Parallel Gröbner Basis Computation in ||MAPLE||

Kurt Siegl
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University
A-4040 Linz, Austria

e-mail: siegl@risc.uni-linz.ac.at
phone: +43 7236 3231 66
fax: +43 7236 3338 30

May 11, 1992

Abstract

This paper presents a new system for parallel symbolic computation called ||MAPLE|| (speak: parallel Maple), in which logic programming provides parallelism and imperative programming provides efficiency. The system is built as a combination between the parallel declarative programming language Strand and the sequential computer algebra system Maple.

We describe a novel approach to the parallelization of Buchberger's Gröbner bases algorithm using a medium grain pipe-line principle for the polynomial reduction. Additionally we give a reformulation of the "optimization criteria" suitable for a distributed algorithm. The approach is implemented in ||MAPLE|| on a 17 Processor transputer distributed memory system and tested on a 20 Processor Sequent shared memory machine where it shows a remarkable speed-up.

Keywords: Parallel Gröbner Bases Computation, Parallel Computer Algebra Systems, Logic Programming.

Submitted to ISSAC '92.

1 Motivation

Gröbner bases computation and other algorithms that solve problems in symbolic computation are inherently time consuming. In practical applications a lot of problems occur that cannot be solved within a reasonable amount of time using todays computers.

In solving large problems, we face the following situation:

- With todays fast parallel computers we should be able to solve huge problems.
- Sequential computer algebra systems have big sequential libraries and fast implementations of the basic algorithms, but are not yet suitable for parallel computers.
- Parallel programming languages are well suited for parallel programming, but have no libraries for solving problems in symbolic computation.

Thus our goal should be to reuse the huge library in existing computer algebra systems and additionally get easy access to the power of todays parallel computers.

To achieve this goal we combine the computer algebra system Maple with the parallel programming language Strand. The result is a parallel programming system with the full functionality of Maple. This method allows easy writing and porting of parallel algorithms.

As a big advantage we managed to keep ||MAPLE|| absolutely portable. A user may develop and test parallel algorithms on his personal SUN workstation and execute it on a parallel supercomputer where the same program runs faster. Our algorithm has been developed on a 16-transputer distributed memory parallel machine and tested on a Network of SUN workstations as well as on a 20-processor Sequent shared memory machine where it shows a high speed-up for suitable large examples.

2 The ||MAPLE|| System

Several attempts at parallelizing Maple are not widely accepted by the Maple programming community.

In [Watt, 1986] the UNIX primitives are used for parallelism. The programmer has to develop his parallel algorithm on a very low level. An improvement was done in [Char, 1990] where Linda is used. Linda is more hardware independent, but programmers still have to worry about creation of parallel processes and communication between them.

These methods of writing parallel programs are difficult to use and to debug. Therefore they are not acceptable for programmers who want to develop a large amount of code.

Our System is easy to use and absolutely portable. ||MAPLE|| programs may run on different hardware without any modification or recompilation. All necessary communication is done automatically by the system without any additional programming effort.

Since ||MAPLE|| uses implicit parallelism, which allows writing parallel programs without any expert knowledge in parallel programming, it should be acceptable in the Maple programmer world.

The ||MAPLE|| system can be viewed as an extension of the parallel declarative programming language Strand in the sense that the top level program structure is Strand from where arbitrary Maple functions or sequences of Maple statements can be called to perform the sequential tasks.

We distinguish between two layers. The logic (top) layer provides parallelism and the imperative layer provides efficiency. The top layer is Strand which controls the parallel execution of an algorithm. For performing sequential tasks we have to call the underlying Maple system.

This method combines the advantages of Strand as well as of Maple.

Strand is a programming language of the guarded Horn clause type.

That means that the basic construct is a clause of the form:

$$H :- G_1, ..., G_n | B_1, ..., B_m. \qquad (n, m \geq 0)$$

where the bodies B_j are executed in parallel as soon as all the guards G_i are fulfilled.

In Strand pattern matching is used instead of unification, and assignments are used to describe output bindings to increase the performance.

Data dependencies can be expressed by Data-flow synchronization and stream communication.

Additionally Strand allows the use of arbitrary sequential subroutines, written in C or Fortran, as guards or body calls.

Maple is a well known computer algebra system with one of the biggest libraries available.

Maple has a Kernel written in C which can be linked as a subroutine to C programs. Input and output may be handled via strings representing Maple expressions.

