Parallel Computation of Involutive and Gröbner Bases

Vladimir P. Gerdt and Denis A. Yanovich

Laboratory of Information Technologies
Joint Institute for Nuclear Research
141980 Dubna, Russia
gerdt@jinr.ru yan@jinr.ru

Abstract. In this paper we describe a parallel version of the algorithm for constructing polynomial involutive bases. Having computed an involutive basis, the algorithm can also output the reduced Gröbner bases without any extra computational costs. This algorithm is an improved version of our previous parallel algorithm specialized for Janet bases. We implemented the improved version for Janet bases too, and in such a way that our parallel code can compute these bases not only for modular coefficients as in the previous implementation but also for long integers. We illustrate the experimental efficiency of our new implementation by the standard Gröbner bases software benchmarks.

1 Introduction

In the last decade several attempts have been undertaken [1–3] on experimental parallelization of the classical Buchberger algorithm [4] for computing Gröbner bases. However, because of a very strong dependence of the computational costs on the selection strategy for $S-$polynomials these attempts did not reveal a reasonable scalability of the parallelization. A selection strategy that is heuristically good for the sequential algorithm, such as "sugar" [5], can easily be destroyed by a parallelization. Preserving such a strategy [2] leads to extra overheads which tends to annihilate potential advantages of the parallelization.

In [6] we showed that a slight modification of a sequential involutive algorithm for computing minimal Janet bases [7] reveals its natural and effective parallelism. Experimental demonstration of this property, however, was restricted to modular computation since our implementation in [6] was not able to work in parallel with *GNU Mutiprecision library* [8] to handle long integers.

In the present paper we describe a more general parallel algorithm which computes an involutive basis for an arbitrary noetherian and constructive division as defined in [9] and for an arbitrary admissible monomial order. As well as in our previous papers [6, 7], the data structures used in the algorithm define a reduced Gröbner basis as the internally fixed part of the involutive basis. Thus, the reduced Gröbner basis can be immediately output after termination of the algorithm.

In addition to the algorithmic modifications improving the parallelization features, the below algorithm description contains some optimizations which are also relevant to sequential computation of involutive bases.

As before, we use the threads model of parallel programming and implemented the algorithm for Janet division with threads running on the same computer environment as in [6]. But this time we modified the previous parallel version of the algorithm in such a way that provided with the latest versions of the Linux based software the implementation, allows to manipulate with long integers. For the present, due to some peculiarities of our parallel codes directed to increase its efficiency, our actual implementation is restricted to the degree-lexicographical and degree-reverse-lexicographical Janet and Gröbner bases.

Our first experimental results obtained with the new parallel implementation revealed substantially more gain for integer coefficients in comparison with that we profited in [6] for modular coefficients.

2 Preliminaries

In this paper we use the following definitions and notations:

$\mathbb{X} = \{x_1, \ldots, x_n\}$ is the set of polynomial variables.
$X \subseteq \mathbb{X}$ is a subset (possibly empty) of the variables.
$\mathbb{R} = \mathbb{K}[\mathbb{X}]$ is a polynomial ring over zero characteristic field $\mathbb{K}$.
$Id(F)$ is the ideal in $\mathbb{R}$ generated by $F \subset \mathbb{R}$.
$\mathbb{M}_X$ is the monoid of monomials, i.e. power products, in variables in $X \subseteq \mathbb{X}$ that is a submonoid of $\mathbb{M} \equiv \mathbb{M}_{\mathbb{X}}$.
$\deg_i(u)$ is the degree of x_i in $u \in \mathbb{M}$.
$\deg(u) = \sum_{i=1}^{n} \deg_i(u)$ is the total degree of u.
$\succ$ is an admissible monomial ordering such that $x_1 \succ x_2 \succ \cdots \succ x_n$.
$u \mid v$ is the conventional divisibility relation of monomial v by monomial u. If $u \mid v$ and $\deg(u) < \deg(v)$, i.e. u is a *proper divisor* of v, we shall write $v \sqsupset u$.
$\mathrm{lm}(f)$ and $lt(f)$ are the leading monomial and the leading term of $f \in \mathbb{R} \setminus \{0\}$, respectively.
$\mathrm{lm}(F)$ is the leading monomial set for $F \subset \mathbb{R} \setminus \{0\}$.
$\mathrm{card}(F)$ is the cardinality of set $F \in \mathbb{R}$.
$\mathrm{lcm}(u, v)$ is the least common multiple of monomials $u, v \in \mathbb{M}$.

