A parallel Buchberger algorithm for multigraded ideals

Emil Sköldberg¹, Mikael Vejdemo-Johansson², and Jason Dusek³

¹ School of Mathematics, National University of Ireland, Galway, Ireland, emil.skoldberg@nuigalway.ie
² Department of Mathematics, Stanford University, mik@math.stanford.edu
³ Noisebridge, jason.dusek@gmail.com

Abstract. We demonstrate a method to parallelize the computation of a Gröbner basis for a homogenous ideal in a multigraded polynomial ring. Our method uses anti-chains in the lattice $\mathbb{N}^k$ to separate mutually independent S-polynomials for reduction.

1 Introduction

In this paper we present a way of parallelizing the Buchberger algorithm for computing Gröbner bases in the special case of multihomogeneous ideals in the polynomial algebra over a field. We describe our algorithm as well as our implementation of it. We also present experimental results on the efficiency of our algorithm, using the ideal of commuting matrices as illustration.

1.1 Motivation

Most algorithms in commutative algebra and algebraic geometry at some stage involve computing a Gröbner basis for an ideal or module. This ubiquity together with the exponential complexity of the Buchberger algorithm for computing Gröbner bases of homogeneous ideals explains the large interest in improvements of the basic algorithm.

1.2 Prior Work

Several approaches have been tried in the literature. Some authors, such as Chakrabarti–Yelick [1] and Vidal [2] have constructed general algorithms for distributed memory and shared memory machines respectively. Reasonable speedups were achieved on small numbers of processors. Another approach has been using factorization of polynomials; all generated S-polynomials are factorized on a master node, and the reductions of its factors are carried out on the slave nodes. Work by Siegl [3], Bradford [4], Gräbe and Lassner [5]. In a paper by Leykin [6] a coarse grained parallelism was studied that was implemented both in the commutative and non-commutative case.
Good surveys of the various approaches can be found in papers by Mityunin and Pankratiev [7] and Amrhein, Bündgen and Küchlin [8]. Mityunin and Pankratiev also give a theoretical analysis of and improvements to algorithms known at that time.

1.3 Our approach

Our approach restricts the class of Gröbner bases treated to homogenous multigraded Gröbner bases. While certainly not general enough to handle all interesting cases, the multigraded case covers several interesting examples. For these ideals we describe a coarsely grained parallelization of the Buchberger algorithm with promising results.

Example 1. Set \( R = k[x_1, \ldots, x_{n^2}, y_1, \ldots, y_{n^2}] \) where \( k \) is a field. Let \( X \) and \( Y \) be square \( n \times n \)-matrices with entries the variables \( x_1, \ldots, x_{n^2} \) and \( y_1, \ldots, y_{n^2} \) respectively. Then the entries of the matrix

\[
I_n = XY - YX
\]

form \( n^2 \) polynomials generating the ideal \( I_n R \).

The computation of a Gröbner basis for \( I_1 \) and \( I_2 \) is trivial and may be carried out on a blackboard. A Gröbner basis for \( I_3 \) is a matter of a few minutes on most modern computer systems, and already the computation of a Gröbner basis for \( I_4 \) is expensive using the standard reverse lexicographic term order in \( R \); the Macaulay2 system [9] several hours are needed to obtain a Gröbner basis with 563 elements. However, using clever product orders, Hreinsdóttir has been able to find bases with 293 and 51 elements [10, 11]. As far as we are aware of, a Gröbner basis for \( I_5 \) is not known.

By assigning multidegrees \((1, 0)\) to all the variables \( x_1, \ldots, x_{n^2} \) and \((0, 1)\) to all the variables \( y_1, \ldots, y_{n^2} \), the ideal \( I_n \) becomes multigraded over \( \mathbb{N} \times \mathbb{N} \), and thus approachable with our methods.

Example 2. While this paper presents only the approach to multigraded ideals in a polynomial ring, an extension to free multigraded modules over multigraded rings is easily envisioned, and will be dealt with in later work.