For parallelizing Maple with Strand we developed an interface consisting of one new Strand guard and two bodies.

- A *m_test* guard that takes a boolean Maple function and succeeds if the function evaluates to true and fails otherwise.
- A *maple* body call takes an arbitrary Maple function and its arguments and returns the result computed by Maple.
- An *inline* operation allows the inclusion of a sequence of arbitrary Maple statements. Additionally we may specify which variables should be used as input and which ones are output variables.

Communication between Strand and Maple is done via Maple strings. A Maple string is a Maple expression in legible form and may be manipulated by using the Strand string facilities.

This method allows easy porting and developing parallel algorithms used in symbolic computation.

Since we can use the Maple guard operation for solving arbitrary constraints in the guard, ||MAPLE|| may also be used as a constraint logic programming language.

3 Gröbner Bases Computation

The Gröbner bases algorithm invented by [Buchberger, 1965] is one of the most important and time intensive algorithms in symbolic computation. It is used for solving many problems in polynomial ideal theory.

We assume that the reader is familiar with the basic notions of Gröbner bases theory and the sequential algorithm. For an introduction and the notation see [Buchberger, 1985]. This paper presents the main method and results for the ||MAPLE|| implementation. The interrested reader may found more details and all proofs in [Siegl, 1990].

The reader should have some basic knowledge in parallel logic programming to understand the given programming aspects. For an introduction to Strand we refer to [Foster and Taylor, 1989]

3.1 Basic Concepts

Our parallel implementation of the Gröbner bases algorithm uses the general scheme of pipelining.

This may be used in the case where we have to test or modify the input with respect to a set of properties. After the test (modify) part we may (like in the well known Concurrent Prolog implementation of the Sieve of Eratosthenes [Shapiro, 1985]) append some of the tested values to our set in order to use it for testing the rest of the input.

We distinguish between two basic types of methods using this scheme.

3.1.1 Onedirectional Pipe-line

This method may be used for solving a problem were we test a given input set with respect to a test set with fixed elements. For parallelizing this method, we put every element of the test set to one node in a pipe. The input values are driven subsequently through the pipe. Each intermediate process takes the first element from its previous neighbour and tests, modifies or cancels it. The result is sent to the next process. Then it waits for the next input for computing the result. Whenever a value reaches the end of the pipe-line it is possible to extend the test set by simply creating a new process holding the new value at the end of the pipe. Thus, the future input values will be tested with respect to the new value, too.

Examples where we may use this method is a pipeline of operations and the Sieve of Erathostenes to compute prime numbers.

3.1.2 Bidirectional Pipe-line for Backward Communication

In some cases the result of a test has to be used for generating new input values. Typical examples are the Critical/Pair-Completion procedures. Here a set of critical pairs is built and tested with respect to a given set of properties. Pairs that are not cancelled are appended to the test set, and new critical pairs are built.

The parallelization is similar to the previous method. We only have to send the computed values back through the pipe. In each intermediate pipe process, we generate new input values and send them back to the front end. The front end process takes the new values comming back from the pipe and send them through the pipe again.

Examples of this kind of use are the Knuth-Bendix procedure and the Gröbner bases algorithm.

3.2 Main Algorithm

According to the basic concept of pipelining we have in Gröbner bases computation processes reducing the input polynomials with respect to a basis. The irreducible input values are appended to the reduction processes. For handling the computation of S-polynomials, the new polynomials joining the basis are sent back through the pipe building the S-polynomials with the earlier computed basis polynomials. The process at the front of the pipe only needs to collect the computed S-polynomials and sends them to the reduction processes again.

As a result we get a bidirectional pipe forming a dynamically expandable ring. The organization of the pipe to compute the Gröbner basis is shown in (Figure 1):

Our proposed algorithm has the following features:

- The individual polynomials of the (intermediate) basis are distributed among the nodes of the (distributed) parallel machine arranged in a pipe.
- An S-polynomial is driven through the pipe and each node reduces the polynomial by applying its basis polynomial.
- If the (partial) reduction does not yield zero, the polynomial is put into the basis (and placed on a new node).
- "On its way back", the new polynomial is combined with the existing polynomials in the basis for forming the new S-polynomials that are subsequently driven through the pipe. We would like to note that in the process of forming the new S-polynomials we apply the "criteria" for omitting unnecessary reductions.

```
-mode main(InPoly?,Basis^).
main(InPoly,Basis):-    % Main loop of the parallel Gröbner basis algorithm
    append_spol(InPoly,Spol,Testpoly),          % Front end
    sieve(Testpoly,Spol,Basis)@fwd.             % Back end

-mode sieve(Test-polys?,S-polys^,Basis-polys^).
sieve([BP|TPs],[sp(BP,[])|SPs1],[BP|BPs]):-
    BP =\= '0' |                                % New basis polynomial found
      filter(BP,TPs,SPs,RPs,SPs1),              % New intermediate process
    sieve(RPs,SPs,BPs)@fwd                      % Extend pipe by one
sieve(['0'|TPs],SPs,B):-                        % Zero reduction
    sieve(TPs,SPs,B).
```

