

10-1-1994

Tree Structured Grobner Basis Computation on Parallel Machines

Hemal V. Shah

Purdue University School of Electrical Engineering

Jose A. B. Fortes

Purdue University School of Electrical Engineering

Follow this and additional works at: <http://docs.lib.purdue.edu/ecetr>

Shah, Hemal V. and Fortes, Jose A. B., "Tree Structured Grobner Basis Computation on Parallel Machines" (1994). *ECE Technical Reports*. Paper 199.

<http://docs.lib.purdue.edu/ecetr/199>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

TREE STRUCTURED GRÖBNER BASIS
COMPUTATION ON PARALLEL
MACHINES

HEMAL V. SHAH
JOSE A. B. FORTES

TR-EE 94-30
OCTOBER 1994



SCHOOL OF ELECTRICAL ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Tree Structured Grobner Basis Computation on Parallel Machines

Hemal V. Shah and Jose A. B. Fortes
Department of Electrical Engineering
Purdue University
West Lafayette, In 47907, USA
hvs@ecn.purdue.edu
fortes@ecn.purdue.edu

Abstract: With the advent of symbolic mathematical software packages such as Maple, Mathematica, and Macsyma, symbolic computation has become widely used in many scientific applications. Though a significant effort has been put in performing numeric computation on multiprocessors, symbolic computation on parallel machines is still in an unexplored state. However, symbolic mathematical applications are ideal candidates for parallel processing, because they are computationally intensive. This paper considers the parallel computation of Grobner basis, a special basis for a multivariate polynomial ideal over a field that plays a key role in symbolic computation. Large Gröbner basis computation poses a challenging problem due to its dynamic data dependent behavior and resource-intensiveness. In an attempt to meet this challenge, a new tree structured approach for Gröbner basis computation in parallel is proposed in this paper. It constructs the Grobner basis of a set of polynomials from Grobner basis of its subsets. The tree structured approach proposed in this paper lends itself to parallel implementation and significantly reduces the computation time of large Grobner basis. Finally, experimental results illustrating the effectiveness of the new approach are provided.

Keywords: Symbolic computation, Grobner basis, mathematical software, numeric computation; parallel machines, SIMD, MIMD.

1 Introduction

In spite of the fact that most scientific computation has been numeric in nature, the availability of mathematical software packages like Maple, Mathematica, Macsyma, etc., has enabled and popularized the use of symbolic manipulation in many applications. Problem areas that require the usage of symbolic computation or mixed computation include geometric modeling, geometric theorem proving, robotics and control. Grobner basis, a special multivariate polynomial basis, is a very important concept in symbolic computation. Generally, construction of a Grobner basis is time-consuming and resource intensive, because of both the need of exact arithmetic and the possibility of generating and analyzing many polynomials. It has been shown that its worst-case complexity is doubly exponential. [Hof90]. In this paper, a tree structured approach for performing Grobner basis computation is proposed. Using this approach, large polynomial computations can be performed more efficiently in an uniprocessor and tree-based parallel implementations are possible. The parallel algorithm presented in this paper is based on divide-and-conquer strategy.

Several researchers have addressed the issue of parallel execution of a Gröbner basis algorithm. In his paper on `MAPLE`, Siegl [Sie93] described a way of performing parallel symbolic computation using parallel declarative programming language Strand and the sequential computer algebra system Maple. Ponder [Pon89, Ponb, Pona] proposed two parallel algorithms for Grobner basis computation (Parallel S-poly and Parallel reduction). In his views, both algorithms were "less parallel" and slow in convergence. Inferring from his conclusions, exploring other ways to parallelize Gröbner basis computation need to be considered. For parallel Grobner basis computation, Schreiner and Hong [SH93b, SH93a] showed that by invoking optimized routines of PACLIB, a system for parallel algebraic computation on shared memory computers, a maximum speedup of 10 could be achieved on a 20 processor Sequent Symmetry (a MIMD computer with shared memory) for Grobner basis computation. Senechaud [Sen89, Sen90, Sen91] computed Grobner basis by computing Grobner basis of the subsets of a set of polynomials and combining them. The algorithm presented in this paper takes a similar approach. Vidal [Vid90] proposed algorithms for Grobner basis computation on shared memory multiprocessor using synchronization between processes. In his approach, each processor reduced S-polynomial of an unreduced pair in parallel. Depending on the result of the reduction, the processor updated the basis and the set of pairs.

Despite previous research on parallel computation of Grobner basis, there is a room for improvement. In this paper, a variant of Buchberger's algorithm for Grobner basis computation is presented. The new algorithm is easy to visualize as a tree structured computation and provides more insights into parallel Grobner basis computation. In the next section, basic concepts and definitions related to polynomials and Grobner basis computation are briefly reviewed. In Section 3, a parallel algorithm for Grobner basis computation is described and its correctness is proven. Finally, some results and conclusions are provided in Section 4.

2 Polynomials and Grobner Basis

2.1 Definitions

In this subsection, basic concepts related to polynomials and Grobner basis computation are presented for later use in the paper. An example illustrating the use of definitions is given before subsection 2.2. The definitions are quite similar to those of [BW93, CLO93].

