H. Melenk, H.M. Möller *, W. Neun

On Gröbner Bases Computation on a Supercomputer Using REDUCE

* FB Mathematik und Informatik
der Fernuniversität Hagen
D-5800 Hagen
Federal Republic of Germany

Preprint SC 88-2 (January 1988)

Konrad-Zuse-Zentrum für Informationstechnik;
Heilbronner Straße 10; D-1000 Berlin 31

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Strasse 10
1000 Berlin 31
Verantwortlich: Dr. Klaus André
Umschlagsatz und Druck: Verwaltungsdruckerei Berlin

ISSN 0933-7911

Contents

Abstract

Gröbner bases are the main tool for solving systems of algebraic equations and some other problems in connection with polynomial ideals using Computer Algebra Systems. The procedure for the computation of Gröbner bases in REDUCE 3.3 has been modified in order to solve more complicated algebraic systems of equations by some general improvements and by some tools based on the specific resources of the CRAY X-MP. We present this modification and illustrate it by examples.

1. Gröbner bases in REDUCE 3.3

1.1 Gröbner bases and Buchberger's algorithm

The concept of Gröbner bases, a special kind of bases for polynomial ideals, was introduced by Buchberger in his thesis (1965) and developed in forthcoming papers. Using Gröbner bases, many problems dealing with multivariate polynomials, especially algebraic equations can be solved. For a survey see Buchberger (1985). We assume familiarity with the basic notions of Gröbner bases namely: *term ordering, leading term (lt), leading coefficient (lc), reduction modulo a finite polynomial set, irreducibility modulo such a set,* and *S-polynomial.* For polynomials over a field, we employ the same definition for Gröbner bases as Buchberger, i.e. G is called a Gröbner basis if for any two $f, g \in G$ the S-polynomial $S(f,g)$ reduces modulo G to 0. For polynomials over $\mathbb{Z}$, we define Gröbner bases in the same way but using pseudo-reduction for the S-polynomials as explained in 2.2. This leads to a Gröbner basis notion, which is called in Möller (1988) a weak Gröbner basis.

A Gröbner basis G is reduced if every $f \in G$ is irreducible modulo $G \setminus \{f\}$. In case of polynomials over fields, every ideal possesses exactly one reduced Gröbner basis F, where $lc(f) = 1$ for every $f \in F$, cf. Buchberger (1985). Normalizing every element of a reduced (weak) Gröbner basis F over $\mathbb{Z}$ by dividing it by its lc, we get such uniquely determined Gröbner basis, but now for the ideal generated by F over $\mathbb{Q}$ instead over $\mathbb{Z}$.

Given a finite polynomial set F generating an ideal I, Buchberger's algorithm consists essentially in calculating the S-polynomials, (pseudo-)reducing them modulo F to irreducible, so called H-polynomials, enlarging F by the nonzero H-polynomials, removing from F such polynomials, whose lt is a multiple of an lt of an other polynomial in F, and then continuing by calculating new S-polynomials until no more S-polynomials exist leading to non-zero H-polynomials. At termination, F is a Gröbner basis of I. Buchberger (1985) gave criteria for predicting, that some H-polynomials are 0. This allows to skip some S-polynomial calculations.

1.2 Buchberger's algorithm in REDUCE 3.3

Buchberger's algorithm was implemented in the Computer Algebra System SAC-2, see Böge, Gebauer, and Kredel (1986). It is based on the description of the algorithm given by Buchberger (1985). The implementation of the algorithm in REDUCE 3.3 by Gebauer, Hearn and Kredel is based on a variant, which differs from the original Buchberger algorithm by a more efficient testing of the criteria for omitting S-polynomial calculations and reductions. This variant is described by Gebauer and Möller (1988) and also implemented in the meantime in SCRATCHPAD II.

The REDUCE 3.3 implementation includes an internal package for the manip-

ulation of polynomials in a distributive representation. In this data structure a polynomial is represented as linear sorted sequence of monomials. One advantage of this kind of representation is the possibility to select an ordering of the monomials by ordering the terms, i.e. the power products of the variables $x_1, \ldots, x_n$ in dependence of the desired application. In REDUCE 3.3, the following two orderings are available.
The INV(erse) LEX(icographical) ordering:

$$x^{(a_1,\ldots,a_n)} >_{invlex} x^{(b_1,\ldots,b_n)}$$

$$:\Longleftrightarrow$$

$$\exists i < n : (\forall j < i : a_j = b_j) \wedge a_i > b_i$$

The I(nverse) T(otal) D(e)G(ree) ordering:

$$x^{(a_1,\ldots,a_n)} >_{itdg} x^{(b_1,\ldots,b_n)}$$

$$:\Longleftrightarrow$$

$$(\sum a_i > \sum b_i) \vee (\sum a_i = \sum b_i \wedge x^{(a_1,\ldots,a_n)} >_{ilex} x^{(b_1,\ldots,b_n)})$$

The INVLEX ordering is usefull for the solution of algebraic equations, see example 5.5. These two orderings are the same as in SAC-2, cf. Böge, Gebauer, Kredel (1986), but different e.g. from those in Scratchpad II, cf. Jenks et al (1986), such that reduced Gröbner bases in different systems, even if they are normalized in the same way, can not be compared directly.

