going to define, with the Sequential Algorithm, we can assume w.l.o.g.:

(vii) $r = log_2 p$. This is a reasonable assumption since if $r > log_2 p$ then, from THEOREM 3 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 $P \in \mathbb{F}_{p,r}$.

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 extent of k to some bound since also a k too great could be unsuitable. Hence, also in order to make less cumbersome the notation in the evaluations of computational complexity, we state the following assumption:

(viii) $k = r$.

Moreover, we rewrite the estimate (5) as follows:

$$\begin{cases} O_B(m(r^{3\ log_2 3}\ log_2 r)) & \text{if } u = 1 \\ O_B(m^u(r^5)) & \text{if } u > 1 \end{cases} \tag{7}$$

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

Parallel Algorithm

Input: $p_i,\ i = 1, \ldots, k$, number of processors;
r, code length $(r = \overline{r}/k)$;
$I_P = \{q_h = a_h/b_h,\ h = 1, \ldots, m\}$;

Output: the rational solution $a/b \in \mathbb{F}_{p_1 \cdot \ldots \cdot p_k, r}$;

2.1 Each rational number $q_h \in I_P$ is mapped into k Hensel codes, each belonging respectively to $\mathbb{H}_{p_1,r}, \ldots, \mathbb{H}_{p_k,r}$. Let $(mant_{q_{hi}}, exp_{q_{hi}})$ be the image of the h-th number q_h in the i-th Hensel code set.

2.2 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 $\mathbb{H}_{p_i,r}$, according to the model given in (1), where $i = 1, \ldots, k$ computing the k solutions:

$$(mant_{\gamma_i}, exp_{\gamma_i}) \in C_{P_n}$$

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

2.3.2 The EEA is applied to the previous result (like step **1.3**).

Figure 3: Parallel algorithm for p-adic computations.

THEOREM 5 (COMPUTATIONAL COMPLEXITY OF THE PARALLEL ALGORITHM) *The Parallel Algorithm takes*

$$O_B(max(m\ r^{3\ log_2 3}\ log_2 r,\quad n\ r^3,\quad r^{2(1+log_2 3)})) \tag{8}$$

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

$$O_B(max(m\ r^{2\ log_2 3}\ log_2 r,\quad n\ r^3,\quad r^{2(1+log_2 3)})) \tag{9}$$

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

$$\begin{cases} O_B(m\ r^{2\ log_2 3}\ log_2 r) & \text{if } u \leq 2 \\ O_B(m^u\ r^3) & \text{if } u > 2 \end{cases} \tag{10}$$

$$\begin{cases} O_B(\ r^{2(1+log_2 3)}) & \text{if } u = 1 \\ O_B(m^u\ r^3) & \text{if } u > 1 \end{cases} \tag{11}$$

PROOF: See (Limongelli, 1993). □

If we analyze the detailed algorithm described in (Colagrossi & Limongelli, 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. Actually the crucial step of this approach is the detection of the unique solution via the CRA.

5 The problem of the recovery step

It is very hard to parallelize the sequential CRA as it is optimized for the sequential case by iterating over the input lists, where in each iteration only a quite cheap computation is necessary. For a more detailed discussion about problems that arise with this parallelized algorithm see (H.W., 1993).

We discuss now three main approaches which have been considered. The first CRA parallelization has been proposed in (Colagrossi & Limongelli, 1988). This algorithm starts from the k given Hensel codes

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

where $q_{j,i} \in \mathbb{Z}_{p_i}$ 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 $mant_{q_i}$ to obtain the j–th digit δ_j of the g-adic expansion of the final code $\overline{q} \in \mathbb{Z}_g$, where $g = p_1 \cdot p_2 \cdot, \ldots, \cdot p_k$ (also called the g-adic result):

$$\overline{q} = \delta_0 + \delta_1 g + \cdots + \delta_{r-1} g^{r-1}. \tag{12}$$

We call this algorithm $\mathcal{S}_{CRA}$ (Stepwise CRA). If we analyze this algorithm described in detail in (Colagrossi & Limongelli, 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.

