Fifth International Conference on Rewriting Techniques and Applications, Montreal, Canada, June 1993.

On the Correctness of a Distributed Memory Gröbner Basis Algorithm*

Soumen Chakrabarti and Katherine Yelick

University of California, Berkeley, CA 94720, USA

Abstract. We present an asynchronous MIMD algorithm for Gröbner basis computation. The algorithm is based on the well-known sequential algorithm of Buchberger. Two factors make the correctness of our algorithm nontrivial: the *nondeterminism* that is inherent with asynchronous parallelism, and the *distribution* of data structures which leads to inconsistent views of the global state of the system. We demonstrate that by describing the algorithm as a nondeterministic sequential algorithm, and presenting the optimized parallel algorithm through a series of refinements to that algorithm, the algorithm is easier to understand and the correctness proof becomes manageable. The proof does, however, rely on algebraic properties of the polynomials in the computation, and does not follow directly from the proof of Buchberger's algorithm.

1 Introduction

Buchberger introduced the notion of a *Gröbner basis* of a set of polynomials and presented an algorithm for computing it [4]. We present an algorithm based on his for computing Gröbner bases on a MIMD distributed memory multiprocessor.

Although somewhat controversial [12], Buchberger and others believe that interreduction (keeping the basis reduced with respect to itself) is essential to performance. Our algorithm executes interreduction steps concurrently with the standard critical pair and reduction steps. We believe this is the first attempt at computing Gröbner bases in parallel in an asynchronous message passing framework while performing interreduction. The completion method used in the Gröbner basis computation is typical of other completion procedures, so we expect our design techniques to have wider application.

In this paper, we focus on the question of correctness of the algorithm and how it is affected by parallelization. We identify some of the key points here.

- The proofs rely on algebraic properties of polynomials, rather than being a direct proof that the parallel program is equivalent to the sequential one. The parallelization is not a simple semantics-preserving transformation on the sequential program.

* This work was supported in part by the Advanced Research Projects Agency of the Department of Defense monitored by the Office of Naval Research under contract DABT63-92-C-0026, by AT&T, and by a National Science Foundation through an Infrastructure Grant (number CDA-8722788) and Research Initiation Award (number CCR-9210260). The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

- The proofs are structured around distributed data structures. While a single data structure may be quite complicated internally, its value is abstracted in the proof to a single shared object which does not necessarily exist in the computation.
- Interreduction complicates the parallel algorithm and its proof. Without interreduction, the basis grows monotonically, but with interreduction, elements may be modified and deleted. Thus the algorithm has to ensure correctness in the presence of multiple, *inconsistent* copies of the basis.
- Finally, the design extends the transition-based approach [21] to distributed memory machines.

We have reported on engineering issues and more extensively on performance elsewhere [8]. The algorithm has been implemented on a CM-5 multiprocessor. It outperforms previous parallel algorithms on shared memory machines.

This paper is organized as follows. §2 gives background definitions. §3 presents the parallel algorithm and correctness proof, using a succession of refinements. §4 gives performance numbers, and §5 discusses the relation to the Knuth-Bendix procedure. We finish with some concluding remarks in §6.

2 Notation

In this section we briefly introduce some notation. A more detailed treatment can be found in [14].

Let K be a field and $x_1, ..., x_n$ be variables, arbitrarily ordered as $x_1 > x_2 > \cdots > x_n$. Then $\mathcal{K} = K[x_1, ..., x_n]$ defines a ring of polynomials under standard polynomial arithmetic. A total order $\succ$ on monomials is *admissible* if for all monomials a, p, q it satisfies (1) $p \succeq 1$ (note that $1 = x_1^0 \cdots x_n^0$) and (2) $p \succeq q \;\Rightarrow\; ap \succeq aq$. Also, $p \succ q$ iff $p \succeq q$ and $p \neq q$.