The REDUCE standard quotients are used as coefficients for the polynomials. Standard quotients can be rational numbers or rational functions over variables or formal functions (more general: kernels). So the algorithm in REDUCE 3.3 can be applied to polynomials over $\mathbb{Q}$, the field of rational numbers, and to polynomials over fields of rational functions.

1.3 Cray computer usage for Gröbner calculations

In 1986 REDUCE 3 was implemented for Cray 1 and Cray X-MP computers, Melenk and Neun (1987), based on Portable Standard LISP (PSL 3.4), see Anderson et al. (1987) and Melenk and Neun (1986), and with Version 3.3 of REDUCE the Gröbner package became available for these computers too. The objective was to use the combination of REDUCE and the supercomputer for the enlargement of the application range of Gröbner bases calculations towards 'larger' problems, especially for the solution of systems of algebraic equations.

1.4 Introduction of new facilities

It soon became clear that the increased computing power alone was not sufficient to reach our goal. So we started a four level approach:

- refinement of the Buchberger algorithm,
- improvement of the implementation,
- special tuning of basic package parts for the target machine and
- optimization of the underlying LISP.

The first two groups will be discussed in the following sections. They are machine independent (in fact we developed the software using Sun and IBM computers as workstations). The Cray specific parts will be mentioned only briefly.

2. Refinement of Buchberger's algorithm and of its environment

2.1 The coefficient domains

We enlarged the range of applicable coefficient domains and fixed for any specific coefficient domain an adequate normalization. In the field case all polynomials are normalized internally to have a leading coeffient 1, in the ring case (no general division available) the normalization reduces in general the contents of the polynomials to 1.

Normalization of the leading coefficients to 1

$\mathbb{Q}$	rational numbers
$\mathbb{Q}(a_1,\ldots,a_s)$	rational functions in $a_1,\ldots,a_s$ over $\mathbb{Q}$
$\mathbb{Z}_p$	integers modulo a prime number p

Content of polynomial coefficients normalized to 1

$\mathbb{Z}$	integer numbers of arbitrary length
$\mathbb{Z}[a_1,\ldots,a_s]$	polynomials in $a_1,\ldots,a_s$ over $\mathbb{Z}$

Other normalizations

$M(p_1,\ldots,p_s)$	Cartesian product of the prime fields $\mathbb{Z}_{p_i}$,i.e. $M(p_1,\ldots,p_s) = \mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_s}$. Here the leading coefficient is an s-tuple. It is normalized to $(1 \bmod p_1,\ldots,1 \bmod p_s)$, see 2.6.3.
$H(p,f)$	hybrid numbers, i.e. pairs of an integer modulo a given prime p and a floating point number. The leading coefficient is here normalized to a hybrid number of type $(1 \bmod p, x), x$ a floating point number, see 2.6.2.

Experiments have shown, that for Gröbner basis calculations with non modular coefficients the ring version of the algorithm is faster than the field version. Of course, the calculations based on $\mathbb{Z}$ tend to produce bigger coefficients compared to the $\mathbb{Q}$ applications, but this fact is not so important if a fast arithmetic is available, see the statistics in 5.3. So the ring version is the default and input polynomials are converted from $\mathbb{Q}$ and $\mathbb{Q}(a_1,\ldots,a_s)$ to $\mathbb{Z}$ and $\mathbb{Z}[a_1,\ldots,a_s]$

by multiplying every coefficient of a polynomial with the lcm of all coefficient denominators (and using the identity $a/1 = a$ for all $a \in \mathbb{Z}$ or $a \in \mathbb{Z}[a_1, \ldots, a_s]$ resp.).

2.2 Polynomial reductions

A single reduction step modulo a finite set G of non-zero polynomials reduces a polynomial f to a polynomial $h := f - tg$, with $g \in G, t$ a monomial, such that the monomial m in f with $lt(m) = lt(t) \cdot lt(g)$ is canceled by $t \cdot g$, but h contains in addition some more monomials of lower order. A pseudo-reduction step modulo G reduces f to $c \cdot f - t \cdot g$, with $g \in G, t$ a monomial and c a constant coefficient, such that again the monomial m with $lt(m) = lt(t) \cdot lt(g)$ in f is canceled. This pseudo-reduction is installed in addition to the existing reduction and used for polynomials over $\mathbb{Z}$ and $\mathbb{Z}[a_1, \ldots, a_s]$.

For testing the efficiency of reduction procedures, we installed also the so called lt-reduction. This is also a reduction in the sense explained before, but the monomial m in f which is canceled must be maximal: $lt(m) = lt(f)$. This reduction is already sufficient in Buchberger's algorithm for obtaining a Gröbner basis, see for instance Möller (1988). However, as our tests in 5.3 indicate, the lt-reduction is in general slower.