An *involutive separation* L of variables is said to be defined on $\mathbb{M}$ if for any finite monomial set $U \subset \mathbb{M}$ and for any $u \in U$ there is defined a subset $M(u, U) \subseteq \mathbb{X}$ of variables generating monoid $L(u, U) \equiv \mathbb{M}_{M(u,U)}$ such that

1. $u, v \in U,\ uL(u, U) \cap vL(v, U) \neq \emptyset \Longleftrightarrow u \in vL(v, U)$ or $v \in uL(u, U)$.
2. $v \in U,\ v \in uL(u, U) \Longleftrightarrow L(v, U) \subseteq L(u, U)$.
3. $V \subseteq U \Longrightarrow L(u, U) \subseteq L(u, V)\ \ \forall u \in V$.

Variables in $M(u, U)$ are called $(L-)$*multiplicative* for u and those in $NM(u, U) \equiv \mathbb{X} \setminus M(u, U)$ are $(L-)$*nonmultiplicative* for u, respectively. If $w \in uL(u, U)$ then u is *involutive divisor* of w, and in this case we shall also write $u \mid_L v$.

Below we shall explicitly put down the subscript L for both sets of $L-$multiplicative $M_L(u, U)$ and $L-$nonmultiplicative $NM_L(u, U)$ variables for $u \in U$. From the above definition it follows that an involutive separation generates the *involutive division* [9] and vice-versa.

A finite polynomial set F is $L-$*autoreduced* if each term in every $f \in F$ has no $L-$divisors among $\mathrm{lm}(F) \setminus \{\mathrm{lm}(f)\}$. The $L-$*(involutive) normal form* $NF_L(p, F)$ of $p \notin F$ modulo $L-$autoreduced set F is given by

$$NF_L(p, F) = \tilde{p} = p - \sum_{ij} \alpha_{ij} m_{ij} g_j$$

where $\alpha_{ij} \in \mathbb{K},\ g_j \in F,\ m_{ij} \in L(\mathrm{lm}(g_j), \mathrm{lm}(F)),\ \mathrm{lm}(m_{ij} g_j) \preceq \mathrm{lm}(p)$, and there are no monomials in $\tilde{p}$ which have $L-$divisors in $\mathrm{lm}(F)$. If $\mathrm{lm}(f)$ $(f \notin F)$ has no $L-$divisors in $\mathrm{lm}(F)$ then we shall call it $L-$*head reduced* modulo F and write this as $f = HNF_L(f, F)$ where $HNF_L(f, F)$ denotes the $L-$*head normal form* of f modulo F.

Given an ideal $I \subset \mathbb{R}$, an involutive division L and monomial order $\succ$, a finite $L-$autoreduced subset $G \subset \mathbb{R}$ generating I is called its $L-$*(involutive) basis* if

$$(\forall f \in I)\ (\exists g \in G)\ [\ \mathrm{lm}(g) \mid_L \mathrm{lm}(f)\ ]$$

If division L is continuous [9] this is equivalent to

$$(\ \forall f \in G\ )\ (\ \forall x_i \in NM_L(\mathrm{lm}(f), \mathrm{lm}(G))\ )\ \ [\ NF_L(x_i \cdot f, G) = 0\ ]$$

Below we shall often write $NM_L(f, F)$ for $NM_L(lm(f), lm(F))$.

The product $x_i \cdot f$ of polynomial $f \in F$ and $x_i \in NM_L(f, F)$ is called *nonmultiplicative prolongation* of f, and construction of involutive bases is often called *completion.* These terms came from the involution theory of differential equations [10]. An involutive differential system has all its certain differential consequences called *integrability conditions* incorporated in the system. Therefore, to construct an infolutive form of a differential system one has to complete the system with all its integrability conditions. A completion procedure includes differentiations of equations in the system with respect to independent variables. These differentiations are called prolongations. A single derivation of a differential equation corresponds to multiplication by a variable in the case of multivariate polynomial. For polynomial systems the role of integrability conditions is played by non-trivial critical pairs ($S-$polynomials) that are to be processed in the course of the Buchberger algorithm [4].

One of classical, and widely used in the involutivity analysis of differential equations, separations of variables introduced in [11] generates the involutive division called *Janet division* [9]. Given a monomial order $\succ$, a finite set $F \subset \mathbb{R}$, and a polynomial $f \in F$, the *Janet separation* of variables into multiplicative and nonmultiplicative with respect to f is defined as follows.

For each $1 \leq i \leq n$ divide F into groups labeled by non-negative integers $d_1, \ldots, d_i$

$$[d_1, \ldots, d_i] = \{ \ f \ \in F \mid d_j = \deg_j(\operatorname{lm}(f)), \ 1 \leq j \leq i \ \}.$$

x_1 is (Janet) multiplicative for $f \in F$ if $\deg_1(\operatorname{lm}(f)) = \max\{\deg_1(\operatorname{lm}(g)) \mid g \in F\}$. For $i > 1$ x_i is multiplicative for $f \in [d_1, \ldots, d_{i-1}]$ when $\deg_i(\operatorname{lm}(f)) = \max\{\deg_i(\operatorname{lm}(g)) \mid g \in [d_1, \ldots, d_{i-1}]\}$.

