An Estimation of the Parallelization Quality of the Involutive Basis Computation Algorithm[⋆]

V.A. Mityunin and A.S. Semenov

Moscow State University, Department of Mechanics and Mathematics, Laboratory of Computing Methods,
Vorobyovy Gory, Moscow 119899, Russia
vmit@metric.ru semyonov@mccme.ru

Abstract. We present an attempt to estimate the quality of the parallelization of the minimal involutive bases construction algorithm. The problem of the possibility to parallelize the algorithm is very important because of a big computational complexity. The estimation of the quality of the parallelization of the minimal involutive bases construction algorithm, presented in this paper, was never published before. It is shown that the minimal involutive basis algorithm is very sequential like the Gröbner basis construction algorithm.

1 Introduction

In this work we present an attempt to estimate the quality of the parallelization of the minimal involutive bases construction algorithm. The task of the classical Gröbner basis computation is very sequential as Faugère has shown in [2], and, therefore, every attempt to parallelize it essentially breaks the original algorithm. The main reason for this is that the result of the polynomial reduction often depends on other polynomials, in particular, on the last reduced polynomial. Nevertheless, modifying the reduction strategy we can achieve some acceleration. The estimation of the quality of the parallelization of the minimal involutive bases construction algorithm presented in this work, was never published before.

Here we present an estimation done by the method proposed by Faugère. The main idea of this method is that if the polynomial a is to be reduced with respect to a_1, and b with respect to b_1, and a_1 and b_1 are already computed, we can perform these two reductions simultaneously. This algorithm collects all such cases and estimates the quality of the parallelization.

2 Basic Minimal Involutive Basis Algorithm

In paper [2] a general approach to the parallelization of the Buchberger-like algorithms was sketched, and the parallelization of the classical Buchberger algorithm was presented. In our paper we use this approach to the minimal involutive basis construction algorithm.

Below you can see the version of the minimal involutive basis construction algorithm presented in [6]. We used this interpretation to develop a parallel and the probabilistic versions of the algorithm.

Here NF_L denotes the involutive normal form, NM_L the set of non-multiplicative variables, and $Criterion$ is the involutive modification of the standard Buchberger criterion.

$Criterion(g, u, T)$ is true provided that if there is $(f, v, D) \in T$ such that
$lm(f)|_L lm(g)$ and $lcm(u, v) \prec lm(g)$.

In the final basis some polynomials are initial polynomials and others are reduced non-multiplicative prolongations of earlier computed ones.

A polynomial can migrate many times between T and Q, so its reductions sometimes are performed in several portions (not all in one time, as in the Buchberger algorithm without interreductions).

[⋆] This work was supported by INTAS(project No. 99-1222) and RFBR grant 02-01-01033

```
Algorithm **Minimal Involutive Basis**
**Input:** $F$, a finite polynomial set
**Output:** $G$, the minimai involutive basis of the ideal $Id(F)$
**begin**
  $F := Autoreduce(F)$
  **choose** $g \in F$ with the lowest $lm(g)$ w.r.t. $\prec$
  $T := \{(g, lm(g), 0)\}; Q := 0; G := \{g\}$
  **for each** $f \in F \setminus \{g\}$ **do**
  $Q := Q \cup \{(f, lm(f), 0)\}$
  **repeat**
    $h := 0$
    **while** $Q \neq 0$ **and** $h = 0$ **do**
      **choose** $g$ in $(g, u, P) \in Q$ with the lowest $lm(g)$ w.r.t. $\prec$
      $Q := Q \setminus \{(g, u, P)\}$
      **if** $Criterion(g, u, T)$ is false **then** $h := NF_L(g, G)$
    **end**
    **if** $h \neq 0$ **then** $G := G \cup \{h\}$
      **if** $lm(h) = lm(g)$ **then** $T := T \cup \{(h, u, P)\}$
      **else** $T := T \cup \{(h, lm(h), 0)\}$
         **for each** $f$ in $(f, v, D) \in T$ s.t. $lm(f) \succ lm(h)$ **do**
        $T := T \setminus \{(f, v, D)\}; Q := Q \cup \{(f, v, D)\}; G := G \setminus \{f\}$
    **while** exist $(g, u, P) \in T$ and $x \in NM_L(g, G) \setminus P$ and, if $Q \neq 0$,
    s.t. $lm(gx) \prec lm(f)$ for all $f$ in $(f, v, D) \in Q$ **do**
      **choose** such $(g, u, P)$, $x$ with the lowest $lm(g)x$ w.r.t. $\prec$
      $T := T \setminus \{(g, u, P)\} \bigcup \{(g, u, P \cup \{x\})\}$
      **if** $Criterion(gx, u, T)$ is false **then** $h := NF_L(gx, G)$
        **if** $h \neq 0$ **then** $G := G \cup \{h\}$
          **if** $lm(h) = lm(gx)$ **then** $T := T \cup \{(h, u, 0)\}$
          **else** $T := T \cup \{(h, lm(h), 0)\}$
            **for each** $f$ in $(f, v, D) \in T$ with $lm(f) \succ lm(h)$ **do**
            $T := T \setminus \{(f, v, D)\}; Q := Q \cup \{(f, v, D)\}; G := G \setminus \{f\}$
    **end**
  **until** $Q \neq 0$
**end**
```

