Proceedings of PASCO'94

A Parallel Factorization Tree Gröbner Basis Algorithm*

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

e-mail: Kurt.Siegl@risc.uni-linz.ac.at
phone: +43 7236 3231 62
fax: +43 7236 3231 30

Abstract: The idea using polynomial factorization for speeding up the computation of Buchberger's Gröbner bases algorithm for the purpose of polynomial equation solving leads to major improvements in the computation time. In this paper we show how one may introduce factorization within a parallel Gröbner basis algorithm, without unnecessary doubling parts of the work. A reformulation of the sequential optimization criteria for avoiding unnecessary computation is given, to fit the needs of a parallel version.

The approach has been implemented in ||MAPLE|| (speak: parallel Maple), a computer algebra system, in which logic programming provides parallelism and imperative programming provides efficiency. In first experiments with a prototype implementation, we managed to solve examples within a few minutes on a couple of SGI workstations, which can not be solved with a conventional, sequential implementation.

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

1 Introduction

The Gröbner basis algorithm is one of the most important and time intensive algorithm for equation solving. But with todays fast parallel computers we should be able to solve huge problems. Thus our goal should be to adapted the Gröbner basis algorithm in a way that we get access to the power of todays parallel computers.

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

In fact Buchberger himself sketched a parallel version of his algorithm in an early paper [Buchberger, 1987] using a central manager. An implementation of this method using a 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.

A completely different approach was used in [Siegl, 1990]. Here a medium grain pipe-line principle is used for the polynomial reduction. All the individual polynomials of the basis are placed on separate nodes, arranged in a pipeline. Due to a reformulation of the "optimization criteria" suitable for a distributed algorithm the author ma-

*Supported by: Austrian Science Foundation (FWF) Proj. Parallel Symbolic Computation, No S5302-PHY, Proj. POSSO, No. 9181-TEC (ESPRIT BRA No. 6846)

naged to get up to a linear speed-up on a 16 processor distributive memory machine.

In the frame of Japans 5th generation project some contributions to the parallelization are given. Following the previous idea, a stream based method presented by [Hawley, 1991] using a pipeline for reductions was specially designed for the purpose of solving a continuous flow of constraints has been proposed. Based on this work [Sawada *et al.*, 1992] developed a parallel constraint solver.

[Davenport, 1987] suggested to factor polynomials during the computation. This results in an independent sub-problem for each factor. In fact the result of that algorithm is not a Gröbner basis any more, but enough for the purpose of equation solving. Following this idea [Bradford, 1990] developed a parallel algorithm, but he found out that the performance improvement gained was very limited. Due to the independent computation of the sub-problems a lot of individual reductions has to be computed twice.

2 The Pipelining Gröbner Bases Computation

In our earlier work [Siegl, 1990] we proposed a parallel implementation of the Gröbner bases algorithm using a pipeline principle for polynomial reduction. Because of the fact that the pipeline method serves as a basis for the proposed tree algorithm (Section 3), we will give a sketch of the overall method. Details may be found in [Siegl, 1990].

In a pipeline algorithm 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, 1987]) append some of the tested values to our set in order to use it for testing the rest of the input. In some cases the result of a test has to be used for generating new input values. Typical examples are all 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 canceled are appended to the test set and new critical pairs are built.

For parallelizing this method, we assign each element of the test set to one parallel process connected in a pipe. The input values are driven subsequently through the pipe. Each intermediate process takes the first element from its previous neighbor 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.

If necessary, we have to send the computed values back through the pipe for generating new input. 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 coming back from the pipe and sends them through the pipe again.

```
-mode main(InPoly?,Basis^).
main(InPoly,Basis):-                        % Main loop
    append_spol(InPoly,Spol,Testpoly),      % Top 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 poly
      filter(BP,TPs,SPs,RPs,SPs1),
      sieve(RPs,SPs,BPs)@fwd                % Extend pipe
sieve(['0'|TPs],SPs,B):-                    % Zero reduction
    sieve(TPs,SPs,B).
```

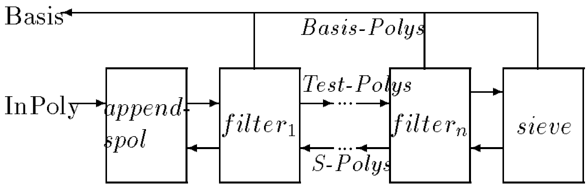

Figure 1: Organization of the Pipe for Computing Gröbner Bases

The sequential Gröbner bases algorithm is a Critical Pair Completion algorithm. Given a set of input polynomials as initial basis, one computes critical polynomials and reduces them with respect to the basis polynomials. The resulting polynomials are added to the basis and new critical pairs are formed.