An involutive basis is a (generally redundant) *Gröbner basis* [9]. Similarly to a reduced Gröbner basis, a monic *minimal involutive basis* [12] is unique.

3 Parallel Involutive Algorithm

In this section we describe a parallel version of the involutive basis algorithm which is an improved version of the parallel Janet basis algorithm of paper [6] generalized to an arbitrary noetherian and constructive division [9].

As well as in [6, 7] we associate with every polynomial $f \in F$ in the intermediate polynomial set $F \in \mathbb{R}$ the triple $p = \{f, \, u, \, vars\}$ with

$$\begin{array}{lll} \operatorname{pol}(p) & = f & \text{is the polynomial } f \text{ itself,} \\ \operatorname{anc}(p) & = u & \text{is the leading monomial of a polynomial ancestor of } f \text{ in } F, \\ \operatorname{nmp}(p) & = vars & \text{is a (possible empty) subset of variables.} \end{array}$$

The *ancestor* of p is a polynomial $g \in F$ of the smallest $\deg(\operatorname{lm}(g))$ among elements in F satisfying $\operatorname{lm}(g) \mid \operatorname{lm}(p)$. This means that for the ancestor g the equality $\operatorname{anc}(g) = \operatorname{lm}(g)$ always holds.

If an intermediate polynomial p arose in the course of the below algorithm and has a proper ancestor g, then p has been obtained from g by examining a sequence of head irreducible nonmultiplicative prolongations.

The set $vars$ accumulates those nonmultiplicative variables for p which have been already used in the algorithm for construction of nonmultiplicative prolongations.

The second and the third elements in the above triple structure keep those parts of the completion history which serve, respectively, to avoid unnecessary reductions by means of the involutive analogues of Buchberger's criteria and do not consider useless repeated prolongations.

The main algorithm **ParallelInvolutiveBasis** (given on the subsequent page) uses the multithread model of parallel programming. Given a noetherian continuous and constructive division [9] L, an admissible monomial ordering $\succ$ and the specified maximal number of threads K_{thr} to be used, it computes a minimal $L-$involutive basis of the ideal generated by the input polynomial set F.

In lines 25, 27 of the algorithms and below, where no confusion can arise, we simply refer to the triple set T as the second argument in NM_L, NF_L and HNF_L instead of the set of polynomials contained in T. Sometimes we also refer to the triple p instead of its polynomial.

Algorithm: ParallelInvolutiveBasis $(F,\ L,\ \prec, K_{thr})$

```
Input: $F \in \mathbb{R}\setminus\{0\}$, a finite set; $\prec$, an admissible ordering;$L$, an involutive division; $K_{thr} \geq 1$,
    the maximal number of threads
Output: $G$, a minimal $L-$basis of $Id(F)$
 1: choose $f \in F$ with the lowest $\mathrm{lm}(f)$ w.r.t. $\succ$
 2: $T := \{f, \mathrm{lm}(f), \emptyset\}$
 3: $Q := \{\{g, \mathrm{lm}(g), \emptyset\} \mid g \in F \setminus \{f\}\} \cup \{\{f \cdot x, \mathrm{lm}(f)x, \emptyset\} \mid x \in NM_L(f, \{f\})\}$
 4: sort $Q$ in increasing order of its polynomials w.r.t. $\succ$
 5: $S := \emptyset$
 6: $P := \{\ q_i \in Q \mid i \leq K_{thr}\ \}$
 7: $Q := Q \setminus P$
 8: create $\min\{\mathrm{card}(P), K_{thr}\}$ threads WorkerThread$(P, T, S, mutex)$
 9: wait for return of all threads
10: $Q := Q \cup S$
11: while $Q \neq \emptyset$ do
12:   choose $p \in Q$ s.t. $(\nexists q \in Q \setminus \{p\})$ $[\ \mathrm{lm}(\mathrm{pol}(p)) \sqsupset \mathrm{lm}(\mathrm{pol}(q))\ ]$
13:   if $\mathrm{lm}(\mathrm{pol}(p)) = 1$ then
14:     return $\{1\}$
15:   else
16:     $Q := Q \setminus \{p\}$
17:     if $\mathrm{lm}(\mathrm{pol}(p)) = \mathrm{anc}(p)$ then
18:       for all $\{\ r \in T \mid lm(\mathrm{pol}(r)) \sqsupset \mathrm{lm}(\mathrm{pol}(p))\ \}$ do
19:         $Q := Q \cup \{r\}$;    $T := T \setminus \{r\}$
20:       od
21:     fi
22:     $\mathrm{pol}(p) := \mathbf{NF_L}(\mathrm{pol}(p), T)$
23:   fi
24:   $T := T \cup \{p\}$
25:   for all $q \in T$ and $x \in NM_L(\mathrm{pol}(q), T) \setminus \mathrm{nmp}(q)$ do
26:     $Q := Q \cup \{\ \{\mathrm{pol}(q) \cdot x, \mathrm{anc}(q), \emptyset\}\ \}$
27:     $\mathrm{nmp}(q) := \mathrm{nmp}(q) \cap NM_L(\mathrm{pol}(q), T) \cup \{x\}$
28:   od
29:   $S := \emptyset$
30:   sort $Q$ in increasing order of its polynomials w.r.t. $\succ$
31:   $P := \{\ q_i \in Q \mid i \leq K_{thr}\ \}$
32:   $Q := Q \setminus P$
33:   create $\min\{\mathrm{card}(P), K_{thr}\}$ threads WorkerThread$(P, T, S, mutex)$
34:   wait for return of all threads
35:   $Q := Q \cup S$
36: od
37: return $G := \{\ \mathrm{pol}(f) \mid f \in T\ \}$
```

