Programming and Computer Software, Vol. 31, No. 2, 2005, pp. 73–80. Translated from Programmirovanie, Vol. 31, No. 2, 2005.

Parallel Computation of Janet and Gröbner Bases over Rational Numbers

V. P. Gerdt and D. A. Yanovich

Laboratory of Information Technologies, Joint Institute for Nuclear Research, Dubna, Moscow oblast, 141980 Russia
e-mails: gerdt@jinr.ru, yan@jinr.ru

Received May 26, 2004

Abstract—In this paper, a parallel algorithm for computation of polynomial Janet bases is considered. After construction of a Janet basis, a reduced Gröbner basis (which is a subset of the Janet basis) is extracted from it without any additional reductions. The algorithm discussed is an improved version of an earlier suggested parallel algorithm. The efficiency of a C implementation of the algorithm and its scalability are illustrated by way of the standard test examples that are often used for comparing various algorithms and codes for computing Gröbner bases.

1. INTRODUCTION

There have been several attempts [1–3] during last decade to parallelize the classical Buchberger algorithm [4] for computing Gröbner bases. However, these attempts failed because the amount of computation greatly depends on the strategy of selection of critical pairs (*S*-polynomials). Indeed, any strategy that is heuristically optimal in the case of sequential computations (i.e., a strategy that minimizes the growth of intermediate coefficients in computations over the ring of integers), such as, for example, the "sugar" strategy [5], is easily broken when reducing critical pairs in parallel. On the other hand, an attempt to implement a sequential strategy in parallel computations reduces the efficiency (scalability) of the parallelization.

The involutive approach [6, 7] gives much less possibilities for selecting nonmultiplicative prolongations compared to the number of critical pairs in the Buchberger algorithm [8]. Moreover, admissible variations of the strategy of the prolongation selection only slightly affect the efficiency of the computations, which makes the involutive algorithms very different from the Buchberger algorithm. It has been shown in [9, 10] that the involutive algorithm for constructing the minimal Janet basis [11] possesses natural and effective parallelism, which can be revealed by appropriately modifying this algorithm. Computational experiments showed high stability of the required computer resources with respect to a particular computation (strategy of selecting nonmultiplicative prolongations and placing their nonzero normal forms into the intermediate basis), which is dynamically implemented in the course of the parallelization.

However, the verification of this property and the scalability analysis were confined to computations in modular arithmetic, since the program implementation suggested in [9, 10] could not work in parallel with the GNU Multiprecision library [12] designed for mathematical operations on integers of arbitrary length.

In this paper, we suggest a parallel algorithm for computing polynomial Janet bases. Like in our previous works [9–11], we use data structures that allow us to organize internal links to elements of a reduced Gröbner basis as a subset of the Janet basis. Thus, the reduced Gröbner basis can be extracted from the Janet basis without any additional reductions.

Like in [9, 10], we use a multithread model of parallel programming, which is implemented on the same machine. However, in the given case, we use a different scheme of the multithread model, which allow individual threads work in parallel with "lengthy" arithmetic based on the GNU Multiprecision library. Computations on a standard suite of test examples, which are widely used for debugging and comparing programs for computing Gröbner bases, showed that the parallelization effect in the case of the "lengthy" arithmetic is considerably higher than that in the case of the modular arithmetic [10].

2. BASIC DEFINITIONS AND NOTATION

We use the following notation:

$\mathbb{X} = \{x_1, \ldots, x_n\}$ is a set of polynomial variables;

$\mathbb{R} = \mathbb{Q}[\mathbb{X}]$ is a ring of polynomials with rational coefficients;

$Id(F)$ is an ideal of the ring $\mathbb{R}$ generated by polynomials $F \subset \mathbb{R}$;

$\mathbb{M}$ is a set of monomials, i.e., the power product of variables from $\mathbb{X}$ with integer nonnegative exponents;

$\deg_i(u)$ is the degree of variable x_i in the monomial $u \in \mathbb{M}$;