Definition 2.1 (Monomial) A *monomial* in x_1, \dots, x_n is a product of the form $x_1^{\alpha_1} \dots x_n^{\alpha_n}$, where all exponents $\alpha_1, \dots, \alpha_n$ are nonnegative integers. The total degree of the monomial is the sum $\alpha_1 + \dots + \alpha_n$. The following notation is used hereon:

$$\begin{aligned}\alpha &= (\alpha_1, \dots, \alpha_n), \\ x^\alpha &= x_1^{\alpha_1} \dots x_n^{\alpha_n}, \\ |\alpha| &= \alpha_1 + \dots + \alpha_n,\end{aligned}$$

For example, let $n = 2$; if $\mathbf{x} = (x_1, x_2)$, $\mathbf{a} = (2, 1)$, then $x^\alpha = x_1^2 x_2$.

Definition 2.2 (Polynomial) A *polynomial* f in x_1, \dots, x_n with coefficients in field k is a finite linear combination of monomials. A polynomial f is written as

$$f = \sum_{\alpha} a_{\alpha} x^{\alpha}, a_{\alpha} \in k,$$

where the sum is over a finite number of n -tuples $\mathbf{a} = (\alpha_1, \dots, \alpha_n)$. The set of all polynomials in x_1, \dots, x_n with coefficients in k is denoted by $k[x_1, \dots, x_n]$.

A simple example of the last definition is $f = 2x_1^2 x_2 - x_1 + x_2 - 1$.

Definition 2.3 (Ideals) A subset $I \subset k[x_1, \dots, x_n]$ is an *ideal* if it contains 0 and it is closed under polynomial addition and polynomial multiplication, i.e.

1. $0 \in I$,
2. If $f, g \in I$, then $f + g \in I$,
3. If $f \in I$ and $h \in k[x_1, \dots, x_n]$, then $hf \in I$.

Definition 2.4 (Monomial Ordering) A *monomial ordering* on $k[x_1, \dots, x_n]$ is any relation $>$ on $Z_{\geq 0}^n$, or equivalently, any relation on the set of monomials x^α , $\mathbf{a} \in Z_{\geq 0}^n$, satisfying:

1. $>$ is a total (or linear) ordering on $Z_{>0}^n$,

2. If $\alpha > \beta$ and $\gamma \in \mathbb{Z}_{>0}^n$, then $\alpha + \gamma > \beta + \gamma$,
3. $>$ is a well-ordering on $\mathbb{Z}_{\geq 0}^n$. This means that every nonempty subset of $\mathbb{Z}_{\geq 0}^n$ has a smallest element; under $>$.

Lexicographical ordering which is used in the remaining paper is defined next.

Definition 2.5 (Lexicographical Ordering) Let $\alpha, \beta \in \mathbb{Z}_{\geq 0}^n$. The exponent vector α is *lexicographically* greater than β written as $\alpha >_{lex} \beta$, iff, in $\alpha - \beta$, the leftmost nonzero entry is positive. Similarly, $x^\alpha >_{lex} x^\beta$, if $\alpha >_{lex} \beta$. For example, if $x = (x_1, x_2)$, then $x_1^2 x_2 >_{lex} 1$, because $(2, 1) >_{lex} (0, 0)$.

Other monomial ordering used in symbolic computations are Inverse *lexicographical*, Graded *lexicographical*, and Graded reverse *lexicographical*.

Definition 2.6 (Coefficient, Term, Multidegree, Leading Coefficient., Leading Monomial, Leading Term) Let

$$f = \sum_{\alpha} a_{\alpha} x^{\alpha} \text{ be a polynomial in } k[x_1, \dots, x_n].$$

1. a_{α} is the *coefficient* of the monomial x^{α} ,
2. If $a_{\alpha} \neq 0$, then $a_{\alpha} x^{\alpha}$ is a term of f ,
3. The multidegree of f is $\text{multideg}(f) = \max(\alpha \in \mathbb{Z}_{>0}^n : a_{\alpha} \neq 0)$ with respect to lexicographical ordering,
4. The *leading coefficient* of f is $\text{LC}(f) = a_{\text{multideg}(f)} \in k$,
5. The *leading monomial* of f is $\text{LM}(f) = x^{\text{multideg}(f)}$,
6. The *leading term* of f is $\text{LT}(f) = \text{LC}(f) \text{LM}(f)$.

Definition 2.7 (S-polynomial) The S-polynomial of two polynomials

$$f = \sum_{\alpha} a_{\alpha} x^{\alpha} \text{ and } g = \sum_{\beta} a_{\beta} x^{\beta} \text{ is}$$

$$S_{poly}(f, g) = \frac{\text{LC}(g)x^{\gamma}f}{\text{LM}(f)} - \frac{\text{LC}(f)x^{\gamma}g}{\text{LM}(g)}$$

where $x^{\gamma} = \text{LCM}(\text{LM}(f), \text{LM}(g))$ is the least common multiple of $\text{LM}(f)$ and $\text{LM}(g)$.