For the parallelization analysis we use a *flow-protocol* of the algorithm which reflects the history of additions and reductions of the polynomials and their migrations from T to Q and back. This idea is due to Faugère and was described in [2].

Each polynomial has a number, which reflects the order of its first appearance. The flow-protocol consists of *traces* which have the form

$$[Number, OperationCode, \ldots]$$

As we have four basic polynomial operations in the algorithm, the second parameter can admit one of four indicators:

– *Introduction of an initial polynomial and reducing it (0)*
– *Introduction of a prolongation and reducing it (1)*
– *Removing a polynomial from T to Q (2)*
– *Reintroducing a polynomial from Q and reducing it (4)*

In every trace except for the third type the list of performed involutive reductions is also kept. (In the second case this list is headed by the parent polynomial, as we assume that a prolongation takes approximately the same time as one reduction.) Every polynomial is represented in a reduction list by its number.

For example, if a polynomial u is a prolongation of the polynomial v and it was reduced with respect to polynomials $v_1, ...v_n$, then the trace has the form

$$[u, 1, v, v_1, \ldots, v_n]$$

If then it was sent back to Q, and reintroduced to T after reductions with respect to $w_1, \dots, w_m$ two traces are to be added to flow-protocol

$$[u, 2]$$

and

$$[u, 4, w_1, \dots, w_m]$$

The traces are added into the flow-protocol following the order of their generation in the algorithm. For example, if polynomial a was proceeded with the trace A, then polynomial b was proceeded with the trace B and then again a was proceeded with the trace C, the flow-protocol will contain the sequence A,B,C of traces.

We call the *history* of a polynomial the list of all **its** traces in their order. In our example, the history of the polynomial a has the form $\{\dots, A, C, \dots\}$.

3 The Principles of Faugère's Estimation

We can see that the flow-protocol gives us almost complete information about the algorithm proceeding (and complete, if the method of involutive reduction is given explicitly).

The reduction list formed by all reductions of polynomial u (it is obtained when we merge all reduction lists in the history of u) is called *the aggregate list.* If, in our example, no more reductions are done to u during the whole algorithm, the aggregate list is

$$[v, v_1, \dots, v_n, w_1, \dots, w_m]$$

But for our model it will be more useful to keep in memory many small lists instead of one bigger, as the order, in which different polynomials are reduced, plays a significant role. For example, if we have such an order of traces

$$[u, 2] \dots [w_1, 1, r_1 \dots, r_k] \dots [u, 4, w_1, \dots, w_m] \dots [w_1, 2] \dots [w_1, 4, q_1, \dots, q_z]$$

the polynomial u is reduced with respect to w_1 in the algorithm although w_1 is not completely computed by that time.

4 Estimation Model

Consider the aggregate list of the polynomial number n. Denote its length by v. Denote by j the number of elements in the list which go before the first occurrence of $(n-1)$-th polynomial. Faugère uses the coefficient $p = j/n$ of a polynomial to measure its independence [2]. If $p = 1$ the polynomial is called *independent*, and if $p > 0.66$ the polynomial is called *almost independent.* The large amount of (almost) independent polynomials can be regarded as the indicator of good parallelization quality.

In our rough model we suppose that all the reductions take the same time. Of course, it is not true, but if the initial ideal is not extremely complicated, this approximation seems to be sufficient.

Let us have N polynomials in flow-protocol, each of them is denoted by p_i. The coefficient v_i is the length of the aggregate list of p_i. The sequential time T_{seq} is the number of all the involutive reductions performed (the prolongation operation itself is considered to be a reduction) and it equals $\sum_{i=1}^{N} v_i$.