Therefore, it would be better to base the implementation of the parallel CRA on an idea that contains more inherent parallelism. The following theorem, which is taken from (Krishnamurthy, 1985), is a good basis for a parallel implementation:

THEOREM 6 (CHINESE REMAINDER THEOREM) *Let $m_1, \ldots, m_k$ be k relatively prime integers > 1. Then for any $s_1, \ldots, s_k (s_i < m_i)$ there is a unique integer s satisfying*

$$s < \prod_{i=1}^{k} m_i =: M$$

and $s_i \equiv s \mod m_i$; the integer s can be computed using

$$s = \sum_{i=1}^{k} (\frac{M}{m_i}) s_i T_i \mod M, \tag{13}$$

where T_i is the solution of

$$(\frac{M}{m_i}) T_i \equiv 1 \mod m_i. \quad \square$$

In (Limongelli, 1993) an efficient algorithm is proposed to compute the recovery step which parallelize the sequential step of the algorithm described above making use of this theorem. In the following we will briefly describe the main steps of this algorithm called $\mathcal{SI}_{CRA}$ (Stepwise Improved CRA) and of a new algorithm that is called $\mathcal{P}_{CRA}$ (Pure CRA) that applies the CRA in a more natural way, applying directly THEOREM 6. Furthermore, we will compare the complexities of these two algorithms. Firstly let us write input and output specifications which are common to all three algorithms.

Let us note that we have arbitrarily many virtual processors available that will be automatically mapped on the real processors.

$\mathcal{SI}_{CRA}$ algorithm is based on $\mathcal{S}_C RA$ algorithm. It essentially applies THEOREM 6 at each step of the lifting process according to the steps shown in Fig. 5.

$\mathcal{P}_{CRA}$ algorithm shown in Fig. 6 applies directly THEOREM 6:

In (Colagrossi & Limongelli, 1988), an algorithm has been devised to perform the recovery step of the unique p-adic solution, with an asymptotic computational

CRA SPECIFICATION

INPUT:	$p_1, p_2, \ldots, p_k$: k prime modules;
	$g = p_1 \cdot p_2 \cdot \ldots \cdot p_k$;
	r: code length;
	$(mant_{\gamma_i}, exp_{\gamma_i}) = (\gamma_{0,i} \ldots \gamma_{j,i} \ldots \gamma_{r-1,i}, exp_{\gamma_i}),\ i = 1, \ldots, k$:
	k Hensel code results;
OUTPUT:	$\overline{q} = \sum_{j=0}^{r-1} \delta_j \cdot g^j$: the unique code result.

Figure 4: Common specification for CRA.

$\mathcal{SI}_{CRA}$: STEPWISE CRA ALGORITHM

$\mathcal{SI}_{CRA}.1$: Apply CRA to the Hensel codes' digit $\gamma_{0,i}$ and moduli p_i, for $i = 1, \ldots, k$, to obtain δ_0;

$\mathcal{SI}_{CRA}.2$: For all j from 1 to $r-1$ do the following:

$\mathcal{SI}_{CRA}.2.1$: Apply the lifting algorithm as described in (Colagrossi & Limongelli, 1988) to δ_{j-1}, $\gamma_{j,i}$ and moduli p_i to obtain the values $\overline{\gamma}_{j,i}$, for $i = 1, \ldots, k$;

$\mathcal{SI}_{CRA}.2.2$: Apply the CRA as it is given in Theorem 6, to the images $\overline{\gamma}_{j,i}$ and to the moduli p_i to obtain δ_j.
In this case we put $s = \delta_j$, $s_i = \gamma_{j,i}$, $m_i = p_i$, $M = g = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, so we have $(M/m_i) = g_i$ and $T_i = |{g_i}^{-1}|_{p_i}$. Hence the summation (13) becomes:

$$\delta_j = \sum_{i=1}^{k} g_i \cdot \gamma_{j,i} \cdot \overline{g}_i \quad mod\ g \tag{14}$$

$\mathcal{SI}_{CRA}.3$: Transform the digits of the p-adic expansion already obtained, according to the relation (12);

Figure 5: Stepwise CRA Algorithm.