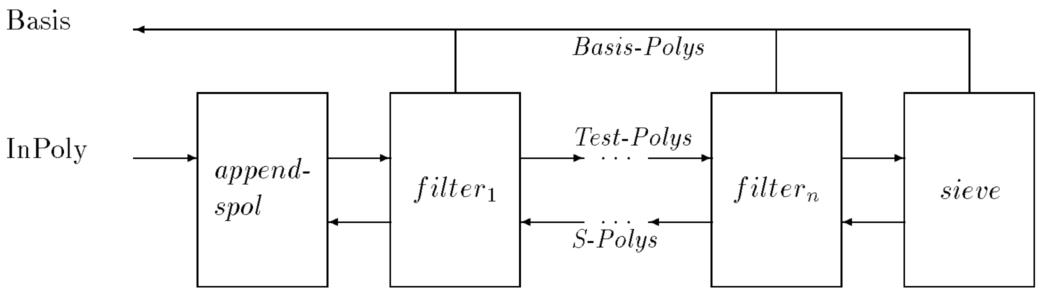

Figure 1: Organization of the Pipe for Computing Gröbner Bases

- Also on its way back the new polynomial reduces the existing polynomials in the basis.

This leads to various advantages, that are crucial to achieve a high performance on a parallel machine:

- The basis is distributed over various nodes in a multiprocessor system.
- All optimization strategies are used.
- All the "criteria" are applied locally in the filter processes without any search.
- The communication effort is minimized.

4 Criteria

In [Buchberger, 1985] Buchberger introduced two criteria for detecting unnecessary S-polynomials. The intensive use will lead to a crucial improvement of the performance.

A parallel algorithm has to compete with the best sequential ones, therefore we have to include all optimizations of a sequential version to the parallel one, too.

Here we describe how we implement all of them locally in the distributed pipe-processes without any search and minimal communication.

4.1 Criterion on leading power product

The first criterion is based on the following theorem [Buchberger, 1965, Buchberger, 1970]:

Theorem 4.1 (Criterion on LP)
If $LP(f_1) * LP(f_2) = lcm(LP(f_1), LP(f_2))$, then S-polynomial$(f_1, f_2)$ may always be reduced to zero modulo F and may be cancelled.

This criterion may always be tested locally in every pipe process that generates new S-polynomials.

4.2 Chain Criterion

The second criterion is originally formulated as:

Theorem 4.2 (Chain Criterion)
If ChainCriterion$(f_1, f_2, Basis)$ then S-polynomial(f_1, f_2) may be cancelled.

ChainCriterion(f_1,f_2,Basis):$\Leftrightarrow$
there exists a $p \in Basis$ such that
$f_1 \neq p \neq f_2$,
lcm(LP(f_1), LP(f_2)) is a multiple of LP(p),
S-polynomial(f_1,p) and S-polynomial(p,f_2) has been chosen for reduction before.

For applying this criterion in a distributive way, we have to split it into two parts.

From now on we assume that the polynomials f in the basis (pipe) are numbered in descendent order at creation time. The newest polynomial has the highest number and is called f_n.

$$\boxed{f_1} \longleftarrow \ldots \longleftarrow \boxed{f_i} \longleftarrow \ldots \longleftarrow \boxed{f_j} \longleftarrow \ldots \longleftarrow \boxed{f_n}$$

Lemma 4.1 (Chain Criterion – Pipe Implementation) *For all i,j,n* $(1 \leq i < j < n)$,

1. *if Criterion2(j,n) at i then S-polynomial*(f_j, f_n) *may be deleted,*
2. *if Criterion3(i,n) then S-polynomial*(f_i, f_n) *may be deleted.*

Criterion2(j,n) at i $:\Leftrightarrow lcm(LP(f_j), LP(f_n))$ *is a multiple of* $LP(f_i)$.