Our previous parallel algorithm in [6] exploited the "thread-pool" model when all the threads created wait until they get information (a polynomial in our case) for its processing (involutive head reduction). The threads are living during the whole program running time, and, hence, consuming the computer resources even if they are not active.

In the present algorithm another model of multi-threading which we shall call "boss-workers" is used. The main procedure ("boss") creates the threads just when there is a need in them (lines 8 and 33) and then wait (e.g. sleeping) until all the threads created finish their job. An advantage of this model over that we used previously lies in simplification of its programming, avoidance deadlocks and reduction of the mutex delays.

The structure of the algorithm **ParallelInvolutiveBasis** which realizes multi-threading, basically is very similar to the algorithm in [6]. This time, however, an involutive division is included in the input parameters what makes the algorithm applicable for construction of other involutive bases and not only Janet ones. The multi-threading parallelization is realized in lines 8 and 33 where the working threads are created. Their number cannot exceed the maximal number of threads K_{thr} specified as an input parameter, but may be less, if K_{thr} is larger than the number of elements in P to be processed. Below we describe subalgorithms **WorkerThread** and $\mathbf{NF_L}(\mathrm{pol}(p), T)$ that are invoked in lines 8, 22 and 33. Respectively, they control working threads and perform the involutive tail reduction.

The sorting of elements in triple set Q that is done in lines 4 and 30 is an auxiliary operation that revealed its heuristical profit in our implementation done for Janet division and for the degree-lexicographical and degree-reverse-lexicographical orders when an element in $p \in Q$ selected in line 12 has the minimal $\deg(\mathrm{lm}(\mathrm{pol}(p)))$. Note that in both cases the sorting of triples in Q is done with respect to the leading monomials of their first elements.

What is essentially different in algorithm **ParallelInvolutiveBasis** in comparison with those in papers [6, 7, 12] is the replacement of condition

$$\mathrm{lm}(\mathrm{pol}(r)) \succ \mathrm{lm}(\mathrm{pol}(p))$$

for the transfer of elements from T to Q by the condition

$$\mathrm{lm}(\mathrm{pol}(r)) \sqsupset \mathrm{lm}(\mathrm{pol}(p))$$

in line 18. If this replacement still preserves the minimality of the output basis then it is clearly an important optimization since it decreases the number of transfers done in the course of algorithm and the related recalculations.

To show correctness of the replacement we refer to the main idea of the correctness proof for the algorithm in paper [12]. It follows that in order to prove minimality of the output $L - basis$ G of algorithm **ParallelInvolutiveBasis** it is sufficient to show that after every execution of line 24 the monomial set $U = \{\ \mathrm{lm}(\mathrm{pol}(p)) \mid p \in T\ \}$ is a subset of the minimal $L-$basis of the monomial ideal generated by U. A polynomial set whose leading monomial set satisfies this property we shall call *compact* (cf. [12]).

We claim that when the **while**-loop runs the very first time this is clearly holds. Indeed, right before step 24 the triple set T contains the single element $\{f, \mathrm{lm}(f), \emptyset\}$ formed in line 2. Let polynomial q ($q = \mathrm{pol}(p)$) is that calculated at step 22, that is $q = NF_L(q, \{f\})$. Denote $\mathrm{lm}(f)$ and $\mathrm{lm}(q)$ by u and v, respectively. If neither $u \mid v$ nor $v \mid u$ the set $\{u, v\}$ is apparently compact. Furthermore, $u \sqsupset v$ cannot hold by admissibility of $\prec$ and the definition of $NF_L(\mathrm{pol}(p), \{f\})$ given in Sect.2. If $u \mid v$, then v/u must contain $L-$nonmultiplicative variables for u. Monomial v does not belong the minimial $L-$basis for the ideal generated by $\{u, v\}$ only if there exists $x_i \in NM_L(u, \{u\})$ such that $v \sqsupset u{\cdot}x_i$. But in this case, by the selection strategy used in line 12, the triple $\{h, \mathrm{lm}(h), \emptyset\}$ with $\mathrm{lm}(h) = u \cdot x_i$ inserted to Q in line 3 must have been be added to T before p. This proves the claim.