Definitions 2.6 and 2.7 are now illustrated for two polynomials, $f_1 = 2x_1^2x_2 - 1$ and $f_2 = x_1x_2^2 - x_1$ with $x_1 >_{lex} x_2$. The multidegrees of f_1 and f_2 are $multideg(f_1) = (2, 1)$, $multideg(f_2) = (1, 2)$, their leading coefficients are $LC(f_1) = 2$, $LC(f_2) = 1$, their leading monomials are $LM(f_1) = x_1^2x_2$, $LM(f_2) = x_1x_2^2$, their leading terms are $LT(f_1) = 2x_1^2x_2$, $LT(f_2) = x_1x_2^2$, the least common multiple of the leading monomials is, $LCM(LM(f_1), LM(f_2)) = LCM(x_1^2x_2, x_1x_2^2) = x_1^2x_2^2$. The S-polynomial of f_1 and f_2 is

$$Spoly(f_1, f_2) = \frac{1 \cdot x_1^2x_2^2 \cdot (2x_1^2x_2 - 1)}{x_1^2x_2} - \frac{2 \cdot x_1^2x_2^2 \cdot (x_1x_2^2 - x_1)}{x_1x_2^2} = 2x_1^2 - x_2.$$

In the next subsection, a polynomial division algorithm that aids in finding a normal form of a polynomial with respect to a set of polynomials is provided. Using this normal form algorithm, a set of polynomials can be reduced.

2.2 Normal Form Algorithm

In this subsection, a normal form algorithm which will be used later in Gröbner basis computation is discussed. A normal form algorithm computes the fully reduced form of a polynomial with respect to a set of polynomials F . A normal form of a polynomial p with respect to a set of polynomials F is denoted by $NF(p, F)$. The algorithm as described in [Hof90] is as follows:

Algorithm 2.1 (Normal Form)

Input: A set F of polynomials, and a polynomial p .

Output: A normal form $NF(p, F)$ of p with respect to F .

Algorithm:

1. Set $p_0 = p$ and $i = 0$.
2. For $i = 0, 1, 2, \dots$ repeat step 3 until p_i cannot be rewritten; then output p_i and stop.
3. If there is a polynomial f in F such that the leading monomial of f divides a term $a_\alpha x^\alpha$ in p_i , then rewrite p_i as $p_{i+1} = p_i - \frac{a_\alpha x^\alpha f}{LT(f)}$.

Using the normal form algorithm, a set of polynomials F can be reduced on itself by replacing each polynomial $f \in F$ with its normal form with respect to set $F - \{f\}$. The reduction algorithm [BW93] is described as below:

Algorithm 2.2 (Reduce)

Input: A set F of polynomials.

Output: A reduced set $F' = Reduce(F)$ of polynomials.

```

begin
   $F' \leftarrow F$ 
  While there is  $p \in F'$  which is reducible on  $F' - \{p\}$  do
    Select  $p$  from  $F'$ 
     $F' \leftarrow F' - \{p\}$ 
     $h \leftarrow NF(p, F')$ 
    if  $h \neq 0$  then
       $F' \leftarrow F' \cup \{h\}$ 
    end
  end
   $F' \leftarrow \{f/HCF(f) | f \in F'\}$ 
end

```

To demonstrate the functionality of these algorithms, an example is presented next. Let $F = \{x_1^2 - x_2, -x_1 + x_2^2\}$, with $x_1 >_{lex} x_2$. The reduced set of F is computed by algorithm Reduce as follows.

Initially, $F' = \{x_1^2 - x_2, -x_1 + x_2^2\}$, then $p = x_1^2 - x_2$ is selected, leading to $F' = \{-x_1 + x_2^2\}$. Now, the computation of $NF(p, F')$ is carried out using algorithm 2.1. Initially, $p_0 = p = x_1^2 - x_2$. After executing step 3 of algorithm 2.1, $p_1 = x_1^2 - x_2 + (-x_1 + x_2^2)x_2 = x_1x_2^2 - x_2$, then, $p_2 = (x_1x_2^2 - x_2) + (-x_1 + x_2^2)x_2^2 = x_2^4 - x_2$. Therefore, $h = NF(p, F') = x_2^4 - x_2$, and algorithm Reduce updates F' to $F' = \{x_2^4 - x_2, -x_1 + x_2^2\}$. Continuing the execution of algorithm Reduce, $p = -x_1 + x_2^2$, is selected leading to $F' = \{x_2^4 - x_2\}$, $NF(p, F') = -x_1 + x_2^2$. So, the final reduced set is,

$$F' = \{x_2^4 - x_2, x_1 - x_2^2\}.$$

The definitions and reduction algorithms presented in the last two subsections are used in Grobner basis computation next.

2.3 Gröbner Basis Computation

In this subsection, the definition of Grobner basis and Buchberger's algorithm to compute it are provided.

Definition 2.8 Let F be a set of polynomials. Then a basis G for $Id(F)$, ideal generated by F , is a Grobner basis iff

1. for all pairs $(g_i, g_j), i \neq j$, the remainder of the division of $Spoly(g_i, g_j)$ by G is zero, in other words $NF(Spoly(g_i, g_j), G) = 0$, and
2. the ideals generated by F and G are identical.

Definition 2.9 A minimal Grobner basis for a polynomial ideal I is a Grobner basis G for I such that

1. $LC(p) := 1$ for all $p \in G$ and
2. for all $p \in G$; $LT(p) \ni Id(LT(G - \{p\}))$.

Definition 2.10 A reduced Grobner basis for a polynomial ideal I is a Gribner basis G for I such that

1. $LC(p) := 1$ for all $p \in G$ and
2. for all $p \in G$, no monomial of p lies in $Id(LT(G - \{p\}))$.