Criterion3(i,n) $:\Leftrightarrow$

- *there exists a* j_1 $(i < j_1 < n)$ *such that* $lcm(LP(f_i), LP(f_n))$ *is a multiple of* $LP(f_{j_1})$,
- *there exists no* j_2 $(i < j_2 < n)$ *with Criterion2*(j_2, n) *at i.*

To proof the correctness we have to show that whenever we cancel a pair in the pipe implementation, we may also cancel it with the chain criterion. The proof is an easy case checking where we know that S-polynomial(f_i, f_j) has been chosen for reduction before.

For applying this criterion locally on each pipe position (i), we only need to know all the S-polynomials computed with f_n driven back through the pipe to the front and the leading power products of the basis polynomials that come later.

All this information may be packed in one message that will be updated in each pipe process. When we transfer such an package through the pipe to the front end we will call it "round". At the front end a "round" consists of the set of all S-polynomials computed with the new basis polynomial f_n.

4.3 Theoretical Aspects

In the case where the leading power product of a basis polynomial (f_1) is a multiple of the leading power product of another one (f_2) a criteria in [Buchberger, 1970] says that we may cancel the basis polynomial (f_1) after computing the *S-polynomial*(f_1, f_2).

With the pipe implementation it is easy to show that this cancellation will be done as a special case of the chain criterion where only the *S-polynomial*(f_1, f_2) will survive the criteria.

In our algorithm we use this fact for further optimization and interreductions of the basis.

This fact also shows that if we use only a partial normal form algorithm we will not create to much S-polynomials. A not fully reduced polynomial will be automatically sent again through the reduction pipe-line until it is fully reduced.

5 Further Optimization

5.1 Computing of S-polynomials

The actual computation of an S-polynomial will be done at the front end process. In the intermediate pipe positions we only have to deal with leading powerproducts.

Since the front process has only access to all S-polynomials of one "round" (previous ones are in the reduction process later ones not yet known), sorting is only done within a "round".

Practical experiences shows that at most three S-polynomials has to be computed in one "round". The average case is one, therefore normally sorting will not pay off.

5.2 Basis Reduction and Sorting

A fully reduced and sorted basis is crucial to the efficiency of the Gröbner bases algorithm. Since this steps are very communication intensive for a distributed algorithm. For reducing the communication overhead we use a sorting on demand scheme.

Because of the fact that we use a partial normal form algorithm it may happen that an S-polynomial may not be reduced to its normal form and will be appended as new basis polynomial. This stems from the fact that a basis polynomial (f_i) may again reduce the S-polynomial and should be ordered to a later pipe step.

By applying the criteria we detect that fact at the pipe position i. By our criteria we know that after the computation of the new S-polynomial with f_i the new basis polynomial may be cancelled. Thus, we replace the new basis polynomial by f_i and have a new ordering. This reordering is time independent and does not require any syncronisation between concurrent steps.

For reducing the basis, we simply kill a reduceable basis polynomial and send the reduced polynomial as the only new S-polynomial through the pipe again.

5.3 Parallel Partitioning

The most time consuming part in the distributed filter process are the reduction steps, that are basically independent, we use a cluster of four processors for each filter process.

A manager process is used for the criteria and controlling the work of three worker processes. Each of them performs one reduction step at the time.

With this technique the S-polynomials may overtake each other during the reduction phase and the faster ones will join the basis earlier. This speculative aspect might result in an superlinear speed-up.

6 Implementation Issues

In the ||MAPLE|| implementation all control information are written in the Strand layer. The basic arithmetic like one reduction step, as well as the application of the criteria for each node and "round" are written as independent Maple programs.

We give some results, how our parallel algorithm behaves in a parallel environment. All timings are given in seconds of execution time, including loading time and garbage collection.

The algorithm has been developed on a 17 transputer distributed memory machine. We give the timing statistics of the unmodified program running on a 20 processor Sequent shared memory machine under normal load, where we achived up to a linear speed-up even with small examples. The current implementation has some limitations to the maximal internode message size, which did not allow us to test bigger examples. We hope to be able to change this limitation soon.

Example	Order	||MAPLE|| on			Speed-up
		1 Proc.	4 Proc.	16 Proc.	achieved
Trinks2	tdeg	125	53	28	4.5
Robbiano	tdeg	156	73	31	5.0
Katsura3	plex	147	38	11	13.4
Katsura3	tdeg	196	51	14	14.0
Trinks2	plex	272	62	16	17.0

7 Related Work

One of the first published method (proposed by [Watt, 1986]) divides the process into alternating phases of S-polynomial computation and reduction of the basis, using parallelism in each stage. But Watt's implementation of the idea was incorrect. Two corrected versions of Watt's version of the algorithm are presented in [Ponder, 1988] but does not show any speed-up.