When written in decreasing order of monomials, $\textsc{Term}(p, i)$ denotes the i-th term of polynomial p ($i \geq 1$). A term contains the *coefficient* and the *monomial*: $\textsc{Term}(p, i) = \textsc{Coef}(p, i) \times \textsc{Mono}(p, i)$. The *head term* of a polynomial p is the leading term: $\textsc{Hterm}(p) = \textsc{Term}(p, 1)$. Similarly, $\textsc{Hcoef}(p) = \textsc{Coef}(p, 1)$ and $\textsc{Hmono}(p) = \textsc{Mono}(p, 1)$. $\textsc{Hmono}$, $\textsc{Hcoef}$ and $\textsc{Hterm}$ are naturally extended to sets of polynomials: $\textsc{Hmono}(S) = \{\textsc{Hmono}(p) : \; p \in S\}$, etc. The admissible ordering $\succ$ is extended to polynomials by defining $p \succ q$ iff $\textsc{Hmono}(p) \succ \textsc{Hmono}(q)$, and $p \succeq q$ iff $\textsc{Hmono}(p) \succeq \textsc{Hmono}(q)$.

Given polynomials p and r such that $\textsc{Hmono}(r)$ divides $\textsc{Mono}(p, i)$ for some i, *reduction* of p by r is defined as:

$$p' = p - \frac{\textsc{Term}(p, i)}{\textsc{Hterm}(r)} \times r. \tag{1}$$

Note that $\textsc{Term}(p, i)$ vanishes out of p'. Reduction by a *set* S of polynomials is done by repeatedly reducing p by some element of S. When no element of S can reduce p, it is *irreducible* or in *normal form*, also written $\textsc{Normal}(p, S)$. The collection of all possible normal forms of p when reduced by S is denoted

$\mathrm{NF}_S(p)$. The zero polynomial, 0, is in normal form with respect to any S. The *highest common factor* of two monomials is denoted

$$\textsc{Hcf}(x_1^{i_1}\cdots x_n^{i_n}, x_1^{j_1}\cdots x_n^{j_n}) = x_1^{\min(i_1,j_1)}\cdots x_n^{\min(i_n,j_n)}. \tag{2}$$

Given polynomials p_1 and p_2, with head terms k_1m_1 and k_2m_2 respectively, their *s-polynomial* is given by

$$\textsc{Spol}(p_1,p_2) = p_1\frac{k_2m_2}{\textsc{Hcf}(m_1,m_2)} - p_2\frac{k_1m_1}{\textsc{Hcf}(m_1,m_2)}. \tag{3}$$

The *ideal* generated by a set S of polynomials is denoted by $\textsc{Ideal}(S)$.
Given a set P of polynomials, a *Gröbner basis* of P is a set G of polynomials satisfying the following:

- $\textsc{Ideal}(G) = \textsc{Ideal}(P)$ and
- For each $p \in \textsc{Ideal}(P)$, $\mathrm{NF}_G(p) = \{0\}$.

A survey of the theory can be found in Mishra [14]. Parallel implementations have been surveyed by Vidal [19]. Earlier network implementations have been reported by Siegl [17], Attardi *et al* [1] and Hawley [13].

3 Algorithm Design

In this section we develop the parallel algorithm, starting from the sequential algorithm. Correctness is proved at each step as part of the design process. §3.1 reviews Buchberger's algorithm without interreduction; §3.2 gives a nondeterministic version, which is essentially a parallel algorithm with atomic operations on shared data structures; §3.3 extends the nondeterministic algorithm to handle interreduction; §3.4 presents the distributed algorithm.

3.1 Sequential Algorithm

Figure 1(a) shows Buchberger's sequential algorithm, called S. The two main data structures are G (the basis) and gpq (the set of pairs for $\textsc{Spol}$ computation). For simplicity, this version is without interreduction: polynomials entering G are completely reduced with respect to all previous elements in G, but old basis elements are not checked for reducibility by new entrants. The effect is that polynomials that have entered the basis once are never modified or deleted. We enhance the algorithm with interreduction after we refine S to a transition axiom form.

A correctness proof of S is given by Mishra and Yap [14]; we sketch their proof of partial correctness and give a different proof of termination. Our proof generalizes better to the interreducing algorithm to be described later.

Theorem 1 (Buchberger). *G is a Gröbner basis iff for all $f, g \in G$, $0 \in \mathrm{NF}_G(\textsc{Spol}(f,g))$ ([14], Theorem 5.8).*

Definition 2. A ring R is defined to be *Noetherian* iff it has no infinite ascending sequence $R_1 \subset R_2 \subset R_3 \subset \ldots$ of ideals of R.