0361-7688/05/3102-0073

$\deg(u) = \sum_{i=1}^{n} \deg_i(u)$ is the total degree of the monomial u;

$\succ$ is an admissible order on the monomials such that $x_1 \succ x_2 \succ \ldots \succ x_n$;

$u|v$ is the conventional divisibility relation of the monomial v by the monomial u; if $u|v$ and $\deg(u) < \deg(v)$, i.e., if u is a *proper divisor* of v, we use the notation $v \sqsupset u$;

$\mathrm{lm}(f)$ and $\mathrm{lt}(f)$ are the leading monomial and leading term of the polynomial $f \in \mathbb{R}\backslash\{0\}$, respectively;

$\mathrm{lm}(F)$ is a set of leading monomials of the polynomial set $F \subset \mathbb{R}\backslash\{0\}$;

$\mathrm{card}(F)$ is the cardinality of the finite set $F \in \mathbb{R}\backslash\varnothing$;

$\mathrm{lcm}(u, v)$ is the least common multiple of the monomials $F \in \mathbb{R}\backslash\varnothing$.

For each $1 \le i \le n$, the finite set of polynomials $F \in \mathbb{R}\backslash\varnothing$ is divided into groups indexed by nonnegative integers $d_1, \ldots, d_i$:

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

A variable x_1 is *J-multiplicative* (*Janet-multiplicative*) [13] for a polynomial $f \in F$ if $\deg_1(\mathrm{lm}(f)) = \max\{\deg_1(\mathrm{lm}(g)) | g \in F\}$. For $i > 1$, x_i is *J-multiplicative* for $f \in [d_1, \ldots, d_{i-1}]$ if $\deg_i(\mathrm{lm}(f)) = \max\{\deg_i(\mathrm{lm}(g)) | g \in [d_1, \ldots, d_{i-1}]\}$.

In what follows, we use the notation $M_J(f, F) \subset \mathbb{X}$ and $NM_J(f, F) = \mathbb{X}\backslash M_J(f, F)$ for the sets of *J*-multiplicative and *J*-nonmultiplicative variables, respectively, for $f \in F$.

Such a separation of variables into nonmultiplicative and multiplicative ones generates involutive division of monomials [6, 14]. This is the so-called Janet division. It is defined by a given finite set of polynomials F and the following monomial order $\succ$: if monomials $u \in \mathrm{lm}(F)$, v, and w are related by the equation $w = uv$ and v contains only Janet-multiplicative variables for u, then u is a *Janet divisor* (or *J-divisor*) of w. In this case, the relation of involutive Janet-divisibility is written as $u \mid_J w$.

A finite set F of nonzero polynomials is *J-autoreduced* (or *Janet-autoreduced*) if, ®$f \in F$, each monomial from f does not have *J*-divisors among $\mathrm{lm}(F)\backslash\{\mathrm{lm}(f)\}$. The *J-normal form* $NF_J(p, F)$ of a polynomial $p \notin F$ with respect to a *J*-autoreduced set F is defined as

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

where $\alpha_{ij} \in \mathbb{K}$, $g_j \in F$, $m_{ij} \in J(\mathrm{lm}(g_j), \mathrm{lm}(F))$, $\mathrm{lm}(m_{ij} g_j) \preceq \mathrm{lm}(p)$, and $\tilde{p}$ does not contain monomials that have Janet divisors among $\mathrm{lm}(F)$.

The subtraction of the terms under the summation sign in the expression for $NF_J(p, F)$ from the polynomial p is called the *multiplicative reduction* of p by the polynomials from F. If $\mathrm{lm}(f)$ ($f \notin F$) does not have *J*-divisors among the elements $\mathrm{lm}(F)$, the polynomial f is said to be in the *J-head normal form* with respect to F, which is written as $f = HNF_J(f, \boldsymbol{F})$.