A well-known algorithm for the computation of Gröbner basis of a given set of polynomials is due to Buchberger in [Buc85a]. A reduced Grobner basis can be obtained by running the algorithm *Reduce* on the Grobner basis computed by Buchberger's algorithm.

Algorithm 2.3 (Grobner)

Input: $F = \{f_1, \dots, f_s\}$.

Output: A Gröbner basis $G = \{g_1, \dots, g_t\} = Gröbner(F)$; such that $Id(F) = Id(G)$.

begin

```

 $G \leftarrow F$ 
 $B \leftarrow \{(g_i, g_j) | g_i, g_j \in G \text{ with } i \neq j\}$ 
While  $B \neq \emptyset$  do
    Select  $(g_i, g_j)$  from  $B$ 
     $B \leftarrow B - \{(g_i, g_j)\}$ 
     $h \leftarrow Spoly(g_i, g_j)$ 
     $h_0 \leftarrow NF(h, G)$ 
    if  $h_0 \neq 0$  then
         $B \leftarrow B \cup \{(g, h_0) | g \in G\}$ 
         $G \leftarrow G \cup \{h_0\}$ 
    end
end

```

end

end

The following is an illustration of the execution of the above algorithm. Let F be a set of two-variate polynomials

$$F = \{x_1^2 - x_2, -x_1 + x_2^2\}.$$

Initially, $G = \{x_1^2 - x_2, -x_1 + x_2^2\}$, $B = \{(g_1, g_2)\}$. A pair (g_1, g_2) is selected by algorithm *Gröbner* leading to $h = Spoly(g_1, g_2) = x_1x_2^2 - x_2$, $h_0 = NF(h, G) = x_2^4 - x_2 \neq 0$. The algorithm Grobner then updates G and B to $G = \{x_1^2 - x_2, -x_1 + x_2^2, x_2^4 - x_2\}$, $B = \{(g_1, g_3), (g_2, g_3)\}$. Furthermore, $NF(Spoly(g_1, g_3), G) = 0$ and $NF(Spoly(g_2, g_3), G) = 0$. So, the algorithm Grobner terminates with the Grobner basis

$$G = \{x_1^2 - x_2, -x_1 + x_2^2, x_2^4 - x_2\}.$$

And, the reduced Grobner basis computed by algorithm *Reduce* is

$$G = \{x_1 - x_2^2, x_2^4 - x_2\}.$$

Though Buchberger's algorithm is a vital step in symbolic computation, it can be resource-intensive and time-consuming for large polynomial computations. Furthermore, it is difficult to parallelize. A new parallel algorithm to overcome these drawbacks is provided in the next section.

3 Parallelization of Grobner Basis computation

In the previous section, the Grobner basis computation details were provided. Due to its dynamic input dependent behavior, Buchberger's algorithm is difficult to parallelize efficiently. In this section a new variant of Buchberger's algorithm is presented which can be inferred from [Sen90]. The main idea behind the alternative approach is to compute Grobner basis of a given set from the Grobner basis of its subsets. The parallel algorithm presented next assumes availability of 2^p processors.

A New Parallel Algorithm for Gröbner Basis computation

Algorithm 3.1 (Par-Gröbner-Tree)

Input: $F = \{f_1, \dots, f_s\}$ given in 2^p subsets F_1, \dots, F_{2^p} such that $F = \bigcup_{i=1}^{2^p} F_i$.

Output: A reduced Grobner basis $G = \{g_1, \dots, g_t\} = Par_Gröbner_Tree(F)$; such that $Id(G) = Id(F)$.

begin

for $i = 1$ to 2^p **pardo**

$G_i = Gröbner_Tree(F_i)$

$G_i = Reduce(G_i)$

end

for $i = 1$ to p **do**

```

    for j = 1 to 2p step 2i pardo
      Gj = Gröbner_Combine(Gj, Gj+2i-1)
    end
  end
  G = G1
end

```

Algorithm 3.1 invokes other algorithms called *Grobner-Combine* and *Grobner-Tree* described as below. *Grobner-Combine* is almost the same as Buchberger's algorithm. But, here pairs of the same set are not considered because they are from the same Grobner basis set, thus they possess the first property of the definition 2.8. On a similar note, *Grobner-Tree* is the algorithm which sequentially computes the Grobner basis of a given set in a tree structure.

Algorithm 3.2 (Grobner-Combine)

Input: Two reduced Grobner basis sets G_1, G_2 .

Output: A reduced Grobner basis $G = \text{Gröbner_Combine}(G_1, G_2)$; such that $\text{Id}(G) = \text{Id}(G_1 \cup G_2)$.

```

begin
  G ← G1 ∪ G2
  B ← {(gi, gj) | gi ∈ G1, gj ∈ G2, criteria(gi, gj) ≠ 0}
  While B ≠ ∅ do
    Select a pair (gi, gj) from B
    B ← B - {(gi, gj)}
    h ← Spoly(gi, gj)
    h0 ← NF(h, G)
    if h0 ≠ 0 then
      B ← B ∪ {(g, h0) | g ∈ G, criteria(h0, g) ≠ 0}
      G ← G ∪ {h0}
    end
  end
  G = reduce(G)
end

```

Algorithm 3.3 (Gröbner_Tree)

Input: A set of polynomials $F = \{f_1, \dots, f_s\}$, where $s = 2^m$.