Input: F, a finite set of polynomials.
Initially:
  $G = F$
  $gpq = \{\ \{f, g\} : f, g \in G\}$
while $gpq \neq \emptyset$ {
  let $\{f, g\}$ be any pair in gpq
  $gpq = gpq \setminus \{\{f, g\}\}$
  $h = \textsc{Spol}(f, g)$
  $h' = \textsc{Reduce}(h, G)$
  if $h' \neq 0$ {
    $gpq = gpq \cup \{\{f, h'\} : f \in G\}$
    $G = G \cup h'$
  }
}

(a)

Input: F, a finite set of polynomials.
Initially:
  $grq = \emptyset$, $G = F$,
  $gpq = \{\ \{f, g\} :\ f, g \in G\}$.
<u>S-Polynomial</u>
  $\exists \{p, q\} \in gpq\ \Rightarrow$
    $gpq = gpq \setminus \{p, q\}$
    $grq = grq \cup \{\langle p, q, \textsc{Spol}(p, q)\rangle\}$
<u>Augment Basis</u>
  $\exists \langle p, q, r\rangle \in grq : \textsc{Normal}(r, G),\ r \neq 0 \Rightarrow$
    $grq = grq \setminus \{\langle p, q, r\rangle\}$
    $gpq = gpq \cup \{\ \{s, r\},\ s \in G\ \}$
    $G = G \cup \{r\}$
<u>Reduce</u>
  $\exists \langle p, q, r\rangle \in grq :\ \neg\textsc{Normal}(r, G)\ \Rightarrow$
    $r = \textsc{Reduce}(r, G)$

(b)

Fig. 1. (a) Sequential Algorithm S [Buchberger]. G is initialized to the input set F and grows to become a Gröbner basis. Elements in G are never modified. gpq is the set of pairs of polynomials. The function $\textsc{Reduce}(h, G)$ returns some element $h' \in \mathrm{NF}_G(h)$, i.e., it reduces h completely to normal form. (b) G-1: Transition Axiom formulation with one copy of G. Data structures G and gpq as before. Unlike in Algorithm S, $\textsc{Reduce}(r, G)$ need not return a normal form; a partially reduced form will do. Note that p, q are not needed in grq; they just help write cleaner invariants in the correctness proof.

Theorem 3 (Hilbert's Basis Theorem). *If R is a Noetherian ring then so is $R[x_1, x_2, \ldots, x_n]$ ([15], Pages 420–425).*

Lemma 4. *Algorithm S terminates with G a Gröbner basis of F.*

Proof. For partial correctness, observe the loop invariant

$$\forall\ p, q \in G,\ \{p, q\} \notin gpq\ \Rightarrow\ 0 \in \mathrm{NF}_G(\textsc{Spol}(p, q)). \tag{4}$$

If S terminates, $gpq = \emptyset$, so G is Gröbner by theorem 1. For termination, note that $\textsc{Reduce}$ is a terminating computation [14] and consider tuples of the form $\langle M, p\rangle$, built of an ideal M over $\mathcal{K}$ and integer $p \geq 0$, ordered lexicographically as $\langle M_1, p_1\rangle \sqsupset \langle M_2, p_2\rangle$ iff

$$M_1 \subset M_2 \textbf{ or } \Big(M_1 = M_2 \textbf{ and } p_1 > p_2\Big)\,. \tag{5}$$

Each loop iteration of S reduces the tuple $\Big\langle \textsc{Ideal}(\textsc{Hmono}(G)), |gpq|\Big\rangle$, as can be verified easily by examining each axiom. (See Dershowitz and Manna [10] for similar termination proving techniques.)

Algebraic optimizations to the basic algorithm have been developed that test s-polynomials to quickly detect reduction to zero, without actually performing the reduction [5]. Although our implementation includes such improvements, we omit them from the proofs for simplicity.