As an optional feature we can start the algorithm by first reducing cyclically the input polynomials with themselves, i.e. reduce each $f \in F$ modulo $\{g \in F | lt(g) < lt(f)\}$ beginning with an f with maximal $lt(f)$ and resulting in a reduced polynomial set F', which can be reduced again in the same manner, etc. Such a specific reduction phase can eliminate redundancies more efficiently than the main algorithm. A similar prereduction takes place, immediately before a Gröbner basis calculation for a subproblem as described in 2.5 is started.

2.3 Extraction of power product factors

When Gröbner bases are needed only for the solving of a set of algebraic equations, multiplicities of zeros may be ignored. Therefore multiple factors of a polynomial may be reduced to simple ones by polynomial division. We installed for the situation, when such multiple factor is a power product, a procedure following a proposal of Davenport (1987). If f has a power product factor,

$$f = x_1^{k_1} x_2^{k_2} \cdots x_m^{k_m} g,$$

then f is replaced by

$$f = x_1^{l_1} x_2^{l_2} \cdots x_m^{l_m} g,$$

with $l_i := 1$ if $k_i \geq 1$, and $l_i := 0$ if $k_i = 0, i = 1, \ldots, m$. (The new f has the same zeros as the old one but the multiplicity is reduced at all hyperplanes $x_i = 0$, where the old f vanishes with multiplicity greater than 1). This causes, that the algorithm calculates no longer a Gröbner basis of the ideal generated by the input polynomials but of an ideal between this ideal and the radical ideal, which consists

of all polynomials vanishing at the common zeros of the input polynomials. Such power product factor can be detected easily in the distributive representation: its exponent vector is the vector of the componentwise minima over all exponent vectors. This vector is computed during the normalization of each polynomial and held in its data structure.

2.4 An additional criterion for detecting redundant pairs

If two polynomials p_1 and p_2 are known to have a common factor g,

$$p_1 = gf_1, p_2 = gf_2,$$

and if

$$\gcd\{lt(f1), lt(f2)\} = 1,$$

then we know in analogy to criterion 1 of Buchberger (1985), that the S-polynomial from p_1 and p_2 will reduce to 0. This criterion will be used for power products divisors g (see 2.3). This gives a new criterion for omitting superfluous reductions cf. Buchberger (1985) or Gebauer and Möller (1988). It can lead in cases, where many polynomials with common factors occur, to a better performance of the algorithm.

2.5 Gröbner factorization

For solving a system of algebraic equations

$$f_1(x) = 0, \ldots, f_r(x) = 0,$$

this means, for finding $Z(f_1, \ldots, f_r)$, the set of common zeros of all polynomials in $Ideal(f_1, \ldots, f_r)$, Gebauer and Möller tested in a preliminary version of REDUCE 3.3 the following procedure. If in the algorithm a polynomial $h \in Ideal(f_1, \ldots, f_r)$ is found, which can be factorized into

$$h = h_1 \cdot \ldots \cdot h_m,$$

then $Z(f_1, \ldots, f_r)$ is the union of the sets

$$Z(f_1, \ldots, f_r, h_k), k = 1, \ldots, m.$$

Thus when the algorithm did actually compute h, it branches. For each h_k in place of h it is continued resulting in the Gröbner bases for $Ideal(f_1, \ldots, f_r, h_k), k = 1, \ldots, m$. This factorization of the Gröbner basis calculations is also investigated by Davenport(1987).

2.5.1 Factorization tree

In the present package we have elaborated this approach by consequently testing all polynomials (input, intermediate and result) for power product factors and

by calling the REDUCE factorizer. If a factorization is detected, the state of the algorithm is encapsulated and for each of the factors a subproblem is generated and stacked into a list of open problems. Of course, the set of all such problems constitutes a tree with the initial problem of computing $Z(f_1, \ldots, f_r)$ as root.

2.5.2 Cancelling of redundant branches

Consider the subproblem for calculating $Z(g_1, \ldots, g_s)$ and let $h \in Ideal(g_1, \ldots, g_s)$ possess a factorization

$$h = h_1 \cdot \ldots \cdot h_m.$$

yielding the subsequent subproblems for calculating the sets $Z(g_1, \ldots, g_s, h_k), k = 1, \ldots, m$. It may turn out, that $Z(g_1, \ldots, g_s, h_k)$ is void, i.e. $1 \in Ideal(g_1, \ldots, g_s, h_k)$, or that $Z(g_1, \ldots, g_s, h_k)$ is contained in a previously computed set of zeros, such that its computation is superfluous. In order to avoid such superfluous sets of zeros, we associate to every problem a set of polynomials, starting with the empty set for the initial problem. If the set P is associated to the problem of calculating $Z(g_1, \ldots, g_s)$, then P is associated also to the problem of calculating $Z(g_1, \ldots, g_s, h_1)$. If P' is associated to the problem of calculating $Z(g_1, \ldots, g_s, h_k), 1 \leq k < m$, then $P' \cup \{h_k\}$ is associated to $Z(g_1, \ldots, g_s, h_{k+1})$. This gives the following criterion

(C): If we start calculating the Groebner basis of $Ideal(g_1, \ldots, g_s)$ and detect a polynomial in it, which belongs also to the polynomial set associated to $Z(g_1, \ldots, g_s)$, then we stop the Gröbner basis calculation and remove the problem of computing $Z(g_1, \ldots, g_s)$ from the set of open problems.