Each polynomial is represented in flow-protocol by its history, which can be obtained as the union of all the traces.

In our estimation algorithm we don't deal with polynomials themselves. The *reduction* operation of a polynomial corresponds to such transformation of history list: (We assume that in a history list all traces with empty reduction lists are removed): we take the first trace and delete the first polynomial number from it. If this trace becomes empty, we erase it and make the next trace to be the first one.

For example, if polynomial u was once reduced by polynomials $[v_1, \dots, v_n]$, then by $[w_1, \dots, w_m]$ its history is

$$[v_1, \dots, v_n][w_1, \dots, w_m]$$

After the first reductions it is transformed into

$$[v_2, \dots, v_n][w_1, \dots, w_m]$$

After the reduction by v_n, the history has the form

$$[w_1, \ldots, w_m]$$

The imitation of sequential version of Minimial Involutive Basis Algorithm can be done as follows: let t be the variable which indicates the time spent (at the beginning it is zero). We take traces according to the order of their appearance. Having taken a trace we delete its members ("reduce it"), increasing the variable t. Having finished the procedure, we have $t = T_{seq}$.

The idea of parallel estimation is the following one.

Let u and v be polynomials, and they are to be reduced at current step by polynomials u_1 and v_1, respectively, where u_1, v_1 are already computed. Then we can reduce u and v simultaneously and in estimation consider that it happens in one time cycle (causing only one increase of t).

How can we determine whether a polynomial u can be reduced with respect to a polynomial u_1? In the case when a polynomial is completely reduced at one time (has only one trace with reduction list in history) the answer is clear: the history of u_1 should be empty. But the Minimal Involutive Basis algorithm is not of that kind. The reduction of one particular polynomial can be performed during many times, interrupted by reductions of other polynomials. Let us consider the polynomial u which is to be reduced with respect to v. Both polynomials are not completely reduced. The first traces in their histories of u and v are T_u and T_v respectively. If T_v precedes T_u in flow-protocol that means that in the initial algorithm u was reduced with respect to v after v was reduced with the trace T_v. This means that we cannot reduce u in our simulation before the trace T_v is empty. On the contrary, we can reduce, when the history of v is empty or T_u precedes T_v in flow-protocol. The function *isReduced* surveys the histories of polynomials and answers whether the reduction can be performed in a sense described above.

Below we outline the estimation of parallelization quality. We can assume that the modular trace of the computations is very similar to the integer trace (in fact this is true with a very high probability), and use traces obtained in the modular computations. It is, however, also possible to use real integer traces. We used modular traces in our estimations and compared the results of the estimations with the real timings. The results were very similar, so the quality of the estimation was good. Having computed the trace of histories, we can use the simple procedure presented below to estimate the parallelization quality of the algorithm.

```
Algorithm **Estimation of the Parallelization Quality**
$MaximalNumberOfProcessors = 0$
$T_{par} = 0$
**while** ( not all histories are empty )
  $NumberAtCurrentStep = 0$
  **for**( $c = 1$; $c <= N$; $c++$)
    **if**(polynomial number $c$ can be reduced)
      **reduce**($c$)
      $NumberAtCurrentStep{+}{+}$
  $T_{par}{+}{+}$
  **if**( $NumberAtCurrentStep > MaximalNumberOfProcessors$)
    $MaximalNumberOfProcessors = NumberAtCurrentStep$
```

Here the variable T_{par} denotes the approximate parallel time measured in the elementary operations (with the given assumptions). Variables *NumberAtCurrentStep* and *MaximalNumberOfProcessors* denote the number of processor used simultaneously at the current step, and the maximal number of processors used, respectively.

We can also introduce the average number of processors used, which equals T_{seq}/T_{par} (the number of all reductions devised by the parallel time). The meaning of this number is the number of processors we can use efficiently during the computation process. While it is possible to perform computation using as much processors as one can afford, it will not be any speedups using more than this number.

The estimation of the quantity of parallelization are given in the following table.

