On parallel computation of Gröbner bases

Anton Leykin
Dept. of Math., Stat., and Comp. Science
University of Illinois at Chicago
leykin@math.uic.edu

Abstract

We have developed a coarse-grain parallelization of the Buchberger algorithm for computing Gröbner bases in algebras of linear differential operators. The implementation of this algorithm provides good speedups on the majority of examples coming from these noncommutative algebras, which are superior to the speedups achieved for (commutative) polynomial rings.

1 Introduction

Algorithms for computing Gröbner bases have become a standard hard problem for computer scientists, since their complexity is proved to have a sharp double exponential bound. That is in the case of (commutative) polynomials; if one considers, for example, the Weyl algebra, this complexity is not known, though it is guaranteed to be worse.

As parallel computation becomes standard and supercomputing facilities more accessible, we turn our attention to parallelization of Gröbner bases computations. This topic has been explored both by mathematicians and computer scientists (see [1], [9], [11], [3]) for Gröbner bases in rings of (commutative) polynomials. Also, there was work done in this direction in more specialized settings (e.g. see [10]).

At the beginning of the project, on top of aiming at constructing a practical implementation of a parallel Buchberger algorithm, we were especially interested in computing Gröbner bases in algebras of linear differential operators. For these algebras the elementary operations (e.g. multiplication of differential operators) are more expensive than in case of polynomials, also there are no elaborate techniques (e.g. see [4]) developed for eliminating the unnecessary s-pairs (ones that reduce to 0), hence, optimizing the Buchberger algorithm. These observations made us believe that parallelization in the noncommutative case would yield better results compared to the commutative one for two reasons:

1. The cost of a reduction step dominates the cost of communication.

2. Updates to the basis are not often because of a larger number of 0-reductions.

Our instincts have not betrayed us. Our implementation – coded in C++ using MPI [7] for communication interface – displays speedups that on average are better in a noncommutative setting than in a commutative one.

2 Preliminaries

In this chapter we shall deal with associative algebras of polynomial type over a field k. In constructing those we use the variables $x_i, \partial_i, s_i, t_i$ where $i \in \mathbb{Z}$ that satisfy the following relations: $[\partial_i, x_i] = 1, [s_i, t_i] = t_i$, where $[a,b] = ab - ba$, and all the pairs of variables that are not mentioned above commute.

Using these variables we can describe the Weyl algebra $A_n = k\langle x_1, ..., x_n, \partial_1, ..., \partial_n\rangle$. Another algebra we used for our experiments is the so-called PBW algebra (see [2])

$$\begin{aligned} P_{n,p} &= A_n\langle s_1, ..., s_p, t_1, ..., t_p\rangle \\ &= k\langle s_1, ..., s_p, t_1, ..., t_p, x_1, ..., x_n, \partial_1, ..., \partial_n\rangle. \end{aligned}$$

If we fix the order of the variables, each element of any algebra allowed by the above description can be written uniquely as a polynomial with monomials with variables written in the specified order: we call this polynomial the *normal form*. For computational purposes we would assume that we always operate with algebra elements in the normal form. According to [8] our algebras are of *solvable type*, i.e. are eligible for Gröbner bases techniques similar to the ones in the ring of polynomials.

In what follows we describe a simple version of the Buchberger algorithm, which is a (sequential) completion algorithm used to compute Gröbner bases.

Let $A = k\langle z\rangle = k\langle z_1, ..., z_n\rangle$ be an associative algebra where variables z_i have names from the list at the beginning of the section and are subject to the corresponding relations. Let f be an element of A having the normal form

$$f = \sum_{\alpha \in \mathbb{Z}^n_{\geq 0}} a_\alpha z^\alpha,$$

where all but a finite number of a_α equal 0. Denote by $\mathrm{inM}(f)$ and $\mathrm{inC}(f)$ the *initial monomial* and *initial coefficient* of f, call $\mathrm{in}(f) = \mathrm{inC}(f)\mathrm{inM}(f)$ the *initial term* of f. For a left ideal $I \subset A$ we define an *initial ideal* $\mathrm{in}(I) = A \cdot \{\mathrm{in}(f) \mid f \in I\}$.

A generating set G of a left ideal $I \subset A$ is called a Gröbner basis if $\mathrm{in}(I) = A\,\{\mathrm{in}(g) \mid g \in G\}$.