Pure *CRA* Algorithm ($\mathcal{P}_{CRA}$)

$\mathcal{P}_{CRA}$.1: From each Hensel code $(\gamma_i, exp_{\gamma_i})$ compute in $\mathbb{Z}_{p_i^r}$

$$\overline{q}_i = \sum_{j=0}^{r-1} \gamma_{j,i} p_i^j = \gamma_{0,i} + \gamma_{1,i} p_i + \ldots + \gamma_{r-1,i} p_i^{r-1}$$

$\mathcal{P}_{CRA}$.2: Compute the moduli p_i^r;

$\mathcal{P}_{CRA}$.3: Apply the CRA as it is given in Theorem 6 to the values $\overline{q}_i = s_i$ and to the moduli ${g_i}^r$ to obtain the unique $\overline{q} \in \mathbb{Z}_{g^r} = s$. Then the summation (13) becomes:

$$\overline{q} = \sum_{i=1}^{k} (g_i)^r \cdot \overline{q}_i \cdot |((g_i)^r)^{-1}|_{g^r} \quad mod\ g^r \tag{15}$$

where ${g_i}^r = (M/m_i)$, $\overline{q}_i = s_i$ and $|((g_i)^r)^{-1}|_{g^r} = T_i$.

Figure 6: Pure CRA Algorithm.

complexity of $O_B(r^2(k\ log_2 p)^{log_2 3})$, where O_B indicates, as usual, the order of magnitude under the bitwise computational model described in (Aho et al., 1975). In (Limongelli, 1993) has been improved by parallelizing some steps which occur in the backward mapping of the former algorithm, i.e. CRA and multiplication ($\mathcal{S}_{CRA}$). The resulting asymptotic computational complexity is

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

The resulting computational complexity of the algorithm $\mathcal{P}_{CRA}$ is

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

In the following the computational complexities of the three algorithms are shown:

$$\begin{array}{ccc} \mathcal{S}_{CRA} & \mathcal{SI}_{CRA} & \mathcal{P}_{CRA} \\ O_B(r^2(k\ log_2 p)^{log_2 3}) & O_B(r\ (k\ log_2 p)^{log_2 3}) & O_B((r\ k\ log_2 p)^{log_2 3}) \end{array} \tag{16}$$

We have to stress that while $\mathcal{S}_{CRA}$ step could eventually dominate, in terms of time, the rest of the computation, the $\mathcal{SI}_{CRA}$ is such that its computational complexity never dominates the asymptotic behaviour of the whole algorithm. Then, $\mathcal{SI}_{CRA}$ algorithm provides a better behaviour when it is applied to problems, where the number of operations (assuming that n input rational numbers are involved) needed to reach the solution is at most n^2. Such a class of problems is actually wide: it contains for example a large set of operations over matrices and over polynomials.

We have said that the actual problem for parallel computation is the heavy computational complexity of the recovering step, performed by CRA application. In (Limongelli, 1993) has been shown that the computational complexity of the $\mathcal{SI}_{CRA}$ does not dominate any more the cost of the parallel computations themselves.

If we want to compare the performance of the parallel algorithm w.r.t. the sequential one, the table below 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) ($k = r = log_2 p$) we note that the complexity of the parallel algorithm decreases of a factor $k^2 = r^2$. It is due to the dependency, introduced by (6), of the code length in the sequential algorithm ($\overline{r}$) by a factor k, w.r.t. the code length r adopted by the parallel algorithms.

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
$O_B(m(r^{4.77}\ log_2 r))$	$= O_B(m(r^{4.77}\ log_2 r))$	$= O_B(m(r^{4.77}\ log_2 r))$	$u = 1$
$O_B(m^u\ r^5)$	$> O_B(m(r^{4.77}\ log_2 r))$	$= O_B(m(r^{4.77}\ log_2 r))$	$u = 2$
$O_B(m^u\ r^5)$	$> O_B(m^u\ r^3)$	$= O_B(m^u\ r^3)$	$u > 2$
under the assumption (iv.1),			

and

Seq. Alg.	**Par. Alg.**	**Imp. Par. Alg.**	u
$O_B(m(r^{4.77}\ log_2 r))$	$> O_B(r^{5.18})$	$> O_B(m\ r^{3.18}\ log_2 r))$	$u = 1$
$O_B(m^u\ r^5)$	$> O_B((r^{5.18})$	$> O_B(m^u\ r^3)$	$u = 2$
$O_B(m^u\ r^5)$	$> O_B(m^u\ r^3)$	$= O_B(m^u\ r^3)$	$u > 2$
under the assumption (iv.2).			

This result allows to appreciate how better efficiency is reached by the $\mathcal{SI}_{CRA}$ algorithm w.r.t. the algorithm $\mathcal{S}_{CRA}$. Moreover it supports a final evaluation of the general parallel approach w.r.t. the sequential one.