Output: A reduced Grobner basis $G = \{g_1, \dots, g_t\} = \text{Gröbner_Tree}(F)$; such that $\text{Id}(G) = \text{Id}(F)$.

```

begin
  for i = 1 to |F| do

```

```

     $G_i = \{f_i\}$ 
  end
  for i = 1 to log2 |F| do
    for j = 1 to |F| step 2i do
       $G_j = \text{Gröbner\_Combine}(G_j, G_{j+2^{i-1}})$ 
    end
  end
end

```

Here, $\text{criteria}(f, g)$ [Buc79] is the check which returns zero if $\text{NF}(\text{Spoly}(f, g), G)$ is zero. This is very useful in computation, because by detecting this criteria, one will avoid an entry to B and then deletion of it from B later on. From implementation point of view, this results in an improvement to overall performance of the algorithm. Next, the correctness of Par-Grobner-Tree is discussed.

3.1 Correctness of the New Parallel Algorithm for Gröbner Basis Computation

In order to prove the correctness of Par-Grobner-Tree, the following two theorems are presented.

Theorem 3.1 The basis G generated from basis G_1, G_2 by algorithm Grobner-Combine, is the reduced Gröbner basis of ideal generated by set F , where $F = F_1 \cup F_2$, G_1 and G_2 are reduced Grobner basis of sets F_1 and F_2 respectively.

Proof: Here G is the reduced basis generated from two sets G_1 and G_2 , so

1. $\text{Id}(G) = \text{Id}(G_1 \cup G_2) = \text{Id}(F_1 \cup F_2) = \text{Id}(F)$,
2. from algorithm Grobner-Combine it is evident that for all pairs $i \neq j$, $\text{NF}(\text{Spoly}(g_i, g_j), G) = 0$, where $g_i, g_j \in G$.

Therefore, basis G generated by the algorithm Grobner-Combine is the reduced Grobner basis of ideal generated by set F .

The tree structured computation for the algorithm Par-Grobner-Tree is shown in figure 1. The relationship between the algorithms Grobner and Grobner-combine is as below:

$$\text{Reduce}(\text{Gröbner}(F)) = \text{Gröbner_Combine}(\text{Reduce}(\text{Gröbner}(F_1)) \cup \text{Reduce}(\text{Gröbner}(F_2))), \quad (1)$$

where $F = F_1 \cup F_2$.

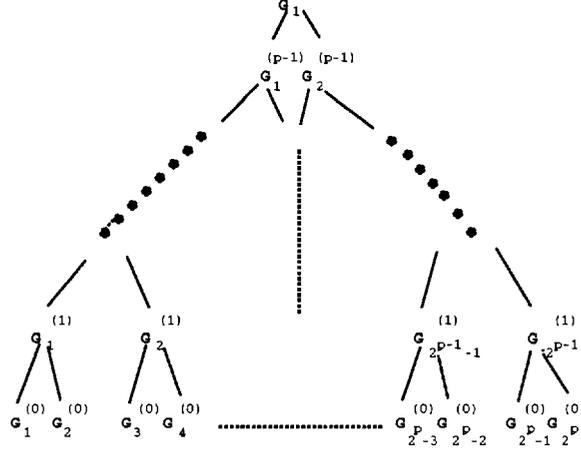


Figure 1: Grobner Basis Computation Viewed as a Tree Computation

Theorem 3.2 In the tree structured computation of the algorithm *Par_Gröbner_Tree*, G is the reduced Gröbner basis of ideal generated by set F .

Proof: In this proof, the associativity of union and Grobner-Combine, and the relationship between Grobner and Grobner-Combine algorithms are used.

$$G = \text{Par_Gröbner_Tree}(F) = G_1^{(p)} \quad (2)$$

$$= \text{Gröbner_Tree}(G_1^{(p-1)} \cup G_2^{(p-1)}) \quad (3)$$

$$= \text{Gröbner_Tree}(\text{Gröbner_Tree}(G_1^{(p-2)} \cup G_2^{(p-2)}) \cup \text{Gröbner_Tree}(G_3^{(p-2)} \cup G_4^{(p-2)})) \quad (4)$$

$$= \text{Gröbner_Tree}((G_1^{(p-2)} \cup G_2^{(p-2)}) \cup (G_3^{(p-2)} \cup G_4^{(p-2)})) \quad (5)$$

$$= \text{Gröbner_Tree}(\dots (G_1^{(0)} \cup G_2^{(0)}) \cup \dots \cup (G_{2^{p-1}}^{(0)} \cup G_{2^p}^{(0)}) \dots) \quad (6)$$

$$= \text{Gröbner_Tree}\left(\bigcup_{i=1}^{2^p} G_i^{(0)}\right) \quad (7)$$

$$= \text{Gröbner_Tree}\left(\bigcup_{i=1}^{2^p} \text{Reduce}(\text{Gröbner}(F_i^{(0)}))\right) \quad (8)$$

$$= \text{Reduce}(\text{Gröbner}\left(\bigcup_{i=1}^{2^p} F_i^{(0)}\right)) \text{ (from equation(1))} \quad (9)$$

$$= \text{Reduce}(\text{Gröbner}(F)) \quad (10)$$

This proves the correctness of the algorithm Par-Grobner-Tree. Furthermore, in Grobner-Tree, a reduced Gröbner basis of the union of two sets is always computed. So, by executing algorithm

Par_Gröbner_Tree, the reduced Grobner basis of a set of polynomials is obtained. The tree structured computation gives the best performance when the initial set on each node is itself a Grobner basis. In that case, $Par_Gröbner_Tree(F) = F$. But, in the worst case, the time complexity of *Par_Gröbner_Tree* can be doubly exponential same as that of the Buchberger's algorithm. The complexity measure used here is based upon the maximum total degree of the resulting polynomials during the computation. The effectiveness of the new algorithm is demonstrated by the results provided in the following section.