Gröbner bases for such free modules would be instrumental in computing invariants from applied algebraic topology such as the rank invariant as well as more involved normal forms for higher dimensional persistence modules.[12]

2 Multigraded rings and the grading lattice

A polynomial ring \( R = k[x_1, \ldots, x_r] \) over a field \( k \) is said to be multigraded over \( P \) if each variable \( x_j \) carries a degree \( |x_j| \in P \) for some partially ordered monoid \( P \). We expect of the partial order on \( P \) that if \( p, q \in P \) then \( p \leq p + q \) and \( q \leq p + q \). The degree extends from variables to entire monomials by requiring \( |mn| = |m| * |n| \) for monomials \( m, n \); and from thence a multigrading of the
entire ring $R$ follows by decomposing $R = \bigoplus_{p \in P} R_p$ where $R_p$ is the set of all homogenous polynomials in $R$ of degree $p$, i.e. polynomials with all monomials of degree $p$. We will in this paper only work with rings graded in $\mathbb{N}^d$ with monoidal structure given by pointwise addition, and $p \leq q$ if each coefficient of $p$ is less than or equal to the corresponding coefficient of $q$; but our results hold for more cases than these. A homogenous polynomial of degree $(n_1, \ldots, n_d)$ is said to be of total degree $n_1 + \cdots + n_d$.

We write $\text{lm} p$, $\text{lt} p$, $\text{lc} p$ for the leading monomial, leading term and leading coefficient of $p$.

**Proposition 1.** Suppose $p$ and $q$ are homogenous. If $|p| \not\leq |q|$ then $\text{lm} p$ does not divide $\text{lm} q$.

**Proof.** If $\text{lm} p | \text{lm} q$ then $\text{lm} q = c \text{lm} p$, and thus $|\text{lm} q| = |c| \ast |\text{lm} p|$, and thus by our requirement for a partially ordered monoid, $\text{lm} p \leq \text{lm} q$.

**Proposition 2.** Reduction in the Buchberger algorithm of a given multidegree for a homogenous generating set depends only on its principal ideal in the partial order of degrees.

**Proof.** We recall that the reduction of a polynomial $p$ with respect to polynomials $q_1, \ldots, q_k$ is given by computing

$$p' = p - \frac{\gcd(\text{lm} p, \text{lm} q_j)}{\text{lm} p} q_j$$

for a polynomial $q_j$ such that $\text{lm} q_j | \text{lm} p$. We note that by Proposition 1, this implies $|\text{lm} q_j| \leq |\text{lm} p|$ and thus $|q_j| \leq |p|$.

We note that Proposition 2 implies that if two S-polynomials are incomparable to each other, then their reductions against a common generating set are completely independent of each other. Furthermore, since $|p'| = |p|$, in the notation of the proof of Proposition 2, a reduction of an incomparable S-polynomial can never have an effect on the future reductions of any given S-polynomial.

Hence, once S-polynomials have been generated, their actual reductions may be computed independently across antichains in the partial order of multidegrees, and each S-polynomial only has to be reduced against the part of the Gröbner basis that resides below it in degree.

## 3 Algorithms

The arguments from Section 2 lead us to an approach to parallelization in which we partition the S-polynomials generated by their degrees, pick out a minimal antichain, and generate one computational task for each degree in the antichain.

One good source for minimal antichains, that is guaranteed to produce an antichain, though most often will produce more tasks than are actually populated
by S-polynomials is to consider the minimal total degree for an unreduced S-polynomial, and produce as tasks the antichain of degrees with the same total degree.

Another, very slightly more computationally intense method is to take all minimal occupied degrees. These, too, form an antichain by minimality, and are guaranteed to only yield as many tasks as have content.

Either of these suggestions leads to a master-slave distributed algorithm as described in pseudocode in Algorithms 1 and 2.

We use the following primitives in our descriptions of the algorithms, these are implemented using the sage, mpi4py and sqlalchemy functionalities.