It is obvious that displacement done in lines 18-19 never breaks compactness of the polynomial set in T. Assume now that the property of compactness holds for the polynomial set $G = \{\mathrm{pol}(f\} \mid f \in T\ \}$ but is destroyed just in line 24. In accordance with property 3 in definition (Sect.2) of involutive separation for L, this may only happen if there exists a polynomial $h \in G$ such that its nonmultiplicative prolongation $h \cdot x_j$ whose leading term is $L-$irreducible modulo G satisfies $\mathrm{lm}(\mathrm{pol}(p)) \sqsupset h \cdot x_j$. But then by exactly the same reasoning as above, the triple in Q with such a prolongation must have been selected and added to the triple set T earlier than p. The obtained contradiction proves correctness of the replacement.

Thereby, the optimization done in line 18 does not break the algorithm correctness, and apparently its termination too. Therefore, both correctness and termination of the algorithm follow from those of the algorithms in [6, 7, 9] if this is true also for the subalgorithms **WorkerThread** and $\mathbf{NF_L}(\mathrm{pol}(p), T)$.

Consider now these these subalgorithms. The subalgorithm **WorkerThread** controls the working threads processing the triple sets P, T, S initially created in lines 2, 5, and 6, of the main algorithm and accessible by each "worker" thread.

Algorithm: WorkerThread($P, T, S, mutex$)

Input: P, S and T, sets of triples
Output: S, a set with polynomials which Janet head reduced modulo T

```
 1: while $P \neq \emptyset$ do
 2:    lock($mutex$)
 3:    choose $p \in P$
 4:    $P := P \setminus \{p\}$
 5:    unlock($mutex$)
 6:    $h := \mathbf{HNF_L}(p, T)$
 7:    if $h \neq 0$ then
 8:       lock($mutex$)
 9:       if $\mathrm{lm}(\mathrm{pol}(p)) \neq \mathrm{lm}(h)$ then
10:          $S := S \cup \{h, \mathrm{lm}(h), \emptyset\}$
11:       else
12:          $S := S \cup \{p\}$
13:       fi
14:       unlock($mutex$)
15:    fi
16: od
17: return $S$
```

The fourth argument of the algorithms is mutex (**mut**ual **ex**clusion object) that is a system object used to assure that simultaneous access to global shared resources (such as P, T, S) is avoided. Mutex is created by the starting program. Then any created thread that needs the resources must lock the mutex from other threads while it is using the resource. The mutex must be unlocked when the shared data are no longer needed.

The commands in the pseudocode form are shown for a thread performing a head involutive reduction of a polynomial in P modulo polynomial set in T by invoking the subalgorithm $\mathbf{HNF_L}(p, T)$. For nonzero $L-$head reduction the corresponding triples are added to S. With all this going on, if the leading term of the taken polynomial h was affected by $L-$reduction, then it loses the link to his former ancestor and becomes its own ansestor in the upgraded set S. Otherwise, the second and the third elements in the triple are inherited from the input triple.

Therefore, correctness and termination of algorithm **WorkerThread** is determined by those of $\mathbf{HNF_L}(p, T)$. The last subalgorithm as well as subalgorithm $\mathbf{NF_L}$ is exactly that considered in [6, 7] for Janet division and extended to a general involutive division. Its termination and correctness immediately follow from the basic properties of involutive reductions [9] and correctness of the criteria used in line 8 introduced in [7, 9, 12].

For the head reducible input polynomial pol(f) what is checked in line 7, the two criteria are verified in line 8:

CriterionI(f, g) is true iff $\mathrm{anc}(f) \cdot \mathrm{anc}(g) \mid \mathrm{lm}(\mathrm{pol}(f))$.

CriterionII(f, g) is true iff $\mathrm{lm}(f) \sqsupset \mathrm{lcm}(\mathrm{anc}(f) \cdot \mathrm{anc}(g))$.

Criteria I and II follow from the Buchberger criteria [4] adapted to the involutive completion procedure. If either of the two criteria is true, then $HNF_L(\mathrm{pol}(f), T) = 0$ [7].

The last subalgorithm $\mathbf{NF_L}$ completes the $L-$head reduction by performing the involutive tail reduction and is invoked in line 22 of the main algorithm **ParallelInvolutiveJBasis**.