4 Results and Conclusions

4.1 Results

The results provided in this section are based on experiments performed on Intel Paragon, a distributed memory parallel machine. Two different types of polynomial equations are considered. The *lexicographical* term ordering is used to order the terms of polynomials. The first type of set of polynomials arises from the equations of n cylinders in n -dimensional space. The form of the polynomial equations is as described below:

$$F = \{f_1, \dots, f_n\}, \text{ where}$$

$$f_i = \left(\prod_{\substack{1 \leq j \leq n \\ j \neq i}} x_j^2 \right) - 1, \text{ for } 1 \leq i \leq n \text{ and } x_1 >_{lex} x_2 >_{lex} \dots >_{lex} x_n.$$

The second type of set of polynomials consists of the following equations,

$$F = \{f_1, \dots, f_n\}, \text{ where}$$

$$f_{i-1} = -x_{i-1}^2 + 2x_i^2 - x_{i+1}^2 \text{ for } 2 \leq i \leq n-1, \\ f_{n-1} = 2x_{n-1}^2 - x_n^2, f_n = x_n^2 - 1,$$

$$\text{and } x_1 >_{lex} x_2 >_{lex} \dots >_{lex} x_n.$$

Figure 2 shows the execution times of both the tree structured algorithm *Gröbner_Tree* and Buchberger's algorithm for both types of polynomials. It can be ascertained that, for the same set of polynomials, the computation time for executing Buchberger's algorithm grows exponentially faster than the time required to compute the Grobner basis using the tree algorithm (*Gröbner_Tree*). This suggests that for a large set of polynomials, Grobner basis can be sequentially computed more efficiently and faster by using tree structured algorithm *Grobner_Tree* than

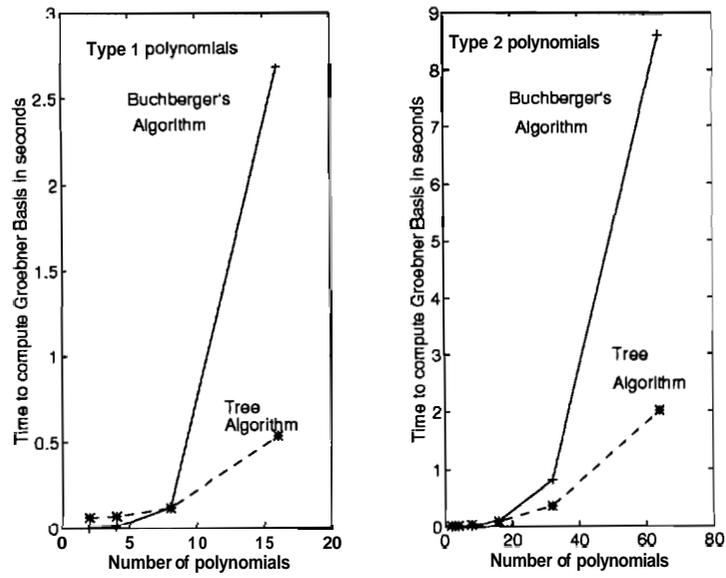


Figure 2: Comparison of *Groebner_Tree* algorithm with Buchberger's Algorithm

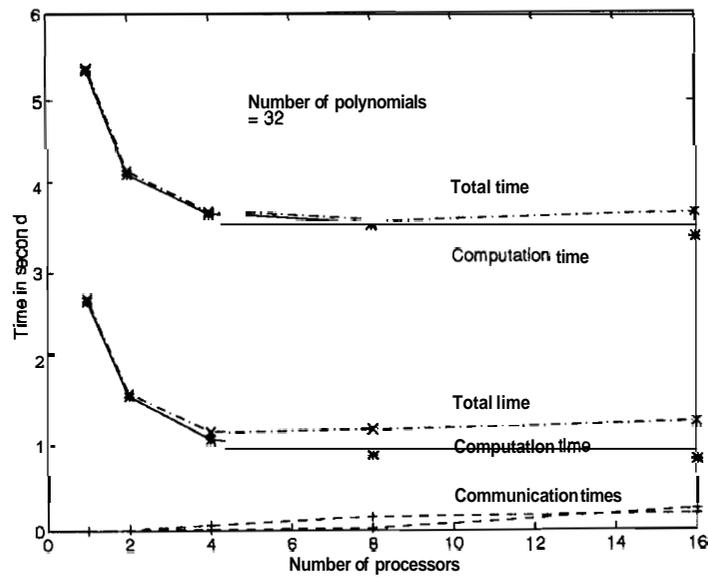


Figure 3: Timing information of *Par-Groebner-Tree* algorithm for Type 1 polynomials

