The computation of Gröbner bases on a shared memory multiprocessor

Jean-Philippe Vidal
Computer Science Department
Carnegie Mellon University
PITTSBURGH, PA 15213
jpv@CS.CMU.EDU

Abstract

This article presents a system which computes Gröbner bases on a shared memory multiprocessor. The basic idea is that each processor picks an element in the set of unreduced critical pairs, reduces the S-polynomial associated with it and updates the basis and the set of pairs according to the result. The originality of this algorithm relies on the small amount of synchronization it requires among the processes. The details of an implementation on a 16 processors Encore machine are given together with results of tests performed with well-known examples of the literature.

1 Introduction

In his Ph.D. [4], Buchberger presented an algorithm which has been studied extensively in the recent years for at least two reasons. The first one is the large list of its applications, including the determination of the membership of a polynomial to an ideal and the solution of systems of algebraic equations. The second one is the apparent contradiction between the simplicity of the algorithm and the difficulty to obtain practical bounds on its complexity.

The definitions and facts that we present here can be found, for example, in Robbiano, [12]. K is a field and \( A = K[x_1, \ldots, x_n] \) is the ring of polynomials in \( x_1, \ldots, x_n \) over K. It is well-known from Hilbert that any ideal \( I \) of \( A \) has a finite basis \( B \) of polynomials.

An admissible order \( \succ \) on the set of power products of \( x_1, \ldots, x_n \) (a power product is a monomial with coefficient 1, like \( x_1^2x_2^3 \)) is a total order such that \( T \succ 1 \) for every power product \( T \) and such that \( T_1 \succ T_2 \) implies \( T_1T_3 \succ T_2T_3 \) for all power products \( T_1, T_2, T_3 \). If \( P \) is a polynomial, we call \( T(P) \) the leading power product of \( P \) with respect to \( \succ \) and \( C(P) \) the corresponding coefficient. The initial \( \text{In}(P) \) of \( P \) is the product \( C(P)T(P) \).

For a set of polynomials \( B \) and an admissible order \( \succ \), we can define a reduction relation \( \longrightarrow \). We say that \( P \longrightarrow Q \) if and only if there exists in \( B \) a polynomial \( R \) such that \( R \succ 1 \) for every power product \( T \) and such that \( T_1 \succ T_2 \) implies \( T_1T_3 \succ T_2T_3 \) for all power products \( T_1, T_2, T_3 \). If \( P \) is a polynomial, we call \( T(P) \) the leading power product of \( P \) with respect to \( \succ \) and \( C(P) \) the corresponding coefficient. The initial \( \text{In}(P) \) of \( P \) is the product \( C(P)T(P) \).

Buchberger proved that \( B \) is a Gröbner basis iff \( \text{Spol}(P, Q) \longrightarrow 0 \) for all pairs \( \{P, Q\} \) of polynomials of \( B \).

An algorithm can then be designed. Its two main variables are \texttt{Basis}, the set of polynomials, which is initially equal to the given basis of \( I \), and \texttt{Pairs}, the set of unreduced pairs, which is
initially equal to the set of pairs of elements of Basis. Now, the algorithm picks a pair in Pairs and reduces the corresponding S-polynomial w.r.t. \( > \) and Basis. If the resulting polynomial \( Q \) is not equal to 0, then it is added to Basis and all the pairs formed by \( Q \) and every other polynomial in Basis are added to Pairs. When, finally, Pairs is empty, Basis is a Gröbner basis of the ideal generated by the input set of polynomials.

This basic algorithm and two criteria used to avoid the reduction of certain pairs are described by Buchberger in [5]. We discuss here the design and implementation of a parallel version of this algorithm.

2 Description of the tools for synchronization

The following concepts appear in many places in the literature. Our source is the book “Operating Systems Principles” by Brinch Hansen, [3], pages 77-116.

A variable \( v \) of type \( T \) shared among several processes will be noted:

\[
\text{shared } T \ v
\]

Critical regions. Sometimes, a process executing a certain sequence of instructions using the variable \( v \) needs that this variable is not modified by another process during this execution. Therefore, at most one process should execute such a sequence involving \( v \) at a given time. The object enforcing this policy is called a critical region associated with \( v \). We will use the following notation:

\[
\text{region } v \ \text{do sequence}
\]

Waiting for a condition to hold. A process may need to wait for a shared variable \( v \) to have a certain property, i.e. for another process to modify \( v \). The process must test \( v \) inside a critical region but it must wait outside this critical region. Otherwise, another process would not be able to modify \( v \). To allow this, we introduce the primitive \( \text{await} \). When a process, in a critical region, executes the statement:

\[
\text{await } B(v),
\]

it evaluates the Boolean function \( B(v) \). If the result is true, it goes on. If it is false, it quits the critical region and joins the waiting queue for this condition. Naturally, when the condition becomes true, only one process of the waiting queue is readmitted in the critical region.

Readers-writers exclusion.

The concept of critical region is somehow limiting. We need a tool allowing an arbitrary number of processes to access \( v \) simultaneously, if they do not change \( v \). Such processes will be called readers and the processes updating \( v \) will be called writers. Of course, the same process can be a reader and a writer at different points of time. At any given time, at most one process should be a writer and, if there is such a writer, there should be no reader. Therefore, we introduce the three following notations, in which RWE stands for readers-writers exclusion:

\[
\text{RWE } T \ v
\]
\[
\text{reading.access.to } v \ \text{do sequence}
\]
\[
\text{writing.access.to } v \ \text{do sequence}
\]

The first one states that the variable \( v \) of type \( T \) is protected by the described mechanism. The other two are executed by a process which asks to enter a sequence of statements with reading or writing privileges on \( v \). A reading access is granted if there is no writer and no process waiting to become a writer. A writing access is granted if there is no reader. Therefore, we see that writers have priority over readers.