The usage of this criterion is justified by the following reason. If f belongs to the polynomial set associated to $Z(g_1, \ldots, g_s)$ and $f \in Ideal(g_1, \ldots, g_s)$, then a $j < s$ exists by construction, such that f and g_{j+1} are both factors of g_j, f the lower indexed factor. Since $f \in Ideal(g_1, \ldots, g_s)$, we have $Ideal(g_1, \ldots, g_j, f) \subseteq Ideal(g_1, \ldots, g_s)$. Therefore $Z(g_1, \ldots, g_s) \subseteq Z(g_1, \ldots, g_j, f)$ holds, i.e. the problem of computing $Z(g_1, \ldots, g_s)$ is superfluous.

2.5.3 Restrictions imposed by the application

In some applications only solutions are of interest which fullfill certain algebraic restrictions. In some examples, we are only interested in nonnegative (real) solutions. (For instance negative mass makes no sense.) So we implemented a hook for the used to decide via a procedure whether a polynomial violates these restrictions. As a standard procedure the testing for nonnegative solutions is present and can be activated on request

- A polynomial $\sum a_i x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} + a_0$ with all a_i and a_0 positive has no zero $(x_1, \ldots, x_n)$ with all x_i nonnegative (and real). So the calculation branch can be cancelled.

- A polynomial $\sum a_i x_1^{i_1} \cdot \ldots \cdot x_n^{i_n}$ without an absolute term a_0 and with all a_i positive can vanish only if all of its summands vanish. This is possible only if in any summand at least one variable is zero. In most cases there are several possible combinations. The list of combinations play the same role as a factorization and is handled in the same manner.

Although this approach is independent from factorization, it plays its most important role in this context: the main problem of factorization is the fight against the great number of branches. The earlier a polynomial without (componentwise) nonnegative zeros is detected in a branch, the earlier the computation in this branch may be interrupted (at the expense of an additional branching or a complete cancellation of the branch).

2.6 Modular coefficients

Modular techniques are helpful, when the coefficients of the output polynomials are expected to be of the same (moderate) size as the input coefficients, but intermediate polynomials with big coefficients occur. We have three methods for computing a Gröbner basis over the rationals by means of modular techniques.

2.6.1 The direct method

Take a (sufficiently big) prime p. Transform each input polynomial to a polynomial over $\mathbb{Z}_p$ by substituting its coefficient c/d by a, such that $c \equiv a \cdot d \bmod p, 0 \leq a < p$. Compute a reduced Gröbner basis over $\mathbb{Z}_p$ with each leading coefficient normalized to 1. Reconstruct each Gröbner basis coefficient $a \bmod p$ to a rational number by means of the algorithm of Wang (1981). The resulting polynomials constitute the reduced normalized Gröbner basis over $\mathbb{Q}$, if p is a sufficiently big lucky prime.

2.6.2 The hybrid method

Many polynomials come from scientific applications, where the coefficients are only approximations to some 'real life' quantities. So the coefficients are typically floating point numbers. On the other hand Buchberger's algorithm requires exact coefficients. Therefore we compute via REDUCE rational numbers approximating the floating point values and from that we produce a modular number. The modular number and the floating point value then are paired for Buchberger's algorithm to a hybrid number. The arithmetic is defined componentwise. The modular number now controls Buchberger's algorithm allowing exact calculations, while the floating point values are calculated as side effects. At termination of the algorithm, the polynomials with the modular part of the hybrid numbers as coefficients constitute a Gröbner basis. The polynomials with the float number parts as coefficients are then expected to be approximations to the Gröbner basis polynomials of the initial 'real life' problem. Because we do

not reconstruct the 'true values' from the modular number, the prime number can be of moderate size.

2.6.3 The parallel method

As proposed by Knuth (1981), each integer a mod $p_1 \cdot \ldots \cdot p_s$ with pairwise different primes $p_i, i = 1, \ldots, s$, can be represented uniquely by an s-tuple $(a_1, \ldots, a_s)$ with $a \equiv a_i \bmod p_i$ and given the a_i's the integer a can be reconstructed by the Chinese remainder theorem. Therefore, we proceed as in 2.6.1 in order to find Gröbner bases over $\mathbb{Z}_{p_i}$. The arithmetic operations for these tuples are defined componentwise

$$(a_1, \ldots, a_s) \bullet (b_1, \ldots, b_s) = (a_1 \bullet b_1, \ldots, a_s \bullet b_s)$$

with $\bullet \in \{+_{mod}, -_{mod}, *_{mod}, /_{mod}\}$.