For a given ideal $I \subset \mathbb{R}$ and the order $\succ$ on the monomials, a finite *J*-autoreduced set of nonzero polynomials $G \subset \mathbb{R}$ generating I is its *Janet basis* if the following condition [6] holds:

$$(\forall f \in G)\ (\forall (x_i \in NM_J(f, G))[NF_J(x_i \cdot f, G) = 0].$$

The product $x_i \cdot f$ of a polynomial $f \in F$ and $x_i \in NM_J(f, F)$ is called a *nonmultiplicative prolongation* of f. Thus, any nonmultiplicative prolongation of the Janet basis is multiplicatively reduced to zero.

A Janet basis is a Gröbner basis, although it is not necessarily autoreduced in the sense of the Gröbner reductions [6]. Like a reduced Gröbner basis, the minimal monic Janet basis is uniquely defined by the ideal and the order on the monomials [7].

3. PARALLEL ALGORITHM

In this section, we describe an improved parallel version of the involutive algorithm for computing polynomial Janet bases suggested in [10].

Like in [10, 11], we make each polynomial $f \in F \subset \mathbb{R}$ correspond to a structure consisting of three elements, $p = \{f, u, vars\}$, where

pol(p) = f is the polynomial f itself,

anc(p) = u is the leading monomial of the ancestor of the polynomial $f \in F$, and

nmp(p) = *vars* is a subset (possibly, empty) of variables from $\mathbb{X}$.

The *ancestor* of the polynomial f is a polynomial $g \in F$ that has the least $\deg(\mathrm{lm}(g))$ among the polynomials from F and satisfies the divisibility condition $\mathrm{lm}(g) \mid \mathrm{lm}(p)$. If the set q of triple elements contains the ancestor, then $\mathrm{anc}(g) = \mathrm{lm}(g)$. If an intermediate polynomial f appearing in the course of the execution of the algorithm to be described below has its *proper* ancestor in the system (i.e., it is not an ancestor to itself), this implies that f has been obtained from g by way of nonmultiplicative prolongations with involutively irreducible leading monomials.

The set *vars* contains all those nonmultiplicative variables of the polynomial in p that have already been used in the course of the execution of the algorithm for constructing nonmultiplicative prolongations.

The second and third elements of the introduced triple structure serve for storing the history of the Janet basis construction with the aim of applying involutive criteria for elimination of useless reductions and repeated prolongations.

The main algorithm **ParallelInvolutiveBasis** uses the multithread model of parallel programming. For a

Input: a finite set $F \in \mathbb{R}\backslash\{0\}$, an admissible order $\prec$ on monomials, and the maximum number $K_{thr} \geq 1$ of threads.
Output: minimal Janet basis G for the ideal $Id(F)$