3.2 Transition Axiom Specification

Transition axioms are a means to exploit non-determinism in a sequential algorithm description. Inspired by guarded command languages [9, 11], and augmented by linearizable data types [21], this style was used to implement a shared-memory Knuth-Bendix procedure [22]. Transition axioms help break the computation into independently schedulable chunks, so the scheduling decisions are deferred until late in the design process. They are written in the form $C \Rightarrow A$ where C is the *enabling condition* (a guard predicate) and A is the action. An execution proceeds by repeatedly *firing* enabled axioms nondeterministically. Termination occurs when none of the axioms can be fired. Parallelism results from being able to overlap axioms in time on multiple processors.

There are two sources of non-determinism in S.

- Reduction has many degrees of freedom, since the choice of a reducer is not specified. Also, it is not required to reduce the argument polynomial completely to normal form with respect to the reducing set; any positive number of reduction steps will do.
- The choice of a pair from gpq to compute the SPOL is not specified (although selection heuristics affect performance). Thus one can work on several pairs simultaneously.

Algorithm G-1 in figure 1(b) is the result of rewriting S as a transition axiom specification. There are three data structures: G is the growing basis, gpq is the pair set as before and grq is a temporary set of polynomials in some stage of being reduced[2]. We now prove that G-1 correctly computes a Gröbner basis.

Definition 5. Let S, T be finite multisets of polynomials with $|S| = |T|$. Define the irreflexive, non-symmetric and transitive ordering $\rhd$ between such multisets as $S \rhd T$ iff there is a bijection $\sigma : S \rightarrow T$ such that $\forall s \in S : \; s \succeq \sigma(s)$ and $\exists s \in S : \; s \succ \sigma(s)$.

Clearly, $\rhd$ is Noetherian. We use it to show that G-1 terminates.

Lemma 6. *Algorithm G-1 terminates with G a Gröbner basis of F.*

Proof. For partial correctness, we give an "axiom invariant" that is true *between* any two successive rules in the firing sequence, specifically, $\forall \; f, g \in G$:

$$\Big(\{f,g\} \in gpq\Big) \;\textbf{ or }\; \Big(\exists r \neq 0 : \; \langle f,g,r\rangle \in grq\Big) \;\textbf{ or }\; \Big(0 \in \mathrm{NF}_G(\mathrm{SPOL}(f,g))\Big)\,. \quad (6)$$

[2] An explanation of names: g for global, meaning they are shared by all processors; p for pairs and r for reducts; q for queues because of the heuristic ordering on monomials in gpq and grq.

The result follows from theorem 1 and the observation that when all the guards are false, $gpq = \emptyset$ and $\langle f, g, r\rangle \in grq \;\Rightarrow\; r = 0$. For termination, consider tuples of the form $\langle M, p, R\rangle$ constructed as in the proof of S, but with the additional field R, a multiset of polynomials. Tuples are ordered lexicographically as before, but with R ordered by $\rhd$. Then the tuple $\left\langle \textsc{Ideal}(\textsc{Hmono}(G)), |gpq|, grq \right\rangle$ is reduced upon firing any axiom.

Input: F, a finite set of polynomials. **Initially:** $grq = \emptyset$, $G = F$, $gpq = \{\ \{f, g\} :\ f, g \in G\}$. <u>S-Polynomial</u> $\exists\{p, q\} \in gpq \;\Rightarrow$ $gpq = gpq \setminus \{p, q\}$ $grq = grq \cup \{\langle p, q, \textsc{Spol}(p, q)\rangle\}$ <u>Reduce</u> $\exists\langle p, q, r\rangle \in grq :\ \neg\textsc{Normal}(r, G) \;\Rightarrow$ $r = \textsc{Reduce}(r, G)$	<u>Augment Basis</u> $\exists\langle p, q, r\rangle \in grq : \textsc{Normal}(r, G),\ r \neq 0 \Rightarrow$ $grq = grq \setminus \{\langle p, q, r\rangle\}$ $gpq = gpq \cup \{\ \{s, r\},\ s \in G\ \}$ $G = G \cup \{r\}$ <u>InterReduce</u> $\exists p, q \in G :\ q$ reduces $p \;\Rightarrow$ $p' = \textsc{Reduce}(p, \{q\})$ $G = (G \setminus \{p\}) \cup \{p'\}$ $gpq = gpq \cup \{\ \{p', g\} :\ g \in G,\ g \neq p'\}$

Fig. 2. IG-1: Transition Axiom formulation for interreduction with one copy of G. <u>InterReduce</u> might reduce a basis element to zero; we assume for simplicity that zero elements are left around in G but are never considered as reducers.