When one component of a nonzero tuple is 0, then the corresponding p_i is unlucky and henceforth ignored for the reconstruction of the integer a by the Chinese remainder theorem. In that case, the reconstructed a by the Chinese remainder theorem is at most exact modulo $p_1 \cdot \ldots \cdot p_s/p_i$.

3. Technical improvements

3.1 Deferred branching for Gröbner factorization

When doing Gröbner factorizations as described in 2.5 for a problem with many variables and a 'sparse' coefficient pattern, we detected, that different branches starting from one $h = h_1 \cdot \ldots \cdot h_m$ had to do the same calculations for those polynomials depending only on variables different from those in $h_1, \ldots, h_m$. So we implemented a delay: if a branching point is encoutered, the algorithm sets aside the factors and continues the calculations for those pairs with leading terms in variables disjoint to those of $h_1, \ldots, h_m$. Here new factorizations can be found, which are handled in the same way. Finally we have several levels of factorization at the same time giving a rich information for the configuration of subproblems.

In most cases factors themselves are of very low degree and so we do an additional reduction of all polynomials in one branch with its creating factors, which gives a further reduction of problem size.

3.2 Dynamic selection of coefficient domains

The interface for the coefficient manipulation of the distributive polynomials was redesigned completely. So it now has one separate package for each coefficient domain; one of them is the interface to the REDUCE standard quotients (SQ). Others are those for $\mathbb{Q}, \mathbb{Z}_p, H(p, f), M(p_1, \ldots, p_s)$. Additional packages can be plugged in easily. As default the algorithm expects coefficients in the most general domain SQ. In its starting phase it does a general data type analysis and selects automatically the simplest arithmetic for the given problem.

This is an efficient mixture of static and dynamic typing: the type analysis is done dynamically once in the starting phase. In the main working phase there is no more type checking because then all switches are already set. This procedure is justified by the fact, that the type analysis is very small compared with Buchberger's algorithm itself.

3.3 Data Representation

In the original version polynomials were represented as standard lists. This organization is efficient for the manipulation, but it consumes too much storage if large problems with thousands of intermediate polynomials are handled and it does not allow to link additional information to the individual polynomials.

So the polynomials now are embedded in an object oriented structure, which allows several different memory organizations with compatible generic access methods at the same time. Each polynomial posesses a property list for descriptive slots, e.g. a name of the polynomial for trace purposes, its known factors, the subspace of its variables. Because of the object oriented approach different memory organizations can be used at the same time. By default two organizations are active: one simple (in fact the traditional list in an encapsulated version) for temporary intermediate results and a compressed representation for those polynomials which will survive local calculations (e.g. the H-polynomials). So calculations with many intermediate H-polynomials are possible in memory. If the memory limit is exceeded, an additional organization is available which places polynomials on secondary storage. The only inference between the algorithm and the memory organization is the explicit marking of a polynomial to be of the 'survival' type.

3.4 Polynomial reduction sets

It turned out, that Gröbner bases calculations are more efficient, when we distinguish between the set G of all polynomials (from input and H-polynomial calculation) and the set F obtained from G by cancelling redundant elements; F is finally the reduced Gröbner basis whereas G is used in place of F for reducing the polynomials during the algorithm as sketched in 1.1. If G contains a great number of polynomials, the search for a candidate for the reduction may require some time. Therefore G is stored as a tree, which reflects the structure of the lt part such that the optimal polynomial for a given reduction step can be found by 'addressing' it.

4: Cray specific optimizations

4.1 Coefficient arithmetic

By analyzing the run-time behaviour of the original package it soon became clear, that the heavy usage of arbitrary precision integer arithmetic was one main bottleneck. So we rewrote the PSL 'big' package now using the Cray vector hardware. This gave a significant run-time reduction and stressed the superiority of $\mathbb{Z}$ based computations compared to $\mathbb{Q}$ based computations.

An obviously 'native' area for a vector computer are the $\mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_s}$ coefficients, where the transformation from the algorithmic level to the machine level is (almost) obvious. The operations $+, -, *$ can be mapped to one vector instruction cycle each, doing the basic integer operation and adjusting components with overflow (masking out non overflow components). The pseudo-division is a little bit more complicated because the number of loops needed by the extended Euclidean algorithm depends of the magnitude of the input values; here the loop is performed until all values are finished; the components which are ready earlier are ruled out by masking instructions.

4.2 Exponent vector processing

The processing of exponent vectors is the second bottleneck of distributive polynomial processing. Exponent vectors traditionally are stored as lists of (small) integers. Most of the processing time spent with exponent vectors is needed for the comparison (term ordering). The list structure is very adequate for this type of operation: two vectors are compared from 'left to right' breaking when the first non equal position is found. Total degree requires additionally a horizontal sum over the vectors. We have improved this type of processing by exploiting the pipelined memory access of the vector computer: while processing two exponent vectors four items are loaded in parallel in each cycle, the two elements to be processed and the two pointers to the rest of the structures.