In the parallel version we have a pipeline of processes reducing the input polynomials with respect to an increasing set of basis polynomials. The irreducible input values are appended to the reduction processes. For generating new critical polynomials (called S-polynomials) as input, 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):

In more detail the main procedure generates the top end (append_spol) and the back end (sieve) process of the pipe. The top end process collects the newly generated S-polynomials (Spol) from the pipe, appends them to the initial input polynomials (InPoly) and sends the whole to the pipe again. The intermediate (filter) processes take each polynomial, reduces it with respect to its basis polynomial and returns the result to the next pipe process. As soon the back end receives a non zero polynomial it extends the pipe by a new intermediate (filter) process and sends that new basis polynomial (BP) back through the pipe for generating new S-polynomials. Polynomials reducing to zero are canceled out. As soon there will be no further polynomials flowing in the pipe the algorithm terminates.

3 From the Pipe to a Tree

While experimenting with the parallel pipeline method we found out that distributing every basis polynomial to a separate processor introduces too much overhead, so a grouping of basis polynomials should be done. In particular this is important if we move from dedicated parallel computers with fast intercommunication networks towards networks of Workstations, where the communication time is rather high. Additionally it is crucial for the efficiency of a parallel algorithm, that the polynomials which can be reduced faster should enter the basis as soon as possible and should be used for creating further S-polynomials first. On the other hand [Davenport, 1987] showed that factorization of polynomials may result in a faster algorithm even in the sequential case.

Thus our goal will be to combine the advantages of the pipeline algorithm with the factorization method. As long as no polynomials will factor we will reduce them with the conventional methods, to cover the grouping demand. This part will be internally parallelized using the manager - worker approach like in [Vidal, 1989] to guarantee that faster reducible polynomials will show up earlier in the basis. But as soon a polynomial may be factored we add a new pipeline process for each factor. As result we get a tree structure (Figure 2) of parallel processes, where each branch can be seen as a grouped reduction pipeline.

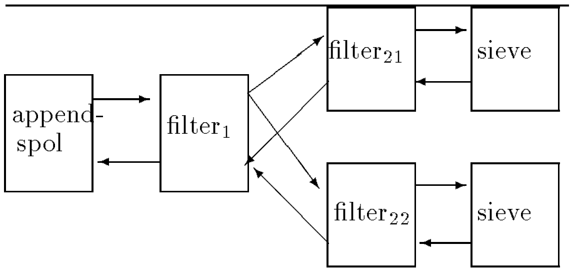

Figure 2: Splitting up a pipe into a factorization tree

To illustrate the method on an example we will manually compute the factored lexicographic $(x > y)$ Gröbner Basis of the polynomial set:

$$F := \{x^2 * y - 1, x * y^2 + 1\}$$

Our algorithm starts with an empty basis and the set F as initial set of S-polynomials. The first polynomial ($f_1 := x^2 * y - 1$) can not be reduced with respect to the still empty basis and will be added to the basis. Since we now only have a single polynomial in the basis, we can not build new S-polynomials yet.

The next S-polynomial ($f_2 := x * y^2 + 1$) is still irreducible with respect to the basis ($\{f_1\}$) and will be added to the basis as well. As new S-polynomial we get $f_3 := x + y$ which must be added to the basis as well and it leads to two new S-polynomials ($f_4 := y^3 - 1$, $f_5 := x * y^2 + 1$).

f_4 will not change after reducing by the basis ($\{f_1, f_2, f_3\}$). But factorization will lead to two new polynomials ($\{f_{4,1} := y - 1, f_{4,2} := y^2 + y + 1\}$). Now we have to split up the pipe into a tree. The first part ($B_1 := \{f_1, f_2, f_3\}$) remains and we get now two sub pipes ($B_{2,1} := \{f_{4,1}\}$,$B_{2,2} := \{f_{4,2}\}$). The remaining S-polynomial f_5 will be reduced by the first part of the tree already and need not be splitted up for the second parts. All new S-polynomials generated by $\{f_{4,1}, f_{4,2}\}$ will be reduced to zero. In fact we will detect that fact already by applying the criteria for omitting unnecessary S-polynomials (Section 4).

As result we get two factored Gröbner bases ($B_1 \cup B_{2,1}$, $B_1 \cup B_{2,2}$). By removing all polynomials whose leading power product is a multiple of an other basis polynomial and complete interreduction of the remaining ones we get the sets:

$$\{\{x + 1, y - 1\}, \{x + y, y^2 + y + 1\}\}$$

With this method we managed to combine all advantages of the currently known techniques for parallelizing the Gröbner basis algorithm.