Given a polynomial $f \in A$ and a finite subset $B \subset A$ we can perform reduction as follows:

Algorithm 1 $f' =$ **REDUCE**(f, B)

```
    $f' := f$
    WHILE there is $g \in B$ such that inM$(g)$ divides inM$(f')$
        Set $f' := f' - (in(f')/in(g))g$
    END WHILE
```

For two polynomials $f, g \in A$ we define the s-polynomial

$$\mathrm{sPoly}(f,g) = \mathrm{inC}(g)\frac{l}{\mathrm{inM}(f)}f - \mathrm{inC}(f)\frac{l}{\mathrm{inM}(g)}g,$$

where $l = \mathrm{lcm}(\mathrm{inM}(f), \mathrm{inM}(g))$. Then an alternative definition could be given as follows: A generating set G of a left ideal $I \subset A$ is a Gröbner basis if **REDUCE**$(\mathrm{sPoly}(f,g), G) = 0$.

The following is a sketch of the simplest completion algorithm known as Buchberger algorithm.

Algorithm 2 $G =$ **BUCHBERGER**(B)

```
    $G := B$, $S := \{(f,g) | f,g \in B,\ f \neq g\}$
    WHILE $S \neq \emptyset$ do
        Pick $(f,g) \in S$ according to the "strategy"
        $S := S \setminus \{(f,g)\}$
        $h :=$ REDUCE$(sPoly(f,g), G)$
        IF $h \neq 0$
            $S' := \{(h,g) | g \in G\}$
            Eliminate redundant s-pairs from $S$ and $S'$
                according to "criteria"
            $S := S \cup S'$ and $G := G \cup \{h\}$
        END IF
    END WHILE
```

Here "strategy" refers to an algorithm of determining which s-pair to consider next; the most popular strategies are sorting s-pairs either by total degree or by "sugar" degree (see [6]).

"Criteria" are sets of rules helping to eliminate redundant s-pairs, i.e. the ones that are guaranteed to reduce to 0 under selected strategy. Most commonly used criteria are the ones you may find in Gebauer-Möller [5]. Several of these criteria generalize for the noncommutative setting, however, there are some that work only in the commutative case. A simple example is the so-called T-criterion: if $\gcd(\mathrm{inM}(f), \mathrm{inM}(g)) = 1$ then **REDUCE**$(sPoly(f,g), \{f,g\}) = 0$, which does not hold for $f = \partial$ and $g = x$ in the Weyl algebra A_1.

3 Parallel Buchberger algorithm

As you may see, though we have a loop in the above algorithm and the **REDUCE** tasks at each iteration seem routine, these tasks are not independent of each other, since the reduction basis G may change from one step to another.

On the other hand, it is not hard to imagine many s-pairs in a row reducing to 0: if we knew ahead of time that this would happen we could have assigned these routine 0-reductions to different processors. This point, in our opinion, is the starting motivation for coarse-grain parallelization of Buchberger algorithm.

What we developed is an algorithm similar to Attardi-Traverso [1]. It uses a master-slave paradigm and distributed computing in the following way. Let $n+1$ threads be at our disposal, each assigned to a node with one processor. Make one of the nodes the master and the rest the slaves; the master can communicate with each of the slaves, however, there is no communication between the slaves. Each slave maintains a local copy of reduction basis $B \in A$ and is capable of completing two tasks: updating for the basis upon reception of an update message from the master and reducing a polynomial $h \in A$ supplied by the master with respect to B.

Algorithm 3 SLAVE

```
    $B := \emptyset$
    LOOP
        IF RECEIVE(MASTER, $h$, update) THEN
            $B := B \cup \{h\}$
        END IF
        IF RECEIVE(MASTER, $h$, reduce) THEN
            SEND(MASTER, REDUCE$(h, B)$,
                slaveDone)
        END IF
    END LOOP
```

Here **SEND**$(THREAD, DATA, TAG)$ sends the message tagged with TAG containing $DATA$ to $THREAD$ and **RECEIVE**$(THREAD, DATA, TAG)$ receives the message with TAG containing $DATA$ from $THREAD$. **SEND** is nonblocking and **RECEIVE** returns $TRUE$ if the message has been received. Note that, in practice, a slave is terminated by a separate message from the master, but here we prefer to use an infinite loop for simplicity.

Let us describe the master now. In what follows an s-pair shall be represented by a quadruple $(f, g, h, \texttt{status})$ containing two additional elements: h is a partially reduced s-polynomial sPoly(f, g) and `status` is either `red` or `nonRed` depending on whether the s-polynomial has been reduced completely or h may still be reduced with respect to the current basis G. The variable m_i stores the s-pair that is being reduced by **SLAVE**$_i$, $i = 1, ..., n$.