3.3 Interreduction

We have parallelized and distributed S without interreduction [7]. The proof of that algorithm is a special case of the algorithm with interreduction: interreduction introduces mutation of polynomials in the basis. We present only the more general case here, and therefore proceed by introducing interreduction into our nondeterministic algorithm.

Buchberger describes an elaborate way to keep track of polynomials that become reducible each time the basis grows, so that after each addition the basis is *interreduced*, i.e., basis polynomials are sequentially reduced by each other until nothing more can be reduced [4]. In a parallel algorithm, this global interreduction could potentially change a polynomial used by any other transition axiom, yet we cannot afford to stop other work while interreduction proceeds. We therefore introduce a single interreduction step as a separate transition axiom.

Figure 2 shows the transition axioms IG-1 for interreducing Gröbner basis computation. The only modification is the addition of <u>InterReduce</u>. As in G-1, there is one shared copy of G. While the extent of reduction done in <u>Reduce</u> is not specified, in the proof we assume only a single reduction step occurs in <u>InterReduce</u>. It follows that correctness is preserved if <u>InterReduce</u> were to reduce multiple steps, which is done in the implementation. The additional properties to be proved for IG-1 are that interreduction maintains the axiom invariant, and does not destroy progress.

Lemma 7. *Let* $\underline{\textsc{InterReduce}}$*, be invoked on* G_1*, resulting in* G_2*. Then for any polynomial* p*, if* $0 \in \mathrm{NF}_{G_1}(p)$ *then* $0 \in \mathrm{NF}_{G_2}(p)$*.*

Proof. There must be $g, h \in G_1$ such that h reduces g for $\underline{\textsc{InterReduce}}$ to be enabled. Suppose $g \xrightarrow{h} g'$. Let $0 \in \mathrm{NF}_{G_1}(p)$. We need to show that $0 \in \mathrm{NF}_{G_2}(p)$. Consider the reduction sequence $p \rightarrow \cdots p_i \rightarrow p_{i+1} \cdots \rightarrow 0$ in G_1. If there is no reduction by g there is nothing to prove, so suppose the $p_i \rightarrow p_{i+1}$ reduction is by g, denoted $p_i \xrightarrow{g} p_{i+1}$. In G_2 we can achieve the same reduction of p_i to p_{i+1} in two steps:

$$p_i \xrightarrow{h} p_{\text{new}} \xrightarrow{g'} p_{i+1}. \tag{7}$$

We can do this for all steps that used g as a reducer to get a reduction sequence using only reducers from G_2. Thus, $0 \in \mathrm{NF}_{G_2}(p)$.

Lemma 8. *Let* $\underline{\textsc{InterReduce}}$ *be invoked on* G_1*, resulting in* G_2*. Then*

$$\textsc{Ideal}(\textsc{Hmono}(G_1)) \subseteq \textsc{Ideal}(\textsc{Hmono}(G_2)).$$

Proof. For $\underline{\textsc{InterReduce}}$ to be enabled, $\exists g, h \in G_1$ such that h reduces g to g'. If $\textsc{Hmono}(g)$ is unaffected there is nothing to prove, so suppose $\textsc{Hmono}(h)$ divides $\textsc{Hmono}(g)$. So, $\textsc{Ideal}(\textsc{Hmono}(G_1 \setminus \{g\})) = \textsc{Ideal}(\textsc{Hmono}(G_1))$. Thus, $\textsc{Ideal}(\textsc{Hmono}(G_2)) = \textsc{Ideal}\left(\textsc{Hmono}(G_1 \setminus \{g\} \cup \{g'\})\right) \supseteq \textsc{Ideal}(\textsc{Hmono}(G_1 \setminus \{g\}))$.