1: **choose** $f \in F$ with minimal $\mathrm{lm}(f)$ with respect to $\succ$
2: $T := \{f, \mathrm{lm}(f), \varnothing\}$
3: $Q := \{\{g, \mathrm{lm}(g), \varnothing\} \mid g \in F\backslash\{f\}\} \cup \{\{f \cdot x, \mathrm{lm}(f) \cdot x, \varnothing\} \mid x \in NM_L(f, \{f\})\}$
4: **sort** Q in ascending order of polynomials with respect to $\succ$
5: $S := \varnothing$
6: $P := \{q_i \in Q \mid i \leq K_{thr}\}$
7: $Q := Q\backslash P$
8: **create** $\min\{\mathrm{card}(P), K_{thr}\}$ threads **WorkerThread**(P, T, S, *mutex*)
9: **wait** termination of all threads
10: $Q := Q \cup S$
11: **while** $Q \neq \varnothing$ **do**
12: **choose** $p \in Q$ such that $(\nexists q \in Q\backslash\{p\})$ $[\mathrm{lm}(\mathrm{pol}(p)) \sqsupset \mathrm{lm}(\mathrm{pol}(q))]$
13: **if** $\mathrm{lm}(\mathrm{pol}(p)) = 1$ **then**
14: **return** $\{1\}$
15: **else**
16: $Q := Q\backslash\{p\}$
17: **if** $\mathrm{lm}(\mathrm{pol}(p)) = \mathrm{anc}(p)$ **then**
18: **for all** $\{r \in T \mid \mathrm{lm}(\mathrm{pol}(r)) \sqsupset \mathrm{lm}(\mathrm{pol}(p))\}$ **do**
19: $Q := Q \cup \{r\}$; $T := T\backslash\{r\}$
20: **od**
21: **fi**
22: $\mathrm{pol}(p) := \mathbf{NF_J}(\mathrm{pol}(p), T)$
23: **fi**
24: $T := T \cup \{p\}$
25: **for all** $q \in T$ and $x \in NM_J(\mathrm{pol}(q), T)/\mathrm{nmp}(q)$ **do**
26: $Q := Q \cup \{\{\mathrm{pol}(q) \cdot x, \mathrm{anc}(q), \varnothing\}\}$
27: $\mathrm{nmp}(q) := \mathrm{nmp}(q) \cap NM_J(\mathrm{pol}(q), T) \cup \{x\}$
28: **od**
29: $S := \varnothing$
30: **sort** Q in ascending order of polynomials with respect to $\succ$
31: $P := \{q_i \in Q \mid i \leq K_{thr}\}$
32: $Q := Q\backslash P$
33: **create** $\min\{\mathrm{card}(P), K_{thr}\}$ threads **WorkerThread**(P, T, S, *mutex*)
34: **wait** termination of all threads
35: $Q := Q \cup S$
36: **od**
37: **return** $G := \{\mathrm{pol}(f) \mid f \in T\}$

Fig. 1. Algorithm **ParallelJanetBasis**(F, $\prec$, K_{thr}).

given order $\succ$ on the monomials and a fixed maximum number K_{thr} of the threads used, this algorithm computes the minimal Janet basis [7] of the ideal generated by an input set F of nonzero polynomials (Fig. 1). On steps 25 and 27 of the algorithm (and in what follows), to shorten the notation, we use the following convention: a set of triple structures T used for the second argument in NM_J, NF_J, and HNF_{Jb} is meant to be a set of polynomials contained in T. Sometimes, instead of a polynomial, we use the triple structure containing it.

Our previous version of the parallel algorithm [10] used the so-called “thread–pool” model, where all cre-

ated threads wait for input information (a polynomial) for processing (reducing the leading monomial). Note that, in this version, all threads are in the working state during all time of the program execution and, thus, consume resources even if they do not perform their immediate work on the reduction of the leading monomials.

In this paper, we use a different multithread model of parallel programming, the so-called "boss–workers" model. The main program ("boss") creates working threads only when there is a need in them (steps 8 and 33 of the main algorithm) and, then, turns to a sleeping mode until all threads created stop. The advantage of this model over the previous one consists in the simplification of the program implementation, avoiding deadlocks, and reducing mutex delays.

The structure of the **ParallelJanetBasis** algorithm is similar to that of the algorithm from [10]. The multithread parallelization is implemented at steps 8 and 33, when the working threads are created. The number of threads created cannot exceed the maximum number of threads K_{thr}, which is specified at the input of the algorithm, but can be less than this number if K_{thr} is greater than the number of the elements of the set P being processed. Below, we describe the **WorkerThread** and $\mathbf{NF_J}(\mathrm{pol}(p), T)$ subalgorithms, which are invoked at steps 8, 22, and 33. They perform the head and tail J-reductions, respectively.

The sorting of elements of the set Q at steps 4 and 30 is an auxiliary operation. It is especially convenient when the input monomial order is degree compatible. In the program implementation of this algorithm, we use the following two orderings: first, the total-degree ordering and, then, (1) lexicographical or (2) inverse lexicographical ordering. In this case, when selecting an element $p \in Q$ at step 12, it is sufficient to take an element with the least $\deg(\mathrm{lm}(\mathrm{pol}(p)))$. Note that the sorting of Q is performed with respect to the leading monomials of the polynomials contained in it.