We are able to maximize the possible parallelism gained by factorization and need not compute parts of the reduction twice as it is done in [Bradford, 1990]. Due to the additional, internal parallelization of each tree node with the manager - worker aproach, we may easily solve the problem of load balancing by overloading a certain number of processors on the border line with tasks of two nodes.

Additionally all optimization criteria can be implemented in a distributive fashion.

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- (tree-) 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-poly(f_1,p) and S-poly(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.

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

Lemma 4.1 (Chain Criterion – Pipe Version)
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).

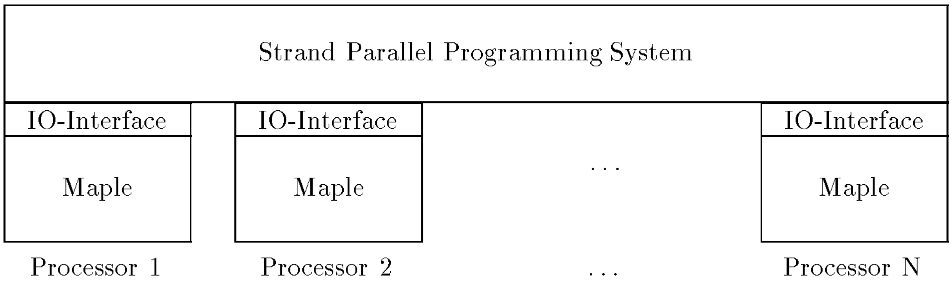

Figure 3: Structure of the ||MAPLE|| system

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.

Interreduction of the Basis would introduce a bottleneck in the parallel program, therefore we may use this fact and simply remove basis polynomials instead of reducing it. In the tree algorithm one has to guaranty, that there is a polynomial which may reduce the basis polynomial in every possible sub-pipe. Anyway, practical examples showed that it is faster if the basis polynomials will not be reduced at all.

Additionally, this result also shows that if we use only a partial normal form algorithm, as we do, we will not create too many S-polynomials. A not fully reduced polynomial will be automatically sent again through the reduction pipe-line until it is fully reduced, without introducing additional S-polynomials.

5 Implementation Issues

All our algorithms have been implemented in ||MAPLE|| [Siegl, 1993] which is a portable system for parallel symbolic computation. The core of the system is built on top of an interface between the parallel declarative programming language Strand [Foster and Taylor, 1989] and the sequential computer algebra system Maple [Char *et al.*, 1983], thus providing the elegance of Strand and the power of the existing sequential algorithms in Maple.

||MAPLE|| programs may run on different hardware ranging from shared memory machines over distributed memory architectures up to networks of Workstations without any modification or recompilation. All necessary communication is done automatically by the system without any additional programming effort. Since ||MAPLE|| uses implicit parallelism, it allows writing parallel programs without any expert knowledge in parallel programming.

The ||MAPLE|| system has two layers (Figure 3). The top layer is the parallel declarative programming language Strand which controls the parallel execution of an algorithm. For performing sequential tasks we may call arbitrary Maple functions or sequences of Maple statements in the underlying Maple system over some interface routines. The result is a parallel programming system with the full functionality of Maple and parallel power of Strand.

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 procedures.

Here we give some results of a prototype implementation, which show the behavior of our factorization tree algorithm in a parallel environment. All timings are given in seconds of execution time, including time for loading and garbage collection. Due to the fact that the execution time (in particular on the single processor version) varies widely, dependent on the actual chosen order of the polynomials, we give all timings as an average of several test series.

The algorithm has been developed on a 20 processor Sequent shared memory machine. Since the absolute performance of that machine is rather poor, we give the timing statistics of the unmodified program running on a Network of 10 SGI Workstations, where we could easily solve examples which could not be solved with any other algorithm before.

To have a fair comparison with a state of the Art implementation, we use as sequential reference implementation the built in Gröbner Basis algorithm of Maple V Release 3, which uses the same math engine as our ||MAPLE|| system. Examples marked with "*" did not terminate after one night (Figure 4).

Example	MapleV	||MAPLE|| on		Speed-up
		1 Proc.	10 Proc.	
Challenge	*	225	28	8.0
Arnborg5	*	543	104	5.2
Pavele4	496	90	34	2.5
Trinks2	6	26	12	2.2

Figure 4: Experimental results

From the table one may see that if an example factors well (Challenge, Arnborg5) one may gain an enormous increase of the performance. In case where we can not factor (Pavele4, Trinks2) at all the performance is in the same order of magnitude as the conventional algorithm. Unfortunately trying to factor polynomials which are irreducible introduces an additional overhead.