5. Examples

5.1 A molecular structure

We received a set of algebraic equations

$$f_i(y_1, y_2, y_3) = 0, i = 0, \ldots, 3,$$

from A. Dress (1987). It describes the molecular structure of a cyclic carbon hydrogen molecule with six nodes. The polynomials f_0, f_1, f_2, f_3 are given by

$$f_0 := \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 8/3 & y_1 & 8/3 & 1 \\ 1 & 1 & 0 & 1 & 8/3 & y_2 & 8/3 \\ 1 & 8/3 & 1 & 0 & 1 & 8/3 & y_3 \\ 1 & y_1 & 8/3 & 1 & 0 & 1 & 8/3 \\ 1 & 8/3 & y_2 & 8/3 & 1 & 0 & 1 \\ 1 & 1 & 8/3 & y_3 & 8/3 & 1 & 0 \end{vmatrix}, f_1 := \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 8/3 & y_1 & 8/3 \\ 1 & 1 & 0 & 1 & 8/3 & y_2 \\ 1 & 8/3 & 1 & 0 & 1 & 8/3 \\ 1 & y_1 & 8/3 & 1 & 0 & 1 \\ 1 & 8/3 & y_2 & 8/3 & 1 & 0 \end{vmatrix},$$

$$f_2(y_1, y_2, y_3) := f_1(y_2, y_3, y_1),$$
$$f_3(y_1, y_2, y_3) := f_1(y_3, y_1, y_2).$$

In order to find the common zeros $Z(f_0, f_1, f_2, f_3)$ of $Id\grave{e}al(f_0, f_1, f_2, f_3)$, we proceed as described in 2.5. Buchberger's algorithm is performed with the following features:

- inverse lexicographical ordering
- integer coefficients
- Gröbner factorization.

The factorization produces eight Gröbner bases. $Z(f_0, f_1, f_2, f_3)$ is the union of the sets $Z(G_i), i = 1, \ldots, 8$, where the Gröbner bases $G_1, \ldots, G_8$ are

$$\begin{aligned} G_1 : \; & \{3y_1y_2 + 3y_1y_3 - 22y_1 + 3y_2y_3 - 22y_2 - 22y_3 + 121, \\ & 27y_1y_3^2 - 198y_1y_3 + 75y_1 + 27y_2y_3^2 - 198y_2y_3 + 75y_2 - 198y_3^2 + 1164y_3 + 250, \\ & 81y_2^2y_3^2 - 594y_2^2y_3 + 225y_2^2 - 594y_2y_3^2 + 3492y_2y_3 + 750y_2 + 225y_3^2 + 750y_3 - 14575\}, \end{aligned}$$

$$\begin{aligned} G_2 : \; & \{27360y_1 + 27360y_2 - 1377y_3^3 - 2925y_3^2 + 59685y_3 - 304855, \\ & 82080y_2^2 - 4131y_2y_3^3 - 8775y_2y_3^2 + 179055y_2y_3 - 914565y_2 + 30699y_3^3 + 79695y_3^2 \\ & \quad - 1192215y_3 + 2952125, \\ & 243y_3^4 + 540y_3^3 - 9630y_3^2 + 25500y_3 - 26125\}, \end{aligned}$$

$$\begin{aligned} G_3 &: \{3y_1 - 11, 9y_2 - 25, 3y_3 - 11\}, \\ G_4 &: \{3y_1 - 11, 3y_2 - 11, 3y_3 - 11\}, \\ G_5 &: \{9y_1 - 25, 3y_2 - 11, 3y_3 - 11\}, \\ G_6 &: \{3y_1 + 5, 3y_2 - 19, 3y_3 + 5\}, \\ G_7 &: \{3y_1 + 5, 3y_2 + 5, 3y_3 + 5\}, \\ G_8 &: \{3y_1 - 19, 3y_2 + 5, 3y_3 + 5\}. \end{aligned}$$

Denoting the three polynomials of G_1 by g_1, g_2, g_3, we find

$$76800g_1 = (81y_2^2y_3 - 369y_2^2 - 594y_2y_3 + 1842y_2 + 225y_3 - 1375)g2$$

$$+(123y_1 + 123y_2 - 27y_1y_3 - 27y_2y_3 + 198y_3 - 614)g3$$

i.e. g_1 vanishes at the common zeros of g_2 and g_3. Using y_3 for a free parameter, the common zeros of g_1, g_2, g_3 are hence (y_1, y_2, y_3) with

$$y_1 := \frac{99y_3^2 - 582y_3 - 125 \pm \sqrt{\Delta}}{27y_3^2 - 198y_3 + 75}, y_2 := \frac{99y_3^2 - 582y_3 - 125 \mp \sqrt{\Delta}}{27y_3^2 - 198y_3 + 75},$$

where $\Delta := 7776y_3^4 - 107136y_3^3 + 489024y_3^2 - 835200y_3 + 380000$.