Algorithm 4 G = **MASTER**(B). *Computes a Gröbner basis of the left ideal generated by $B \subset A$ using threads* **SLAVE**$_i$, $i = 1, ..., n$.

```
FOR $i = 1, ..., n$
    $m_i := \emptyset$
    FOR every $g \in B$
        SEND(SLAVE$_i$, $g$, update)
    END FOR
END FOR
$G := B$
$S := \{(f, g, sPoly(f, g), \texttt{nonRed}) | f, g \in B,\ f \neq g\}$
WHILE $S \neq \emptyset$ do
    IF RECEIVE(SLAVE$_i$, $h$, slaveDone) THEN
        Let $(f, g, ..., \texttt{nonRed}) = m_i$
        Replace $m_i$ with $(f, g, h, \texttt{red})$ in $S$
        $m_i := \emptyset$
    END IF
    IF $\exists m_i = \emptyset$ and
        $\exists sp = (f, g, h, \texttt{nonRed}) \in S$ THEN
        SEND(SLAVE$_i$, $h$, reduce)
        $m_i := sp$
    END IF
    Let $sp = (f, g, h, \sigma)$ be the first s-pair in $S$
    IF $\sigma = \texttt{red}$ THEN
        Remove $sp$ from $S$
        IF $h \neq 0$ THEN
            $S' := \{(h, g, sPoly(f, g), \texttt{nonRed}) | g \in G\}$
            Apply "criteria" to $S$ and $S'$
            $S := S \cup S'$ and $G := G \cup \{h\}$
            Reorder $S$ according to the "strategy"
            FOR every $(f, g, h', \texttt{red}) \in S$
                IF $h$ divides $h'$ THEN
                    status = nonRed
                END IF
            END FOR
            FOR $i = 1, ..., n$
                SEND(SLAVE$_i$, $h$, update)
            END FOR
        END IF
    END IF
END WHILE
```

Notice that the presented algorithm clearly preserves the strategy, since no modifications of G and S are possible before the first s-pair in the queue S is completely reduced. Of course, one may experiment with a variation of this algorithm that takes a whatever pair in the queue with its s-polynomial completely reduced to a nonzero and moves on with these modifications, however, our test runs show that in such cases results are highly unpredictable and the output may heavily depend (in a random fashion) on the number of slaves, architecture of the used hardware, as well as random events in the system. Such an approach pays off very rarely and, when it does, there is no guarantee that the obtained good results can be consistently reproduced.

4 Experimental results

We have implemented the algorithm in C++ using MPI libraries for message passing. Let us point out that at the moment computations are done only for algebras over $\mathbb{Z}/p\mathbb{Z}$ for a large prime number p.

Our experiments were conducted on a shared-memory SGI Origin 3800 supercomputer equipped with 500 MHz R14000 processors. Since our study was of a theoretical nature, we developed a simple (sequential) simulator that records the sequences of events happening in the simulated walltime. The lengths of communication events are proportional to the lengths of the sent messages, on top of which a fixed time is added to account for the packing, putting and getting a message from the message queue. It was possible to tweak the parameters so that the simulated results match closely to the real experiments outcomes.

At the present, there does not exist a good library of hard test examples for noncommutative Gröbner bases computation. We tried our software for the "natural" input, i.e. problems that arose naturally in our research. Here are typical ideals that we computed Gröbner bases for:

- The elimination ideal in algebra $A = k\langle u, v, t, \partial_t, x, y, z, \partial_x, \partial_y, \partial_z\rangle$ with a monomial order eliminating commutative variables u and v that leads to the annihilator of f^s in $A_3[s]$ for $f = xyz(x + y)(x + z)$:

  $$I_1 = A \cdot (-ux^3yz - ux^2y^2z - ux^2yz^2 - uxy^2z^2 + t, 3u\partial tx^2yz + 2u\partial txy^2z + 2u\partial txyz^2 + u\partial ty^2z^2 + \partial x, u\partial tx^3z + 2u\partial tx^2yz + u\partial tx^2z^2 + 2u\partial txyz^2 + \partial y, u\partial tx^3y + u\partial tx^2y^2 + 2u\partial tx^2yz + 2u\partial txy^2z + \partial z, uv - 1, 5t\partial t + x\partial x + y\partial y + z\partial z + 5)$$

- The elimination ideal in the PBW-algebra $A = k\langle t, s, x, y, z, \partial_x, \partial_y, \partial_z\rangle$ with a product monomial order that eliminates t and then s, which leads to the same annihilator ideal via route laid out in [2]:

$$I_2 = A \cdot (tx^3yz + tx^2y^2z + tx^2yz^2 + txy^2z^2 + s,$$
$$3tx^2yz + 2txy^2z + 2txyz^2 + ty^2z^2 + \partial x,$$
$$tx^3z + 2tx^2yz + tx^2z^2 + 2txyz^2 + \partial y,$$
$$tx^3y + tx^2y^2 + 2tx^2yz + 2txy^2z + \partial z)$$

To compute **BUCHBERGER**(I_1) it took approximately 1 minute with 1 slave and 11 seconds with 10 slaves. For **BUCHBERGER**(I_2) it was 15 and 2.5 seconds, respectively, which provides a verification of Ucha-Castro results in [12] on the comparison of two different methods of computing the annihilator above.