Algorithm: $\mathbf{HNF_L}(f,T)$

```
Input: $f = \{\mathrm{pol}(f), \mathrm{anc}(f), \mathrm{nmp}(f)\}$, a triple
       $T$, a set of triples
Output: $h = HNF_L(\mathrm{pol}(f), T)$, the $L-$head normal form of the polynomial in $f$ modulo polynomial set in $T$
 1: $G := \{\mathrm{pol}(g) \mid g \in T\}$
 2: if $\mathrm{lm}(\mathrm{pol}(f))$ is $L-$irreducible modulo $G$ then
 3:    return $f$
 4: else
 5:    $h := \mathrm{pol}(f)$
 6:    choose $g \in T$ such that $\mathrm{lm}(\mathrm{pol}(g)) \mid_L \mathrm{lm}(h)$
 7:    if $\mathrm{lm}(h) \neq \mathrm{anc}(f)$ then
 8:       if CriterionI$(f,g)$ or CriterionII$(f,g)$ then
 9:          return 0
10:       fi
11:    else
12:       while $h \neq 0$ and $\mathrm{lm}(h)$ is $L-$reducible modulo $G$ do
13:          choose $q \in G$ such that $\mathrm{lm}(q) \mid_L \mathrm{lm}(h)$
14:          $h := h - q \cdot \mathrm{lt}(h)/\,\mathrm{lt}(q)$
15:       od
16:    fi
17: fi
18: return $h$
```

Algorithm: $\mathbf{NF_L}(f,T)$

```
Input: $f$, the polynomial in a triple $p$ such that $f := HNF_L(p,T)$;
       $T$, a set of triples
Output: $h = NF_L(f,T)$, the full $L-$normal form of $h$
        modulo polynomial set in $T$
 1: $G := \{\mathrm{pol}(g) \mid g \in T\}$
 2: $h := f$
 3: while $h \neq 0$ and $h$ has a term $t$ which is $L-$reducible modulo $G$ do
 4:    choose $g \in G$ such that $\mathrm{lm}(g) \mid_L t$
 5:    $h := h - g \cdot t/\,\mathrm{lt}(g)$
 6: od
 7: return $h$
```

4 Implementation and benchmarking

To investigate experimental behavior of the algorithm **ParallelInvolutiveBasis** we developed a parallel C code on the basis of our earlier, and especially optimized for the sequential (non-parallel) computation, implementation in C described in [7]. In particular, as well as in [7] we exploited the Janet trees as data structures for T and unsorted lists for Q, P, S.

The running times were measured for the degree-reverse-lexicographical monomial ordering compatible (Sect. 2) with $x_1 \succ x_2 \succ \cdots \succ x_n$ on a 2 processor Pentium III 700 Mhz PC with 1Gb RAM computer running under *Gentoo Linux* [14]. As well as in the sequential computation, to perform in the parallel code the arithmetical operations over the integer polynomial coefficients the *GNU Multiprecision Library* [13] was used.

Whereas to run the sequential code we use the `dlmalloc` memory manager, this does not work tolerably in the multi-thread computations. Similarly we could not exploit the `GC` manager. We decided thereby to use the standard `malloc` manager. This leads to experimental slowing down of the one-thread run of our parallel code in comparison with the sequential one (see Table 1) caused by the non-optimized memory access. There are also some overheads owing to organization of the inter-thread synchronization.

The timings for our parallel code in one-thread and three-thread modes[1] are summarized in the below presented Table 1. As benchmarks we use the same ones as in our previous papers [6, 7] and taken from the collections [15, 16]. These benchmarks are widely used for the testing of the modern Gröbner bases software and provide a good platform for experimental analysis of the built-in algorithms and efficiency of their implementation.

To measure a running time we used the following scheme. First, we measured individual timings for reductions done in parallel by those threads which are created in lines 8 and 33 of the main algorithm **ParallelInvolutiveBasis**. Meanwhile during this execution of the reductions the main thread timer (the timer of "boss") was halted (i.e., its timer stopped). After completion of all the reductions done, the maximal value of the individual time spent by the reduction threads was added to the main timer.
Acting so we disposed of swapping time, delay caused by addressing to the hard disk and other time delays irrelevant to the program running.

The second column of Table 1 shows the running times for our (specially optimized for the case of) sequential code. The third column gives timings for the multi-thread implementation with one "worker" thread. For experimental measurement of scalability one could compare these timings with those measured for a parallel code running on our (actually two-processor machine) in three-thread mode shown in the 4th column.

The last two columns show the relative speedup of three-thread parallel code with respect to the sequential code and one-thread code. It should be noted that in our case the three-thread mode runs in most cases faster than the optimized sequential version even if the astronomical time would taken into account.