The polynomials of G_2 have exactly eight zeros (y_1, y_2, y_3) in common. The third component y_3 is a zero of the irreducible polynomial

$$243y_3^4 + 540y_3^3 - 9630y_3^2 + 25500y_3 - 26125$$

(i.e. four different choices), the second component y_2 is then determined by

$$82080y_2^2 - 4131y_2y_3^3 - 8775y_2y_3^2 + 179055y_2y_3 - 914565y_2 + 30699y_3^3 + 79695y_3^2$$

$$-1192215y_3 + 2952125 = 0$$

(i.e. two choices for fixed y_3), and the first component y_1 is determined for given y_2 and y_3 by

$$27360y_1 + 27360y_2 - 1377y_3^3 - 2925y_3^2 + 59685y_3 - 304855 = 0$$

The common zeros of the last six bases can be read off immediately.

The overall time for computing the eight Göbner bases is
3.5 sec (using a Cray X-MP 2400),
104 sec (using a SUN 3-260).
In this example, long integer coefficients occured in the intermediate computations.

5.2 Modular Gröbner computations

As example for the efficiency of vector hardware usage a test calculation may serve: An example in Böge et al.(1986) originating from W. Trinks produces moderate long integer coefficients and so we have calculated it with 63 component modular vectors with different technical supports: sequential processing of vectors (compatible version of this arithmetic) and processing by Cray vector instructions. Of course, this is no real application: the problem and the coefficients are too small and so the computation of the reduced normalized Gröbner basis w.r.t. the INVLEX ordering is totally dominated by the reconstruction effort.

TRINKS1 (6 polynomials in 6 variables over $M(p_1, .., p_{63})$)

sequential processing	5.8 sec
usage of vector hardware	0.7 sec
Reconstruction of coefficients:	8.5 sec

Another example in Böge et al.(1986) originating from M.Rose was mentioned as a case where a computation with lexicographical order was impossible. With our Cray installation we computed the Gröbner basis over $\mathbb{Z}$. But here the modular version is much more efficient, because the Gröbner basis has some very long coefficients (300decimal digits).

ROSE	(3 polynomials in 3 variables)	
	ITD :	see 5.3
	INVLEX over $\mathbb{Z}$:	2288 sec. Cray X-MP in a job with 2.5 million (64-bit-)words of memory.
	INVLEX over $M(536870909)$:	92 seconds Cray X-MP
	INVLEX over $M(p_1,..,p_{120})$:	148 sec. Cray X-MP (for Buchberger's algorithm) plus 112 sec. Cray X-MP (for the coefficient reconstruction)

Of course, the single modular computation indicates as result only the monomial layout. It has the aspect of a 'pioneer' computation. The calculation with coefficient vectors of length 120 modulo primes of moderate size (less than 21 bits each) allows the reconstruction of the true basis coefficients; when normalized by REDUCE common denominator technique these are identical with the coefficients calculated over $\mathbb{Z}$. Eight of the 120 primes were unlucky; they could not take part in the reconstruction. It turned out, that 100 primes of this moderate size are the minimum for the reconstruction of the coefficients. It should be noted here, that because of the vector hardware the time of Buchberger's algorithm with 120 modular coefficients is not much slower than that with one modulus; in fact the additional time needed for the vector version has its origin mainly in the fact that more dynamic memory is wasted (additional garbage collections).

5.3 Some comparisons

For comparisons we have done some calculations with examples listed in Böge et al. (1986). We stress, that our Gröbner package is designed for 'real life' applications requiring the power of a super computer. Therefore we omitted the tests of too simple examples. But for economical reasons, we also did not try all hard problems of Böge et al. We present six medium range problems as candidates for testing the various features of an implementation. The following statistics demonstrate the influence of the options on the computing time. All these examples are no cases where there is an urgent need for the Cray power.

All computing times are given in seconds of a Cray X-MP; the time needed for garbage collections during the computations is included (the memory was limited to 1 million words).

	Bas	D=Q	Lorg	Prered	ltred	Fact	Orig
TRINKS1	0.3	6.3	0.3	0.4	0.2	2.1	17.0
GERDT	13.6	31.1	12.5	13.5	23.1	9.8	32.1
HAIRER3	12.9	14.8	11.7	12.6	136.0	> 999	38.1
KATSURA4	0.8	2.6	0.8	0.8	3.0	1.7	6.0
ROSE itdg	2.1	8.9	1.6	2.1	4.2	26.9	20.9
GONNET	1.4	1.4	0.9	1.0	2.3	0.8	1.1

Meaning of the columns:

Bas: A 'basic' package of options: Domain = $\mathbb{Z}$, polynomials compressed, no prereduction of input, full reduction, no factorization. The subsequent columns (except the last one) differ from this column by only one modification each.