The basic difference of the **ParallelJanetBasis** algorithm from the algorithms from [7, 10, 11] is that the condition

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

used for moving elements from T to Q is replaced by the condition

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

checked at step 18. The correctness of using the latter condition for constructing minimal involutive bases not only in the case of the Janet division but also for any other continuous and constructive involutive division [6] will be proved in a future paper. Clearly, the use of this condition instead of the former condition implies an optimization of the algorithm, since this considerably reduces the number of the element transfers from T to Q in the course of the construction of the involutive basis.

Now, let us consider the **WorkerThread** subalgorithm, which is invoked at steps 8 and 33 of the main algorithm and ensures the parallelization. This algorithm controls the working threads that process the sets P, T, and S consisting of the triple elements that were created at steps 2, 5, and 6 of the main algorithm and are available for any working thread.

The fourth input element of the working thread algorithm is a mutex (**mut**ex stands for mutual **ex**clusion object), a system object that prevents simultaneous access of different threads to common (global) resources and, in particular, elements of the sets P, T, and S. A mutex is created when launching the program. When a thread accesses common resources, it must close the mutex for other threads while this thread uses these resources. When the thread does not need the common resources any more, the mutex is to be turned to an open state (for other threads).

The **WorkerThread** algorithm presented in Fig. 2 performs J-reduction of the leading monomial of the polynomial $p \in P$ with respect to the polynomials contained in the set T and, for this purpose, invokes the $\mathbf{HNF_J}(p, T)$ subalgorithm. If the result of the reduction is nonzero, the corresponding triple element is added to S. In this case, if the leading monomial is reducible, the relationship of the polynomial with its ancestor is lost, and the polynomial becomes its own ancestor. Otherwise, the second and third elements of P are not modified by the working thread algorithm.

Clearly, the correctness and termination of the **WorkerThread** algorithm are completely determined by the $\mathbf{HNF_J}(p, T)$ algorithm the correctness and termination of which (as well as of the $\mathbf{NF_J}$ algorithm) follow from the results of works [10, 11].

In the $\mathbf{HNF_J}(p, T)$ algorithm presented in Fig. 3, for a polynomial $\mathrm{pol}(f)$ with the Janet-reducible leading monomial (which is established at step 7), the following two criteria are checked at step 8:

CriterionI(f, g) is satisfied if and only if $\mathrm{anc}(f) \cdot \mathrm{anc}(g) \mid \mathrm{lm}(\mathrm{pol}(f))$.

CriterionII(f, g) is satisfied if and only if $\mathrm{lm}(f) \sqsupset \mathrm{lcm}(\mathrm{anc}(f) \cdot \mathrm{anc}(g))$.

These criteria follow from the Buchberger criteria [4] adapted to the involutive algorithms. If at least one of these criteria is satisfied, we have $HNF_J(\mathrm{pol}(f), T) = 0$ [11].

The last subalgorithm $\mathbf{NF_J}$ (Fig. 4) computes a complete J-normal form performing the Janet reduction of the "tail parts" of the polynomials; it is invoked at step 22 of the main algorithm **ParallelJanetBasis**.

4. PROGRAM IMPLEMENTATION AND TESTING

To study practical efficiency of the **ParallelJanetBasis** algorithm, it was implemented as a separate program module in C. This program is based on the implementation of the sequential algorithm for computing polynomial Janet bases, which was described in [11]. Like in [11], the binary Janet tree was used for the data structures representing T, and unsorted lists were used for representing Q, P, and S.

Input: sets *P*, *S*, and *T* of three elements
Output: a triple set *S* the polynomials of
which have leading monomials that are Janet-reducible
with respect to the polynomials contained in *T*

```
 1: while P ≠ ∅ do
 2:    lock(mutex)
 3:    choose p ∈ P
 4:    P := P\{p}
 5:    unlock(mutex)
 6:    h := HNF_J(p, T)
 7:    if h ≠ 0 then
 8:       lock(mutex)
 9:       if lm(pol(p)) ≠ lm(h) then
10:          S := S ∪ {h, lm(h), ∅}
11:       else
12:          S := S ∪ {p}
13:       fi
14:       unlock(mutex)
15:    fi
16: od
17: return S
```

Fig. 2. Algorithm **WorkerThread**(*P, T, S, mutex*).