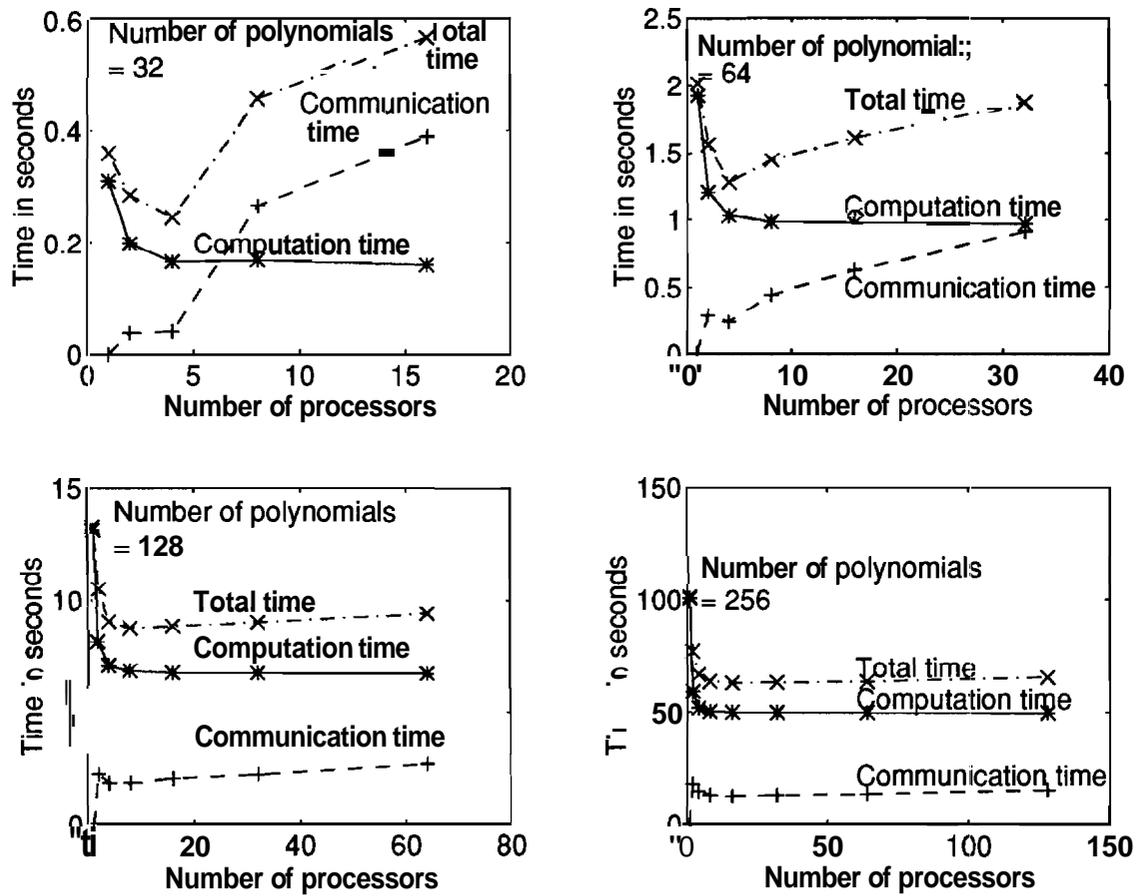


Figure 4: Timing information of *Par_Gröbner_Tree* algorithm for Type 2 polynomials

that by Buchberger's algorithm. Furthermore, the tree structured computation can be naturally parallelized using algorithm *Par-Grobner-Tree*. The parallel execution times obtained using this algorithm can be seen in Figures 3 and 4.

The polynomial sets which are initially combined at the lowest level affect the execution times of the parallel algorithm. For the first type of polynomials, the different initial combinations tried are

1. $(F_1, F_2), (F_3, F_4), \dots, (F_{n-1}, F_n),$
2. $(F_1, F_3), (F_2, F_4), (F_5, F_7), \dots, (F_{n-2}, F_n),$
3. $(F_1, F_5), (F_2, F_6), (F_3, F_7), (F_4, F_8), \dots, (F_{n-4}, F_n).$

From Figure 3, it can be inferred that the execution times for the same set of polynomials differ for different initial combinations. However, for all combinations, a significant reduction in execution time is achieved up to a certain number of processors (4 in this example). On the other hand, an increase in the number of processors after a certain limit results in an increase in communication time without a significant reduction in computation time.

For the second type of polynomial equations, different problem sizes are: used to study the effect of scaling the problem on the computation and communication times. As shown in Figure 4, as the problem size increases, the ratio of the computation time to the communication time also increases. The amount of speed-up achieved is limited after a certain number of processors (4 in this case). The main reason for this behavior is unbalanced load distribution across processors and computation granularity that increases with the level of the nodes in the tree. The nodes at high levels in the computational tree tend to dominate the execution time of the whole computation. So, using more than four processors results in very small execution time savings due to parallel computation at low levels of the tree.