6 Experimental results

In this section we describe the implementation of parallel p-adic arithmetic, to the problem of integer exponentiation. In particular we focus the attention on the recovery step which is the "weak step" of the resolution algorithm.

The algorithm is written in PACLIB which has been designed at RISC-institute especially suitable for the implementation of computer algebra algorithms. This system combines the features of the SACLIB library (Collins & other, 1992; Buchberger & other, 1993) with those of μSystem (Buhr & Stroobosscher, 1990). SACLIB is a library of computer algebra algorithms written in C. It is based on a heap management kernel with automatic garbage collections. μSystem is a package supporting light weight processes running under UNIX on shared memory multiprocessors and on workstations.

When using the parallel p-adic approach to parallelize rational number arithmetic, the recovery of a unique result from the results in the homomorphic images is the most time consuming part. Therefore, we compare two algorithms that use two different ways of performing this recovery step. Both variants ($\mathcal{SI}_{CRA}$ and $\mathcal{P}_{CRA}$) have been implemented and their dynamic behaviour has been evaluated.

In the discussion of the dynamic behaviour we concentrate on the case of integer exponentiation and all timings will refer to this case. The reason for choosing this special case is the fact that almost all the implicit parallelism of the p-adic approach lies in the parts that suffice to perform an integer exponentiation (see Figure 2). Furthermore, it can be easily extended to the general rational case by just adding a final backward mapping step. As we stated in Section 3. in this problem belongs to the class where the size of input data is definitely smaller than the size of the output data. Hence the recovery step will dominate the computation, but it is useful to stress the recovery step for a detailed comparison of these different versions of the algorithm.

Fig. 7 and Fig. 8 show the dynamic behaviour and the utilization of Algorithm based on $\mathcal{SI}_{CRA}$. In the first graph the horizontal axis represents time and the vertical axis represents the different tasks. In the second graph again the horizontal axis represents time and the vertical axis represents the number of active processes.

In this figure the applications of the CRA can be seen as blocks of parallel processes. In the above example 87654^{500} was computed with $k = 120$. This delivered a Hensel code length of $r = 6$, yielding 5 blocks of parallel processes. As for all timings mentioned in this paper, 16 processors have been used. Since k is bigger than the number of processors, many short processes have to be executed. This can be seen from the presence of short lines, instead of long straight lines, inside the blocks. The overhead for starting many parallel processes means an additional loss of efficiency. However, if one uses a smaller k the value for r becomes larger, which means that there are more sequential steps necessary.

Contrary to the previous algorithm, tyhe one based on $\mathcal{P}_{CRA}$ is founded on

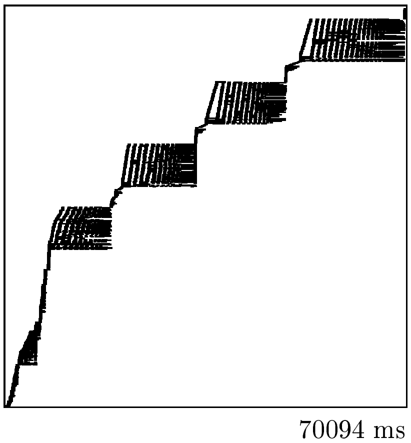

Figure 7: Dynamic Behaviour of the Algorithm based on $\mathcal{SI}_{CRA}(firstpart)$. The orizontal axis represents time and the vertical axis represents the parallel tasks.

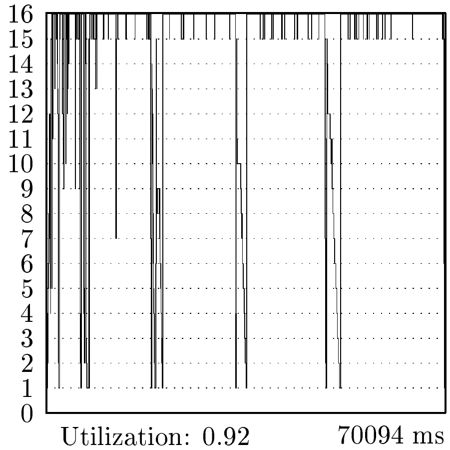

Figure 8: Dynamic Behaviour of the Algorithm based on $\mathcal{SI}_{CRA}$ (second part). In this case the vertical axis represents the number of active processes.

the idea of applying the CRA only once on the list of recovered Hensel codes d_i and the moduli m_i. All d_i and m_i can be computed in parallel as their values do not depend on each other. However, the efficiency of this part can be improved by computing blocks of d_i and m_i in each parallel process.