We also performed the following empirical analysis. By running the multi-thread software on our two-processor machine we varied the number of threads and measured the processor loading. And for all that it turned out that the maximal loading took place just for three threads. This is in unison with the general prescription to have the number of the threads involved equal to the number of available processors plus one.

On some examples we even detected a super-linear growth of speedup with respect to number of threads involved. This is due to the fact that in the multi-threaded run some more short and simple polynomials were computed and added to the triple set T earlier than in the sequential run. These polynomials gave rise to another and faster chain of the successive reductions, and thus, accelerated the process of computation. One can say that in these cases the multi-thread run revealed experimentally better selection strategy for examining nonmultiplicative prolongations than that fixed in our sequential code.

5 Conclusion

If in the last line of algorithm **ParallelInvolutiveBasis** we would use the command

$$\textbf{return}\ \ G := \{\ \mathrm{pol}(f) \mid f \in T \mid\ \mathrm{lm}(f) = \mathrm{anc}(f)\ \}\,,$$

then the algorithm would return the reduced Gröbner basis. This follows immediately from our definition of ancestors as given in the beginning of Section 3. This extraction is done without extra computation, i.e. without Gröbner reductions of the output involutive basis. Therefore, our algorithm can be considered as a parallel one[2] for computation of reduced Gröbner bases. A user of our program may specify on the input which basis to be output: Janet, Gröbner or both.

The above presented first experimental results obtained with our new parallel implementation demonstrate for the standard polynomial benchmarks [15, 16] an experimental evidence of effective parallelization of our involutive algorithms.

[1] That is, with one and and three "workers" which perform the $L-$head reductions.
[2] This is true also and for the sequential algorithm [7].

As the next step in theoretical and experimental parallelization of the involutive algorithms we shall try to decrease further the overheads and to analyze applicability of programming models and software available for distributed computing.

It should be noted that two criteria used in the subalgorithm $\mathbf{HNF_L}(p, T)$ do not fully cover the Buchberger criteria as it was shown in [17]. Some of zero-redundant nonmultiplicative prolongation can be avoided by applying other criteria formulated in [17]. However, our first attempts to build-in the missing criteria in sequential implementation of the Janet basis algorithm [7] have not revealed so far any experimentally notable gain from the use of new criteria for the standard benchmarks set [15, 16]. Moreover, for most of benchmarks it led to increasing the timings caused by extra checks of the applicability conditions for the missing criteria. Perhaps this is because our current data structures [7] are not very suitable for these extra checks. By this reason so we did not include the new criteria in the present parallel version and its implementation. But we plan to continue experimental study of effects from the missing criteria for our involutive algorithms and implementing their codes.

6 Acknowledgments

One of the authors (V.G.) thanks Ralf Hemmecke for useful discussions. The research presented in this paper was partially supported by the grants 04-01-00784 from the Russian Foundation for Basic Research and 2339.2003.2 from the Russian Ministry of Science and Education.

References

1. Faugère, J.C.: Parallelization of Gröbner basis. In: *PASCO'94, Lecture Notes Series in Computing* **5**, World Scientific, Singapore (1994) 124–132
2. Attardi, G., Traverso, C.: Strategy-accurate parallel Buchberger algorithm. *J. Symb. Comp.* **11** (1996) 411-425
3. Amrheim, B., Gloor, O., Küchlin, W.: A case study of multi-threaded Gröbner basis completion. In: *Proceedings of ISSAC'96*, ACM Press (1996) 95–102
4. Buchberger, B.: Gröbner bases: an algorithmic method in polynomial ideal theory. In: *Recent Trends in Multidimensional System Theory*, N.K. Bose (ed.), Reidel, Dordrecht (1985) 184–232
5. Gianni, P., Mora, T., Niesi, G., Robbiano, L., Traverso, K.: "One sugar cube, please" or Selection strategies in Buchberger algorithm. *Proceedings of ISSAC'91*, ACM Press (1991) 49–54
6. Gerdt, V.P., Yanovich, D.A.: Parallelism in computing Janet bases. In: *Proceedings of the Workshop on Under- and Overdetermined Systems of Algebraic or Differential Equations (Karlsruhe, March 18-19, 2002)*, J.Calmet, M.Hausdorf, W.M.Seiler (Eds.), Institute of Algorithms and Cognitive Systems, University of Karlsruhe (2002) 47–56
7. Gerdt, V.P., Blinkov, Yu.A., Yanovich, D.A.: Construction of Janet bases. II. Polynomial bases. In: *Computer Algebra in Scientific Computing / CASC'01*, V.G.Ganzha, E.W.Mayr and E.V.Vorozhtsov (Eds.), Springer-Verlag Berlin (2001) 249–263
8. http://www.swox.com/gmp
9. Gerdt, V.P., Blinkov, Yu.A.: Involutive bases of polynomial ideals. *Math. Comp. Sim.* **45** (1998) 519-542
10. Calmet, J., Hausdorf, M., Seiler, W.M.: A constructive introduction to involution. In: *Proc. Int. Symp. Applications of Computer Algebra - ISACA 2000*, R. Akerkar (ed), Allied Publishers, New Delhi (2001) 33–50
11. Janet, M.: *Leçons sur les Systèmes d'Equations aux Dérivées Partielles*, Cahiers Scientifiques, IV, Gauthier-Villars, Paris (1929)
12. Gerdt, V.P., Blinkov, Yu.A.: Miniamal involutive bases. *Math. Comp. Sim.* **45** (1998) 543–560
13. `http://www.swox.com/gmp`
14. `http://www.gentoo.org`
15. Bini, D., Mourrain, B.: *Polynomial Test Suite* (1996) `http://www-sop.inria.fr/saga/POL`
16. Verschelde, J.: The Database with Test Examples. `http://www.math.uic.edu/~ jan/demo.html`
17. Apel, J., Hemmecke, Ralf.: Detecting Unnecessary Reductions in an Involutive Basis Computation. *RISC Linz Report* 02-22 (2002)