– spol(s1, s2): the set of all S-polynomials that can be generated from a pair of polynomials, the first from s1 and the second from s2.
– send(target, data): sends data to node target
– receive(source, data): receive data from node target
– store(data): store data in the SQL database.
– load(data): load data from the SQL database.
– reduce(s1, s2): fully reduce all polynomials in the set s1 by the polynomials in the set s2.

The resulting master node algorithm can be seen in Algorithm 1, and the simpler slave node algorithm in Algorithm 2.

And the simpler slave node algorithm.

4 Experiments

We have implemented the master-slave system described in Section 3 in Sage [13], using MPI for Python [14, 15] for distributive computing infrastructure and SQLAlchemy [16] for an abstraction of a common storage for serialized python objects.

In order to test the system, we have computed a Gröbner basis for I_3, and we have started computation of a Gröbner basis for I_4. We have run computations on a dedicated server, and measured processor load during the computation there.

While running the computation at full load, we used the tool sar to measure CPU utilization, logging processor load every 5 seconds to a text file for further analysis.

The CPU load for the first 11ksec wall clock time of the computation of the example I_4 is given in figure 1. Since the activities of the master process are almost exclusively while at least one other processor waits, we can expect a maximal utilization of 700% for this computation. The figure shows that the computer swiftly hits full utilization and stays at approximately full speed until the cutoff point of the graph. The measurements beyond our cutoff point are not as definitive as the ones before; we have therefore not included them in the plot at present.

Amazon EC2, 8 core and high memory virtual server
Algorithm 1 Master algorithm for a distributed Gröbner basis computation

loop
  degree ← next degree
  if we are not out of degrees then
    alldegs ← multidegs in degree
    loop
      if all processors are waiting then
        break
      end if
      if more work left in degree then
        send new degree to waiting slave
        continue
      end if
      receive message with tagged 'tag'
      if tag = request for new degree then
        if no more degrees then
          sync
          continue
        end if
        send new degree to requesting slave
      else if tag = new gb elements stored then
        update grobner basis
        compute new spolynomials
      end if
    end loop
  else
    if all slaves are waiting then
      finish up
    end if
    receive message from slave, tagged 'tag'
    if tag = new gb element stored then
      update grobner basis
      compute new spolynomials
    end if
  end if
end loop

Algorithm 2 Slave algorithm for a distributed Gröbner basis computation

loop
  send(master, get new degree)
  d ← receive(master, msg)
  if msg = finish then
    return
  end if
  if msg = new degree d then
    reduce all S-polynomials in degree d and append to Gröbner basis
    send(master, finished degree)
  end if
end loop
5 Conclusions and Future Work

In conclusion, we have demonstrated that while the parallel computation of
Gröbner bases in general is a problem haunted by the ghost of data dependency,
the lattice structure in an appropriate choice of multigrading will allow for easy
control of dependencies. Specifically, picking out antichains in the multigrading
lattice gives a demonstrable parallelizability, that saturates the kind of comput-
ing equipment that is easily accessible by researchers of today.

Furthermore, we have developed our methods publicly accessible,\(^5\) and re-
leased it under the very liberal BSD license. Hence, with the ease of access to
our code and to the Sage computing system, we try to set the barrier to build
further on our work as low as we possibly can.

There are many places to go from here. We are ourselves interested in inves-
tigating many avenues for the further application of the basic ideas presented
here:

– Multigraded free modules, and Gröbner bases of these; opening up for the use
of these methods in computational and applied topology, as a computational
back bone for multigraded persistence.
– Multigraded free resolutions; opening up for the application of these methods
in parallelizing computations in homological algebra.

\(^5\) http://code.google.com/p/dph-mg-gröbner, the code used for Section 4 can be
found in the sage subdirectory
– Adaptation to non-commutative cases; in particular to use for ideals in and modules over quiver algebras.
– Building on work by Dotsenko and Khoroshkin, and by Dotsenko and Vejdemo-Johansson, there is scope to apply this parallelization to the computation of Gröbner bases for operads. [17, 18]

References