D=Q: Usage of rational numbers instead of integers.

Lorg: Switching off internal compression (cf. 3.4).

Prered: Prereducing of the input polynomials.

ltred: Only lt-reduction.

Fact: Factorizing enabled.

Orig: usage of the original Gröbner package of REDUCE 3.3

The statistics give the following result: Domain $\mathbb{Z}$ is 'faster' than $\mathbb{Q}$ in almost all cases (except the last one, where it is equal). The internal compression causes in general a neglegiable overhead for these medium range problems; only the last case suffers explicitly from this option; it causes an almost uniform overhead with the larger problems and one should take into account, that avoiding a memory overflow is a question of computability at all. The prereduction makes sense in all cases: if it produces no time saving, it costs not much. Only one case was found, where restricting the reduction to the lt-parts gives a tiny speedup. The benefit of factorization varies from problem to problem very much; if a factorization is found, it can shorten the calculation (and it gives much better usable results); if not, it requires an important additional computing time.

5.4 Calculations for large chemical reaction systems

These examples have their origin in a system of ODE's describing chemical reaction systems. Up to 60 polynomials in up to 60 variables (number of variables and number of equations not necessarily equal). The polynomials are extracted from the ODE system without further preprocessing. Their maximal degree is 3. The original coefficients are floating point numbers. Most polynomials cite only a few number of variables ('sparse' system). Using the original algorithm the bigger

problems consume all available resources in an exploding manner without producing any result. Therefore we used the following configuration of Buchberger's algorithm.

- reducing the input polynomials with themselves as long as lt parts can be cancelled (actually 5 cycles), cf. 2.2.
- cancellation of multiple power product factors, cf. 2.3.
- factorization on the base of power product factors, cf. 2.5 .
- inverse total degree ordering.
- hybrid coefficients, cf. 2.6.2.

Using this configuration, we find a set of (partial) Gröbner bases. Some of these bases have a very simple structure, e.g. only linear polynomials, such that the solution can be read off immediately.

The following example HAVC4 was given to us by P. Deuflhard (1986). It consists of 50 polynomials in 37 variables. The longest polynomial has 31 terms. Some of the polynomials have only 1 term. We found hundreds of partial Göbner bases which were all of the same simple structure, such that their common zeros could be detected without additional computations. The 'rate' on the Cray X-MP was 'one base per 4 CPU seconds'. An inspection gave, that most of these bases have only solutions with at least one negative component. Therefore the

- restriction for nonnegative values, cf. 2.5.3,

is imposed. This reduces the amount of solutions and of calculation branches dramatically. The algorithm now terminates in 358 seconds (overall computing time incl. garbage collections) on the Cray giving 330 partial Gröbner bases. A postprocessing analyse detects that most of these solutions are still redundant in the sense, that they describe a subspace of other solutions. The final number of independent solution spaces is eleven.

References

J.W. Anderson, W.F. Galway, R.R. Kessler, H. Melenk, W.Neun: *Implementing and Optimizing Lisp for the Cray*, IEEE Software, July 1987.

W.Böge, R. Gebauer, H. Kredel: *Some Examples for Solving Systems of Algebraic Equations by Calculating Groebner Bases*, J. Symb. Comp. 1986, vol.2, pp. 86 – 98.

B. Buchberger: *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal*, Ph.D. thesis, Innsbruck, 1965.

B. Buchberger: *Gröbner bases: An algorithmic method in polynomial ideal theory.* (In: Multidimensional Systems Theory, ed.: N.K. Bose) D. Reidel Publ. Comp. 1985, pp. 184 – 232.

J. H. Davenport: *Private communication 1987.*

P. Deuflhard: *Private communication 1987.*

A. Dress: *Private communication 1987.*

R. Gebauer, H.M. Möller: *On an installation of Buchberger's algorithm.* To appear in J. Symb. Comp. 1988.

A.C.Hearn: *REDUCE User's Manual, Version 3.3*, The Rand Corporation, 1987.

R. Jenks et al.: *Scratchpad II, an experimental computer algebra system, abbreviated primer and examples*, IBM Thomas Watson Research Center, Yorktown Heights, N.Y. 10598, 1986.

D.E. Knuth: *The Art of Computer Programming Vol 2*: Seminumerical Algorithms, 2nd edition, 1981.

H.M.Möller: *On the construction of Gröbner bases using syzygies*, to appear in J.Symb. Comp. 1988.

H. Melenk, W. Neun: *REDUCE Users Guide for Cray 1/ X-MP Series Running COS*, Konrad-Zuse-Zentrum Berlin, Technical Report TR 87-4 , 1987.

H. Melenk, W.Neun: *Usage of Vector Hardware for LISP Processing*, Konrad-Zuse-Zentrum Berlin, 1986.

P. Wang: *A p-adic algorithm for univariate partial fractions*, Proc. SYMSAC'81, ACM, pp. 212 – 216, 1981.