Because of the fact that the tested problems have been rather small the high latency of the workstation network limits the achievable speedup. In fact most of the speedup gained using several workstations has a non deterministic origin. It turns out that polynomials which can be reduced faster (i.e. smaller ones) tend to be more important for further reductions than slower ones.

6 Conclusion

Using factorization for speeding up Gröbner basis computation, for the purpose of equation solving gives us a major improvement in the computation time of a certain kind of examples. We have shown that it is possible to avoid unnecessary doubling of certain intermediate reductions even in a parallel environment.

The parallelization of the Gröbner bases algorithm is shown as an extension of a pipeline structure towards to a tree structure for reduction of polynomials. Additionally we gave a formulation of Buchbergers criteria suitable for parallel algorithms showing that high optimization is not contradictory to parallelism.

In tests on a network of workstations we managed to solve examples, which can not be solved with a conventional algorithm, within a few seconds.

A List of Examples

Challenge: Gröbner Basis challenge posted by Richard J. Fateman

$[S^3+2*R^3+2*Q^3+2*P^3, S^5+2*R^5+2*Q^5+2*P^5, -S^5+(R+Q+P)*S^4+(R^2+(2*Q+2*P)*R+Q^2+2*P*Q+P^2)*S^3+(R^3+Q^3+P^3)*S^2+(3*R^4+(2*Q+2*P)*R^3+(4*Q^3+4*P^3)*R+3*Q^4+2*P*Q^3+4*P^3*Q+3*P^4)*S+(4*Q+4*P)*R^4+(2*Q^2+4*P*Q+2*P^2)*R^3+(4*Q^3+4*P^3)*R^2+(6*Q^4+4*P*Q^3+8*P^3*Q+6*P^4)*R+4*P*Q^4+2*P^2*Q^3+4*P^3*Q^2+6*P^4*Q]$

Arnborg5: also known as Cyclic 5 roots

$[x+y+z+t+u, x*y+y*z+z*t+t*u+u*x, x*y*z+y*z*t+z*t*u+t*u*x+u*x*y, x*y*z*t+y*z*t*u+z*t*u*x+t*u*x*y+u*x*y*z, x*y*z*t*u+1]$

Pavele4: $[x, y, z, t, a, b, c, d]$

$[x*y+x*z+x*t-a, y*x+y*z+y*t-b, z*x+z*y+z*t-c, t*x+t*y+t*z-d]$

Trinks2: $[s, t, z, p, w, b]$

$[10000*b^2+6600*b+2673, 45*p+35*s-165*b-36, 35*p+40*z+25*t-27*s, 15*w+25*p*s+30*z-18*t-165*b^2, -9*w+15*p*t+20*z*s, w*p+2*z*t-11*b^3, 99*w-11*s*b+3*b^2]$

References

[Bradford, 1990] Russell Bradford. A Parallelization of the Buchberger Algorithm. In *Proceedings of the ISSAC'90, Tokyo, Japan, August 20–24*, pages 296–298, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, 1990. ACM Press, New York.

[Buchberger, 1965] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringesnach 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 *et al.*, 1983] Bruce W. Char, Keith O. Geddes, W. Morven Gentleman, and Gaston H. Gonnet. The Design of Maple: A Compact, Portable and Powerful Comuter Algebra System. In *EUROCAL '83, Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 101–115. Springer, Berlin, 1983.

[Davenport, 1987] J.H. Davenport. Looking at a Set of Equations. Technical Report 87-06, Computer Science Department, Bath, UK, 1987.

[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.

[Sawada *et al.*, 1992] Hiroyuki Sawada, Satoshi Terasaki, and Akira Aiba. The Parallel Algebraic Constraint Solver of GDCC. Draft, Institute for New Generation Computer Technology, 1992.

[Shapiro, 1987] Ehud Shapiro, editor. *Concurrent Prolog, Collected Papers, Volumes 1 and 2.* MIT Press, Cambridge, Massachusetts, 1987.

[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.

[Siegl, 1993] Kurt Siegl. Parallelizing Algorithms for Symbolic Computation Using ||MAPLE||. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 179–186, San Diego, California, May 19-22, 1993. ACM Press. Also: Technical Report 93-08, RISC-Linz, Johannes Kepler University, Linz, Austria, February 1993.

[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.