Input: a triple structure $f = \{\mathrm{pol}(f), \mathrm{nmp}(f)\}$ and a set *T* of triples
Output: Janet head normal form $h = HNF_J(\mathrm{pol}(f), T)$ of the polynomial
from *f* with respect to the polynomials from *T*

```
 1: G := {pol(g) | g ∈ T}
 2: if lm(pol(f)) is Janet-irreducible with respect to the polynomials from G then
 3:    return f
 4: else
 5:    h := pol(f)
 6:    choose g ∈ T such that lm(pol(g)) |_J lm(h)
 7:    if lm(h) ≠ anc(f) then
 8:       if CriterionI(f, g) or CriterionII(f, g) then
 9:          return 0
10:       fi
11:    else
12:       while h ≠ 0 and lm(h) is Janet reducible by the polynomials from G do
13:          choose q ∈ G such that lm(q) |_J lm(h)
14:          h := h – q · lt(h)/lt(q)
15:       od
16:    fi
17: fi
18: return h
```

Fig. 3. Algorithm $\mathbf{HNF_J}(f, T)$.

Input: a polynomial f from a triple p such that $f := HNF_J(p, T)$ and a set T of triples
Output: complete Janet normal form $h = NF_J(f, T)$ for f with respect to the polynomials from T
1: $G := \{\mathrm{pol}(g) \mid g \in T\}$
2: $h := f$
3: **while** $h \neq 0$ **and** h contains a monomial t that is J-reducible by G **do**
4: **choose** $g \in G$ such that $\mathrm{lm}(g) \mid_J t$
5: $h := h - g \cdot t/\mathrm{lt}(g)$
6: **od**
7: **return** h

Fig. 4. Algorithm $\mathbf{NF}_J(f, T)$.

The execution times were measured when comparing monomials, first, in terms of the total degree and, then, in terms of the inverse lexicographical order compatible with $x_1 > x_2 > \ldots > x_n$ (see Section 2). The computations were carried out on a two-processor PIII-700 MHz, 1Gb RAM, under Gentoo Linux [15]. To perform operations on integers, the GNU Multiprecision Library [12] was used.

In the program implementing the sequential algorithm [11], the `dimalloc` memory manager was used. This manager, however, as well as the `GC` manager, turned out unacceptable for work with several threads. Therefore, for the memory manager, we used the standard `malloc` manager. This resulted in certain computation slowdown (see table) in the one-thread mode as compared to the parallel code [11], which is explained by the nonoptimality of the memory access and additional expenditures caused by the necessity of synchronizing several threads.

Processing times for the parallel program in the one-thread and three-thread modes[1] are presented in the table. Like in our previous papers [10, 11], for test examples, we used systems of polynomial equations taken from the databases [16, 17]. The examples from these databases are considered to be standard ones for debugging and comparing various programs for computing Gröbner bases.

The execution times were measured as follows. First, the operation times of separate threads working in parallel, which are invoked at steps 8 and 33 of the "boss" algorithm **ParallelJanetBasis**, were measured. The timer of the main thread ("boss") was stopped until all the created working threads terminated their operations. After all reductions have been completed, the maximum of the individual times of the operation of the working threads was added to the record of the main timer.

This scheme of the determination of the computation time makes it possible not to take into account the time spent on swapping and delays caused by accessing to the disk memory, as well as other delays insignificant from the standpoint of the time required for the computation itself.

The second column of the table shows the computation times for our, earlier created and optimized, sequential program. The third column presents the processing times for the multithread version with one working thread. Taking into account that the computations were carried out on a two-processor computer, the scalability can experimentally be observed if we compare the times in the case of one thread with those in the case of three threads, which are shown in the fourth column.