However, the evaluation of d_i and m_i is rather cheap. Timings have shown that these computations require only about 3% of the time needed for the CRA. Therefore, the efficiency of this approach mainly depends on the efficiency of the parallel CRA.

This comparison shows that in practice the new algorithm based on $\mathcal{P}_{CRA}$, applies the CRA only once on big numbers, is more efficient than the previously known Algorithm $\mathcal{SI}_{CRA}$, which uses a stepwise CRA but operates on smaller numbers. This is due to the fact that the Algorithm $\mathcal{P}_{CRA}$ requires a lower amount of synchronization and therefore is better suited for a parallel execution than Algorithm $\mathcal{SI}_{CRA}$. This fact is not captured in the asymptotic complexity analysis given in (16) as details like the overhead for the creation of parallel processes are omitted there.

Although only algorithms for rational number exponentiation have been implemented and analyzed, these results hold for general p-adic algorithms as the recovery step is independent of the computation that is performed on the rational numbers.

The big advantage of an algorithm based on this theorem is the fact that it is inherently parallel as all summands in the definition of s (see Equation (13)) can be evaluated in parallel. Furthermore, the combination of the intermediate results to the final result is rather cheap as only $n-1$ additions mod M have to be performed.

Figures 9 and 10 show the dynamic behaviour and the utilization of Algorithm in the evaluation of 87654^{222} with $k = 80$, delivering a Hensel code length of $r = 5$. That input values are different to those for algorithm based on $\mathcal{SI}_{CRA}$ in order to see the parallel parts in the execution better.

Contrary to the $\mathcal{SI}_{CRA}$, there are only twice as much processes as processors necessary in the main step of the algorithm $\mathcal{P}_{CRA}$. This minimizes the overhead for creating parallel processes. Furthermore, as these processes are completely independent there is no synchronization necessary. This can be seen in figures 9 and 10 where now the main block consists of straight lines instead of interrupted lines as in figures 7 and 8.

7 Final remarks

We have shown a parallel model for p-adic computation and we have analyzed the computational complexity of this model, comparing from both a theoretical and practical point of view the algorithms $\mathcal{SI}_{CRA}$ and $\mathcal{P}_{CRA}$ which solve the most delicate step of all the computation. As shown in (16) the $\mathcal{SI}_{CRA}$ algorithm provides a better behaviour when it is exploited for the solutions of

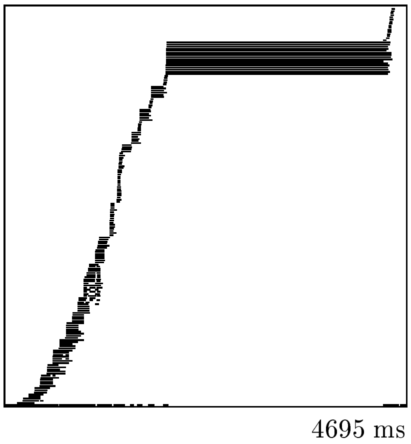

Figure 9: Dynamic Behaviour of the Algorithm based on $\mathcal{P}_{CRA}$ (first part).

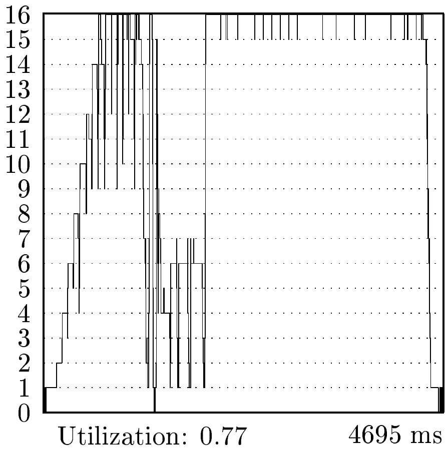

Figure 10: Dynamic Behaviour of the Algorithm based on $\mathcal{P}_{CRA}$ (second part).

problems in which $u \leq 2$. Actually there exists a wide class of such a 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 compute by N^3 multiplications. So the problem belongs to the cited class, since the number of operations is at most $4 \cdot N^4$. The computation of determinant, the matrix inversion and the LUP decomposition are also examples of problems which belong to such class: these problems are reducible to matrix multiplication as shown in (Aho et al., 1975). Other problems in such class are the polynomial multiplication, division and reciprocal see always (Aho et al., 1975). Let us consider, for example, the case of polynomial multiplication. Let $\sum_{i=0}^{d_a} a_i \cdot x^i$ and $\sum_{i=0}^{d_a} b_i \cdot x^i$ 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 following the Horner's rule, 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.