For the commutative case we used such popular benchmarks as `cyclic6`, the ideal of $k[a, b, c, d, e, f]$ generated by polynomials

$$abcdef - 1,$$
$$abcde + abcdf + abcef + abdef + acdef + bcdef,$$
$$abcd + bcde + abcf + abef + adef + cdef,$$
$$abc + bcd + cde + abf + aef + def,$$
$$ab + bc + cd + de + af + ef,$$
$$a + b + c + d + e + f,$$

and `cyclic7`, the ideal of $k[a, b, c, d, e, f, g]$ generated by polynomials

$$abcdefg - 1,$$
$$abcdef + abcdeg + abcdfg + abcefg$$
$$+abdefg + acdefg + bcdefg,$$
$$abcde + bcdef + abcdg + abcfg + abefg$$
$$+adefg + cdefg,$$
$$abcd + bcde + cdef + abcg + abfg + aefg + defg,$$
$$abc + bcd + cde + def + abg + afg + efg,$$
$$ab + bc + cd + de + ef + ag + fg,$$
$$a + b + c + d + e + f + g,$$

as well as the family of ideals of $k[x, y, z]$ generated by x^m, y^m, z^m and a random polynomial.

All the examples that we used as benchmarks produce between 100 and 2000 s-pairs during the computation, which is a relatively small number. However, our objective was simply to see how our implementation behaves for equally intense (in terms of the number of s-pairs) computations in commutative and noncommutative settings.

Figure 1 shows that computing with a small number of processors (slaves) results in good speedups in both cases. These fall behind what would be considered perfect (linear speedup) but not by much.

As you see from the diagram, using more than 5 slaves does not pay off much for both cases and using more than 10 slaves makes no sense in the commutative case, although there is still some progress observed if things do not commute.

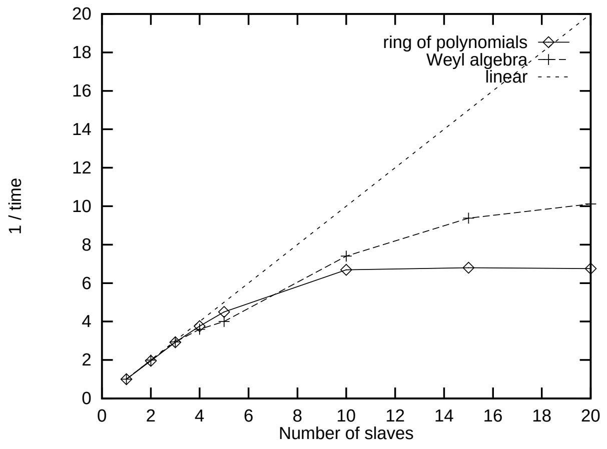

Figure 1. Speedups comparison for commutative and noncommutative setting

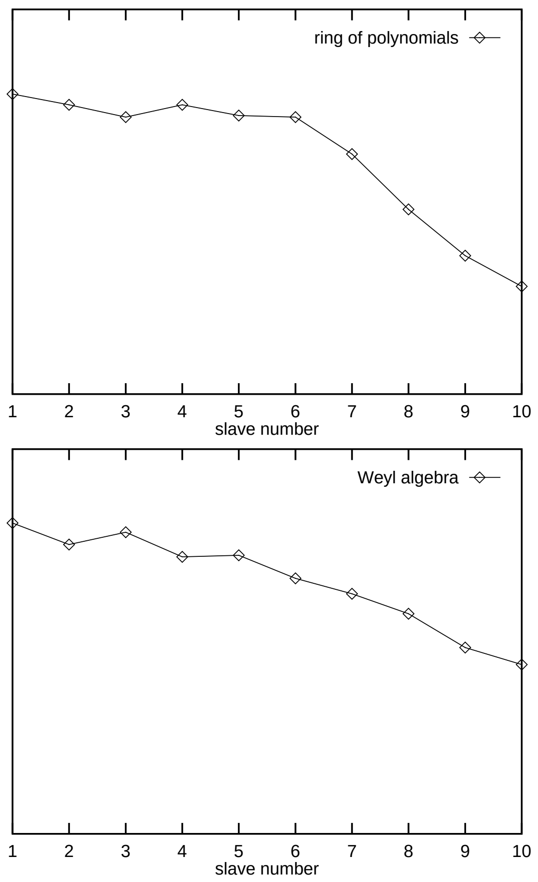

Figure 2. Load distribution for 10 slaves in commutative and noncommutative setting