The last two columns show the ratios of the computation times of the sequential program and multithread program with one thread, respectively, to the computation times of the multithread program with three threads. In the majority of cases, the three-thread version is the fastest one, even in terms of the total (astronomic) time.

In addition to measuring computation times, the following empirical analysis was done: in the multithread version, we varied the number of the working threads and measured the processor load. The load turned out maximal in the case of three working threads. This observation completely agrees with the known rule that the number of the working threads should be equal to the number of the processors plus one.

It should be noted that, in some tests, the computation speedup with the increase of the number of the working threads was even superlinear. This phenomenon can be explained by the fact that, in the case of a multithread computation, there may arise situations when simpler (in terms of the number of the monomials and coefficient lengths) polynomials are placed into T earlier than they would be placed in the case of the sequential computation. Then, these polynomials invoke other, faster, reduction chains for nonmultiplicative prolongations contained in Q and, thus, speed up the computation. This can be stated in a different way: an experimentally obtained strategy of selecting elements from Q may occur more optimal than that

[1] That is, the modes with one and three working threads performing Janet reduction of the leading monomials.

Test execution times (in seconds) and speedup owing to parallelization

Example	Sequential computation	1 thread	3 threads	Speedup	t_{seg}/t_3	t_1/t_3
assur44	65.50	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
cyclic5	0.36	0.79	1.16	−0.87	0.31	0.68
cyclic7	193.6	386.89	182.86	+10.74	1.06	2.12
dl	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	267.5	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
febrice24	649.14	648.78	254.39	+394.75	2.55	2.55
fulter9	38.73	33.58	38.66	+0.07	1.00	0.88
i1	626.30	100.69	371.79	+254.51	1.69	2.69
jcf26	1317.05	1325.57	515.61	+801.44	2.55	2.57
karsura7	5.87	11.28	5.85	+0.02	1.00	1.93
karsura8	71.16	119.92	53.72	+17.44	1.33	2.23
karsura9	911.09	1356.37	587.82	+323.27	1.55	2.31
karsura10	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
kotsiteas	25.19	29.29	13.76	+11.43	1.83	2.13
noon6	3.01	4.56	3.88	−0.87	0.78	2.18
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	12.29	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

used in the sequential algorithm, which was heuristically optimized just for this algorithm.

5. CONCLUSIONS

The last step of the **ParallelJanetBasis** algorithm may contain the command

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

In this case, the algorithm outputs a reduced Gröbner basis. This immediately follows from the definition of the ancestor given in the beginning of Section 3. The extraction of all polynomials that have no ancestors (i.e., they are ancestors of themselves and, thus, form a reduced Gröbner basis) does not require additional reductions. Thus, the suggested parallel algorithm can be used for computing reduced Gröbner bases.[2] The

program has a special option by specifying which the user can output either the minimal Janet basis, or the reduced Gröbner basis, or both.

The first results of the computation of the standard suite of test examples from [16, 17] show high efficiency of the parallelization in the involutive algorithm. Further, we plan to reduce parallelization overheads and to study possibilities of using distributed computation models and the appropriate existing software for computing involutive bases.

It should be noted that the two criteria used in the $\mathbf{HNF}_J(p, T)$ subalgorithm do not completely cover the Buchberger criteria, which was demonstrated in [18]. Some zero reductions can be found by means of two additional criteria formulated in [18]. However, our first attempts to build in the lacking criteria into the existing program for computing Janet bases [11] did not result in a considerable gain in the computation rate, at least, on the standard examples from the databases [16, 17]. Moreover, for the majority of the examples, the incorporation of the new criteria even slowed down the computation. A possible explanation of this slowdown is that the data structures used in [11] are not suitable in terms of the efficient use of the new criteria, which, in practice, work much fewer than the basic criteria included in the $\mathbf{HNF}_J(p, T)$ algorithm. For this reason, we did not include the new criteria in the parallel version of the algorithm and continue to study practical advisability of using them for computing involutive bases.

ACKNOWLEDGMENT

This work was supported by the Russian Foundation for Basic Research, project no. 04-01-00784, and by the Ministry of Education of the Russian Federation, project no. NSh-2239.2003.2.