4.2 Conclusions

In this paper, an alternative approach for Grobner basis computation was presented. For large computations, this approach results in faster execution times than Buchberger's algorithm. Furthermore, in this study, parallelism in the tree structured Gröbner basis computation was exploited and the bottlenecks were identified. The tree structured Grobner basis computation presented in this paper provides a structure which is easy to parallelize. On the other hand, the limitation imposed on speed-up achievable by using this approach suggests that the parallelism available inside each node of the computational tree needs to be exploited. This is the basis for an approach, currently being pursued by the authors, that will result in a balanced tree structured Grobner basis computation. Other issues such as reducing coefficient growth, efficient memory management, and avoiding unnecessary computations are also being considered.

References

- [Buc79] 13. Buchberger, "A criterion for detecting unnecessary reductions in the construction of Grobner-basis", In EUROSAM, pages 1–21, 1979.
- [Buc83] 13. Buchberger, "A note on the complexity of constructing Grobner-basis", In *EUROCAL*, pages 137–145, 1983.
- [Buc85] 13. Buchberger, "The parallel 1-machine for symbolic computation", In *EUROCAL*, pages 541–542. Linz, Austria, 1985.
- [Buc85a] 13. Buchberger. "Grobner Bases: An Algorithmic Method in Polynomial ideal Theory" *Multidimensional Systems Theory*, pages 184–232, 1985.
- [BvzGH82] A. Borodin, J. von zur Gathen, and J. E. Hopcroft, "Fast parallel matrix and gcd computations", *Information and Control*, Vol. 52, pages 241–256, 1982.
- [BW93] T. Becker and V. Weispfenning, "Grobner Basis: a *computational* approach to *commutative* algebra", New York : Springer-Verlag, 1993.
- [CGG] 13. Char, "First leaves : a tutorial introduction to Maple V", New York : Springer-Verlag, 1992.
- [CLO93] D. Cox, J. Little, and D. O'Shea, "Ideals, Varieties, and Algorithms", 1993.
- [Gan84] J. Ganthen, "Parallel algorithms for algebraic problems", *SIAM Journal of Computing*, pages 802–824, 1984.
- [Hof90] C. Hoffmann, "Geometric and Solid Modeling", Morgan Kaufmann, 1989.
- [Joh89] J. Johnson, "Some issues in designing algebraic algorithms for the Cray X-MP" In Della Dora and Fitch, editors, *Computer algebra and parallelism*, pages 178–195. Academic Press, 1989.
- [JSSRV] J. Roch, P. Senechaud, F. Siebert-Roch, and G. Villard, "Computer algebra on MIMD machine", pages 423–439.
- [Küc90] . Kuchlin, "The s-threads environment for parallel symbolic computation", In Zippel, editor, *Computer algebra and parallelism*, pages 1–18. LNCS 584, Springer Verlag, 1990.
- [Mis89] 13ud Mishra, "Notes on Grobner basis", *Information Sciences*, Vol. 48, pages 219–252, 1989.

- [MN89] H. Melenk and W. Neun, "Parallel polynomial operations in the large Buchberger algorithm", In Della Dora and Fitch, editors, *Computer algebra and parallelism*, pages 143–158. Academic Press, 1989.
- [NM90] W. Neun and H. Melenk, "Very large Grobner basis calculations", In Zippel, editor, *Computer algebra and parallelism*, pages 90–99. LNCS 584, Springer Verlag, 1990.
- [Pona] C. G. Ponder, "Parallel processors and systems for algebraic manipulation: Current work", pages 15–21, SIGSAM Bulletin.
- [Ponb] C. G. Ponder, "Parallelism and algorithms for algebraic manipulation: Current work", pages 7–14, SIGSAM Bulletin.
- [Pon89] C. Ponder, "Evaluation of "performance enhancements" in algebraic manipulation systems", In Della Dora and Fitch, editors, *Computer algebra and parallelism*, pages 51–73. Academic Press, 1989.
- [Roc90] J. Roch, "An environment for parallel algebraic computation", In Zippel, editor, *Computer algebra and parallelism*, pages 33–50. LNCS 584, Springer Verlag, 1990.
- [Sen89] J. Senechaud, "Implementation of a parallel algorithm to compute a Grobner basis on boolean polynomials", In Della Dora and Fitch, editors, *Computer algebra and parallelism*, pages 159–166. Academic Press, 1989.
- [Sen90] J. Senechaud, "Boolean Grobner basis and their MIMD implementation" In Zippel, editor, *Computer algebra and parallelism*, pages 90–99. LNCS 584, Springer Verlag, 1990.
- [Sen91] P. Senechaud, "A MIMD implementation of the Buchberger algorithm for boolean polynomials", *Parallel Computing*, Vol. 17, pages 29–37, 1991.
- [SH93a] W. Schreiner and H. Hong, "A new library for parallel algebraic computation", In *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 776–783, 1993.
- [SH93b] W. Schreiner and H. Hong, "PACLIB – a system for parallel algebraic computation on shared memory multiprocessors" In *Parallel Systems Fair at the Seventh International Parallel Processing Symposium*, Newport Beach, CA, pages 56–61., 1993.
- [Sie93] K. Siegl, "||MAPLE|| – a system for parallel symbolic computation" In *Parallel Systems Fair at the Seventh International Parallel Processing Symposium*, Newport Beach, CA, 1993.

- [Vid90] J. Vidal, "The Computation of Grobner Basis on a Shared Memory Multiprocessor",
In Proceedings DISCO'90, pages 81-90, 1990.