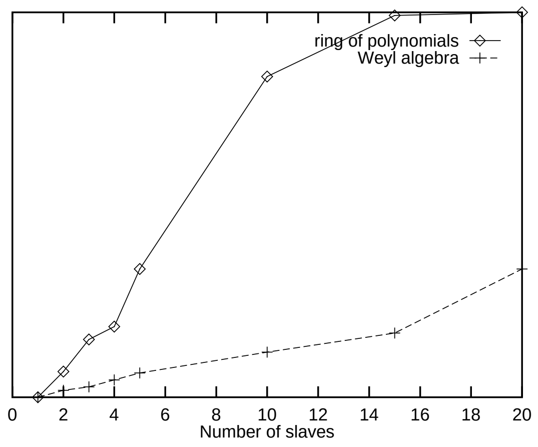

Figure 3. The number of times REDUCE task is sent

Figure 2 provides an explanation why this behavior occurs: we look at the distribution of **REDUCE** requests sent to slaves. When a decision on where to send a request is made, the algorithm chooses the idling slave with the least number. Therefore, the last slave probabilistically has a chance for more rest than the first. But how bad is the distribution of the load? Figure 2 shows that whenever we cease to get a speedup close to linear (we have chosen 10 slaves to support this point) it is quite far from uniform. On the other hand, the situation is slightly better in the noncommutative case.

Another factor that slows down the algorithm is the communication overhead. This can be roughly measured by the total number of times the **REDUCE** request is issued. For $n = 1$ this number is just the number of s-pairs reduced during the computation, however, for $n > 1$ by the time a slave finishes its job and sends the reduced s-polynomial back to the master the master's copy of the reduction basis might grow, making a further reduction possible; then the master proceeds by sending an "extra" **REDUCE**. Usually, the number of **REDUCE**s grows with the number of slaves used, though this growth is noticeably faster in the commutative setting as opposed to noncommutative (see Figure 3).

To summarize, we would like to say that, in general, the experimental results that we obtained confirm the results of previous research on this subject: linear and superlinear speedups are not possible, at least using the strategy preserving approach. However, using small number of CPUs for computing Gröbner bases in parallel is quite efficient. This has also strengthened our believe that the payoff is even bigger if the same technique is used in the noncommutative setting due to a better ratio of (communication overhead)/(time spent in reduction routine). This is also due to the lack of elaborate s-pair selection techniques in the noncommutative case.

References

[1] G. Attardi and C. Traverso. Strategy-accurate parallel Buchberger algorithms. *J. Symbolic Comput.*, 21(4-6):411–425, 1996.

[2] J. Briançon and P. Maisonobe. Remarques sur l'ideal de Bernstein associé à des polynomes. Preprint.

[3] S. Chakrabarti and K. Yelick. Distributed data structures and algorithms for Gröbner basis computation. *Lisp and Symbolic Computation*, 7:1–27, 1994.

[4] J. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of ISSAC 2002*, pages 75–83, 2002.

[5] R. Gebauer and H. M. Möller. On an installation of Buchberger's algorithm. *J. Symbolic Comput*, 6(2-3):275–286, 1988.

[6] A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. One sugar cube, please. In *Proc. ISSAC '91*, pages 49–54, 1991.

[7] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI.* MIT Press, 2nd edition, 1999.

[8] A. Kandri-Rody and V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. *J. Symbolic Computation*, 9:1–26, 1990.

[9] B. Reinhard, G. Manfred, and W. Küchlin. A fine-grained parallel completion procedure. In *ISSAC 1994*, pages 269–277, 1994.

[10] Y. Sato and A. Suzuki. Parallel computation of boolean Gröbner bases. In *Electronic Proceedings of ATCM 1999 (http://www.atcminc.com/mPublications/EP/EPATCM99)*, 1999.

[11] K. Siegl. A Parallel Factorization Tree Gröbner Basis Algorithm. Technical Report 94-51, RISC-Linz, Johannes Kepler University, Linz, Austria, 1994. Published in Proceedings, PASCO'94, World Scientific Publ. Comp.

[12] J. Ucha and F. Castro-Jiménez. Bernstein-Sato ideals associated to polynomials. *Journal of Symbolic Computation*, 37(5):629–639, 2004.