N_{ind}	the number of independent polynomials
N_{almost}	the number of almost independent polynomials
N_{all}	the number of all polynomials
P_{max}	the maximal number of processors
$P_{average}$	the average number of processors

Name	N_{ind}/N_{all}	N_{almost}/N_{all}	T_{seq}	T_{par}	P_{max}	$P_{average}$
assur44	44.49	49.35	18174	9666	326	1.88
boon	74.39	76.83	621	143	68	4.34
butcher	67.79	74.50	1482	460	126	3.22
butcher8	76.97	78.69	12404	5213	448	2.38
camera1s	85.00	86.67	857	302	54	2.84
caprasse4	71.43	71.43	686	190	48	3.61
cassou	38.89	40.74	1339	669	31	2.00
chandra6	61.67	63.33	1011	614	57	1.65
cohn2	76.25	77.50	1487	825	70	1.80
comb3000	57.89	59.65	440	175	39	2.51
conform1	60.00	60.00	2	1	0	2.00
cpdm5	23.08	23.08	6974	5259	86	1.33
cyclic2	100.00	100.00	0	1	0	0.00
cyclic3	75.00	75.00	4	4	1	1.00
cyclic4	71.43	71.43	26	13	3	2.00
cyclic5	38.89	55.56	658	312	31	2.11
cyclic6	21.00	42.00	5995	3101	98	1.93
cyclic7	23.84	47.45	285693	120776	877	2.37
d1	66.41	68.37	8439	2156	567	3.91
des18-3	45.93	45.93	1933	1016	57	1.90
des22-24	43.93	43.93	2177	1385	80	1.57
discret3	81.33	81.78	37703	15261	665	2.47
dl	61.37	61.80	142040	26533	3788	5.35
eco10	47.24	48.10	80772	57013	792	1.42
eco11	53.76	54.66	269387	169729	1409	1.59
eco5	58.82	64.71	135	83	14	1.63
eco6	50.00	55.88	453	296	27	1.53
extcyc5	45.87	47.71	27258	19809	104	1.38
eco7	46.73	50.47	1730	973	72	1.78
eco8	37.71	42.29	5865	3893	130	1.51
eco9	50.47	52.58	19428	14319	315	1.36
extcyc5	45.87	47.71	27258	19809	104	1.38
extcyc6	91.28	91.42	189034	62800	1336	3.01
f744	42.39	42.92	27212	7183	702	3.79

We can see that the parallel acceleration has a modest rate. Not very great amount of polynomials are (almost) independent. The use of parallel model nevertheless seems to be profitable.

Our program complex supports both integer and modular field computations. It is implemented in Microsoft Visual C++ 6.0 and can be compiled on many platforms. Parallelization is supported by means of MPI 1.1 standard.

For the Minimal Involutive Basis algorithm we use the Janet involutive division.

We are grateful to Dr. E.V. Pankratiev and Dr. V.P. Gerdt for remarks and useful discussions.

References

1. Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C.: ”One sugar cube, please” or Selection strategies in Buchberger algorithm. In: *Proc. ISSAC '91*, S.M. Watt (Ed.), ACM Press, New York (1991) 49–54
2. Faugère, J.C.: Parallelization of Gröbner basis. In: *Proc. PASCO'94*, World Scientific Publishing Company, Singapore (1994)
3. Siegl, K.: A parallel factorization tree Gröbner basis algorithm. In: *Proc. PASCO'94*, World Scientific Publishing Company, Singapore (1994)
4. Reeves, A.A.: A parallel implementation of Buchberger's algorithm over Zp for p $\leq$ 31991. *J. Symb. Comp.* **26** (1998) 229–244
5. Gebauer, R., Möller, H.M.: On an installation of Buchberger's algorithm. *J. Symb. Comp.* **6** (1988) 275–286
6. Gerdt, V.P., Blinkov, Yu.A.: Minimal involutive bases. *Math. Comp. Simul.* **45** (1998) 543–560

7. Gerdt, V.P., Blinkov, Yu.A., Yanovich, D.A.: Construction of Janet Bases. In: *Computer Algebra in Scientific Computing/ CASC 2001*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), Springer-Verlag, Berlin (2001) 233–263
8. Mityunin, V.A., Zobnin, A.I., Ovchinnikov, A.I., Semenov, A.S.: Involutive and classical Gröbner bases construction from the computational viewpoint. In: *Proc. CAAP'2001*, V.P. Gerdt (Ed.), JINR, Dubna (2002) 221-230
9. Mikhalev, A.V., Pankrat'ev, E.V.: *Computer Algebra. Calculations in Differential and Difference Algebras*, Moscow State Univ., Moscow (1989) (in Russian)