example	**sequential**	**1 thread**	**3 threads**	**speedup**	t_{seq}/t_{3th}	t_{1th}/t_{3th}
assur44	65.60	73.93	31.34	+34.26	2.09	2.36
butcher8	5.07	6.7	3.93	+1.14	1.29	1.71
chemequs	4.09	4.03	2.6	+1.49	1.57	1.55
chemkin	67.62	88.88	35.82	+31.8	1.89	2.48
cohn3	521.83	554.75	222.69	+299.14	2.34	2.49
cpdm5	4.79	10.07	5.41	-0.62	0.89	1.86
cyclic6	0.36	0.79	1.16	-0.8	0.31	0.68
cyclic7	193.6	386.89	182.86	+10.74	1.06	2.12
d1	40.98	52.2	24.88	+16.1	1.65	2.10
des18_3	0.81	1.23	0.97	-0.16	0.84	1.27
des22_24	3.14	4.43	2.57	+0.57	1.22	1.72
dl	731.31	1150.18	557.00	+174.31	1.31	2.07
eco8	1.12	2.41	1.67	-0.55	0.67	1.44
eco9	12.30	22.23	11.14	+1.16	1.10	2.00
eco10	129.73	227.92	103.99	+25.74	1.25	2.19
eco11	2168.62	2867.5	1254.34	+914.28	1.73	2.27
eco12	25712.00	34255.3	15870.8	+9841.2	1.62	2.16
extcyc5	4.36	7.7	5.89	-1.53	0.74	1.31
extcyc6	1585.63	1689.16	680.26	+905.37	2.33	2.48
f744	13.22	23.76	13.8	-0.58	0.96	1.72
fabrice24	649.14	648.78	254.39	+394.75	2.55	2.55
filter9	38.73	33.58	38.66	+0.07	1.00	0.88
i1	626.30	1000.69	371.79	+254.51	1.69	2.69
jcf26	1317.05	1325.57	515.61	+801,44	2.55	2.57
katsura7	5.87	11.28	5.85	+0.02	1.00	1.93
katsura8	71.16	119.92	53.72	+17.44	1.33	2.23
katsura9	911.09	1356.37	587.82	+323.27	1.55	2.31
katsura10	17676.90	22395.00	9809.34	+7867.56	1.80	2.28
kin1	87.57	86.29	34.46	+53.11	2.54	2.50
kotsireas	25.19	29.29	13.76	+11.43	1.83	2.13
noon6	3.01	4.56	3.88	-0.87	0.78	1.18
noon7	71.34	101.63	59.55	+11.79	1.20	1.71
noon8	3999.61	4454.96	2299.72	+1699.89	1.74	1.94
pinchon1	29.34	48.5	26.7	+2.64	1.10	1.82
rbpl	1150.77	1295.27	778.49	+372.28	1.49	1.66
rbpl24	649.10	648.54	224.74	+424.36	2.89	2.89
redcyc6	0.52	0.92	0.84	-0.32	0.62	1.10
redeco10	42.29	74.63	35.81	+6.48	1.18	2.08
redeco11	398.92	667.03	301.26	+97.66	1.32	2.21
reimer6	50.24	88.99	52.56	-2.32	0.96	1.69
reimer7	8980.36	11659.2	7547.27	+1433.09	1.19	1.55
virasoro	25.10	46.54	21.26	+3.84	1.18	2.19

Table 1. Timings (in seconds) and speedup due to parallelism.