The theoretical analysis of these two algorithms does not agree with the practical one; this is due to the fact that the algorithm $\mathcal{P}_{CRA}$ requires a smaller amount of synchronization with respect to the algorithm $\mathcal{SI}_{CRA}$ its run time behaviour is better.

Note that in the implementation of the integer exponentiation a backward mapping of the result is not necessary as it is guaranteed that the resulting number is an integer number. However, other timings have shown that the backward mapping that is required for the rational case needs about half the time of the overall computation.

Therefore, after having parallelized the main steps of the p-adic approach the sequential backward mapping step becomes the dominant part. As for this backward mapping the application of an EEA is necessary, which is inherently sequential, it is very difficult to parallelize this part. New recent improvements in parallelizing the Euclidean algorithm, (Jebelean, 1993) can be a starting point for the parallelization of the extended version of this algorithm namely EEA.

References

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

Buchberger, B. (1970). An algorithm for finding the basis for the residue class ring of a zero-dimensional polynomial ideal. In Lloyd, J. W. (Ed.), *Ae-*

quationes Math.:374-383. Ph.D.Thesis, Math. Inst. Univ. of Innsbruck, Austria 1965.

Buchberger, B., & Loos, R. (1983). Algebraic simplification. In Buchberger B., Collins E.G., L. R. (Ed.), *Computer Algebra - Symbolic and Algebraic Computation:11-43* Wien, New York. Springer Verlag.

Buchberger, B., & other (1993). A SACLIB 1.1 user's guide. Tech. rep., RISC-Linz, Johannes Kepler University, Austria. RISC-Linz Report Series n. 93-19.

Buhr, A., & Stroobosscher, R. A. (1990). Providing light-weight concurrency on shared-memory multiprocessors computers running unix. *Software - Practice and Experience, 20(9):929-964.*

Colagrossi, A., & Limongelli, C. (1988). Big numbers p-adic arithmetic: a parallel approach. In *Proc. AAECC-6*, LNCS Series n. 357:169-180 Berlin, Heidelberg. Springer Verlag.

Colagrossi, A., Limongelli, C., & Miola, A. (1995). p-adic arithmetic as a tool to deal with power series. In Miola, A., & Temperini, M. (Eds.), *Advances in Design of Symbolic Computation Systems*, Vol. This volume of *Texts and Monographs in Symbolic Computation.* Springer Verlag, Wien New York.

Collins, G. E. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages II G. Conference Kaiserslautern*, Vol. 33:134-183 of *LNCS* Berlin-Heidelberg-New York. Springer Verlag.

Collins, G. E., & other (1992). A SACLIB primer. Tech. rep., RISC-Linz, Johannes Kepler University, Austria.

Dittenberger, K. (1987). An efficient method for exact numerical computation. Diploma Thesis, Johannes Kepler University, RISC-Linz Institute Austria.

Gregory, R., & Krishnamurthy, E. (1984). *Methods and Applications of Error-Free Computation.* Springer Verlag, New York Inc.

H.W., L. (1993). A parallel chinese remainder algorithm on a shared memory multiprocessors. Tech. rep., RISC-Linz, Johannes Kepler University, Austria.

Jebelean, T. (1993). Improving the multiprecision euclidean algorithm. In *International Symposium on Design and Implementation of Symbolic Computation Systems (DISCO '93)*, Vol. 722:45-58 of *LNCS* Berlin Heidelberg. Springer Verlag.

Knuth, D. (1981). *The Art of Computer Programming*, Vol. 2. Addison Wesley Publishing Company, Reading Massachussets.

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

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

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

Limongelli, C. (1993). On an efficient algorithm for big rational number computations by parallel p-adics. *J. of Symb. Comput., 15,2:181-197.*

Wang, D. (1991). Characteristics set and zero structure of polynomial sets. Tech. rep., RISC-Linz, Johannes Kepler University, Austria.