In fact Buchberger himself sketched a parallel version of his algorithm in an early paper [Buchberger, 1987] using an central manager. An implementation of this method using an Shared memory machine was presented in [Vidal, 1989] showing that it is possible to have a speed-up between 2 and 10 on a 16 processor shared memory machine with standard test examples listed in the literature.

An stream based method presented by [Hawley, 1991] using an pipeline for reductions was specially designed for the purpose of solving a continuous flow of constraints.

8 Conclusion

We have developed a system which may be used to parallelize algorithms in symbolic computation in a simple and elegant way. An implementation of the Gröbner Basis algorithm shows how to use the system for developing parallel algorithms.

The parallelization of the Gröbner bases algorithm is shown as in the frame of the general scheme of pipelining one the level of reduction of polynomials. Additionally we give a formulation of Buchbergers criteria suitable for parallel algorithms showing that high optimization is not contradictory to parallelism.

In tests on a shared memory machine we achieved a linear speed-up with suitable large examples. Our non-deterministic reduction process shows also a superliner speed-up with larger computations.

References

[Buchberger, 1965] Bruno Buchberger.
Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-dimensional Polynomial Ideal) (In German).
PhD thesis, University of Innsbruck, Department of Mathematics, Austria, 1965.

[Buchberger, 1970] Bruno Buchberger.
Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems (An Algorithmical Criterion for the Solveability of Algebraic Systems of Equations) (In German).
Aequationes mathematicae, 4(3):374–383, 1970.
Also: Technical Report 70-1, RISC-Linz, Johannes Kepler University, Linz, Austria, 1970.

[Buchberger, 1985] Bruno Buchberger.
Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory.
In N. K. Bose and D. Reidel, editors, *Recent trends in Multidimensional Systems*, chapter 6, pages 184–232. D. Reidel Publishing Company, Dordrecht-Boston-Lancaster, 1985.

[Buchberger, 1987] Bruno Buchberger.
The Parallelization of Critical-Pair/Completion Procedures on the L-Machine.
In *Japanese Symposium on Functional Programming*, pages 54–61, February 1987.
Also: Technical Report 87-12, RISC-Linz, Johannes Kepler University, Linz, Austria, 1987.

[Char, 1990] Bruce W. Char.
Progress report on a system for general-purpose parallel symbolic algebraic computation.
In *Proceedings of the ISSAC'90, Tokyo, Japan, August 20–24*, pages 96–103, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, 1990. ACM Press, New York.

[Foster and Taylor, 1989] Ian Foster and Stephen Taylor.
Strand — New Concepts in Parallel Programming.
Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[Hawley, 1991] David Hawley.
A Buchberger Algorithm for Distributed Memory Multi-Processors based on Distributed Sorting.
In *Proceedings of the First International Conference of the Austrian Center for Parallel Computation*, Salzburg, Austria, September 30 – October 3, 1991.
Lecture Notes in Computer Science, Springer, Berlin.

[Ponder, 1988] C. G. Ponder.
Evaluation of "Performance Enhancements" in Algebraic Manipulation Systems.
Technical Report UCB/CSD 88/438, Computer Science Division (EECS), University of California, August 1988.

[Shapiro, 1985] Ehud Shapiro.
Systolic Programming — A Paradigm of Parallel Processing.
Technical Report CS84-16, Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel, January 1985.

[Siegl, 1990] Kurt Siegl.
Gröbner Bases Computation in STRAND: A Case Study for Concurrent Symbolic Computation in Logic Programming Languages.
Master's thesis, Johannes Kepler University, Linz, Austria, 1990.
Also: Technical Report 90-54, RISC-Linz, Johannes Kepler University, Linz, Austria, 1990.

[Vidal, 1989] Jean-Philippe Vidal.
The Computation of Gröbner Bases on a Shared Memory Multiprocessor.
Technical report, School of Computer Science, Carnegie-Mellon University, 1989.

[Watt, 1986] Stephen Michael Watt.
Bounded Parallelism in Computer Algebra.
Research Report CS-86-12, Faculty of Mathematics, University of Waterloo, Ontario, Canada, 1986.