One of the authors (V.P. Gerdt) is grateful to R. Hemmecke for helpful discussions of involutive criteria.

REFERENCES

1. Faugère, J.C., Parallelization of Gröbner Basis, in *Lecture Notes. Series in Computing 5* (PASCO'94), Singapore: World Sci., 1994, pp. 124–132.
2. Attardi, G. and Traverso, C., Strategy-Accurate Parallel Buchberger Algorithm, *J. Symb. Comp.*, 1996, vol. 5, pp. 411–425.
3. Amrheim, B., Gloor O., and Küchlin, W., A Case Study of Multi-Threaded Gröbner Basis Completion, *Proc. of ISSAC'96*, 1996, pp. 95–102.
4. Buchberger, B., Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, *Recent Trends in Multidimensional System Theory*, Bose, N.K., Ed., Dordrecht: Reidel, 1985, pp. 184–232. Translated under the title *Komp'yuternaya algebra. Simvol'nye i algebraicheskie vychisleniya*, Moscow: Mir, 1986.
5. Gianni, P., Mora, T., Niesi, G., Robbiano, L., and Traverso, K., "One Sugar Cube, Please" or Selection Strategies in Buchberger Algorithm, *Proc. of ISSA'91*, ACM, 1991, pp. 49–54.
6. Gerdt, V.P. and Blinkov, Yu.A., Involutive Bases of Polynomial Ideals, *Math. Comput. Simulation*, 1998, vol. 45, pp. 519–542.
7. Gerdt, V.P. and Blinkov, Yu.A., Minimal Involutive Bases, *Math. Comput. Simulation*, 1998, vol. 45, pp. 543–560.
8. Gerdt, V.P., On an Algorithmic Optimization in Computation of Involutive Bases, *Programmirovanie*, 2002, no. 2, pp. 62–65.
9. Yanovich, D.A., Parallelization of an Algorithm for Computation of Involutive Janet Bases, *Programmirovanie*, 2002, no. 2, pp. 66–69.
10. Gerdt, V.P. and Yanovich, D.A., Parallelism in Computing Janet Bases, *Proc. of the Workshop on Under- and Overdetermined Systems of Algebraic or Differential Equations*, Calmet, J., Hausdorf, M., and Seiler, W.M., Eds., Karlsruhe: Institute of Algorithms and Cognitive Systems, Univ. of Karlsruhe, 2002, pp. 47–56.
11. Gerdt, V.P., Blinkov, Yu.A., and Yanovich, D.A., Construction of Janet Bases II. Polynomial Bases, *Proc. of the Conf. "Computer Algebra in Scientific Computing" (CASC'01)*, (2001), Berlin: Springer, 2001.
12. http://www.swox.com/gmp.
13. Janet, M., *Leçons sur les Systèmes d'Equations aux Dérivées Partielles*, Cahiers Scientifiques, IV, Paris: Gauthier-Villars, 1929.
14. Gerdt, V.P. and Blinkov, Yu.A., Involutive Divisions of Monomials, *Programmirovanie*, 1998, no. 6, pp. 22–24.
15. http://www.gentoo.org.
16. Bini, D. and Mourrain, B., Polynomial Test Suite, 1996, http://www-sop.inria.fr/saga/POL.
17. Verschelde, J., The Database with Test Examples, http://www.math.uic.edu/~jan/demo.html.
18. Apel, J. and Hemmecke, R., Detecting Unnecessary Reductions in an Involutive Basis Computation, *RISC Linz Report π 02-22*, 2002.
19. Calmet, J., Hausdorf, M., and Seiler, W.M., A Constructive Introduction to Involution, *Proc. Int. Symp. Applications of Computer Algebra – ISACA 2000*, Akerkar, R., Ed., New Delhi: Allied Publishers, 2001, pp. 33–50.

[2] Note that the sequential algorithm from [11] also can be used for this purpose.

Spell: Ok