Lemma 9. *Algorithm IG-1 terminates with* G *a Gröbner basis of* F*.*

Proof. Partial correctness is direct from lemma 6 and lemma 7. For termination, we augment our proof for G-1. Consider tuples of the form $\langle M, S, p, R\rangle$ as in the proof of Lemma 6, but with the additional field S, a set of polynomials. Tuples are lexicographically ordered as before, with S ordered by $\rhd$. We can show that firing any axiom in IG-1 reduces the tuple

$$\left\langle \textsc{Ideal}\left(\textsc{Hmono}(G)\right), G, \left|gpq\right|, grq \right\rangle. \tag{8}$$

3.4 Replicating the Basis

For a distributed memory algorithm, it is not realistic to assume that processors always have consistent copies of shared data. Replication may occur either on a large scale by replicating an entire data structure, or on a small scale by keeping temporary copies of individual pointers and values. A consistency problem arises when any of the replicated values may be mutated.

In the Gröbner basis computation, the most important data structure in question is the basis, since it is shared most extensively. The basis could be distributed by partitioning or replication, but a pragmatic analysis of load balance, granularity and communication requirements [8] favor replication. Given a replicated basis, we have to address the problem of maintaining consistency without introducing excessive overhead. Fortunately, the consistency requirement on the basis is rather lax: a processor can do significant amounts of useful work while having an incomplete or even inconsistent copy of the basis.

Allowing Inconsistent Copies

An example of a consistency problem that may occur is the following "race condition" [16]. Suppose processors P_1 and P_2 both have copies of polynomials g, h, which happen to be equal. <u>InterReduce</u> fires on P_1 and P_2. Say g is reduced by h to 0 on P_1. Processor P_2 does not modify its copy of g, instead it reduces h by g to 0. Subsequent invalidation messages lead both processors to discard their copies of g and h, possibly destroying the correctness of the solution. A solution to this special case is to impose a total order Age on polynomials such that if $f = g$, f is allowed to reduce g to 0 only if the order is favorable.

In general, a stronger check is needed, namely, the total order should be used whenever the *head monomials* of the reducer and the reduced are equal, even if they are not completely equal. It is easy to verify that this check prevents the particular error indicated, but it is still non-trivial to show correctness in general.

Version Sequences

To keep track of mutable basis polynomials we introduce the notion of the *version sequence* of a polynomial. Suppose a polynomial $p(0)$ enters the basis, and is successively reduced by $r_1, r_2, \ldots, r_t$ to $p(1), p(2), \ldots, p(t)$. We represent this life history by the notation

$$p = \left[p(0) \xrightarrow{r_1} p(1) \xrightarrow{r_2} p(2) \xrightarrow{r_3} \ldots \xrightarrow{r_t} p(t) \right], \tag{9}$$

where p represents the version sequence and $p(t)$ represents the t-th version of p. In our implementation, version sequences are identified by unique ID's.

The Model

Let each processor i have access to a (possibly inconsistent) local copy G_i of the basis, $1 \leq i \leq P$. In addition, we have a global *shadow set* G' which contains version sequences of all polynomials that ever entered the basis. Also, for ease of both implementation and proof, we assume each polynomial is *owned* by its creator processor which thereafter is the *only* processor authorized to mutate the polynomial[3].

When processor i creates a new polynomial f and value $f(0)$, it creates a new version sequence (which, by abuse of notation, we also call f) $f = [f(0)]$ in G'. When processor i modifies an owned polynomial $f(t)$ to $f(t+1)$ (as a result of reducing by $h_{t+1}(e_{t+1})$: version e_{t+1} of polynomial h_{t+1}) it appends the new value to the version sequence f in G', changing it to

$$\left[f(0) \xrightarrow{h_1(e_1)} \ldots \xrightarrow{h_t(e_t)} f(t) \xrightarrow{h_{t+1}(e_{t+1})} f(t+1) \right]. \tag{10}$$

---

[3] This is not a serious limitation. The alternative is to associate modify locks with each polynomial.

Input: F, a finite set of polynomials.
Initially:
$grq = \emptyset$,
$gpq = \{\ \{f, g\} :\ f(0), g(0) \in F\}$.
$\forall i :\ 1 \leq i \leq P, G_i = F$
$G' = \{[f(0)] :\ f(0) \in F\}$
Processor i, $1 \leq i \leq P$.
<u>VALIDATE</u>
$(gpq \neq \emptyset$ **or** $grq \neq \emptyset)$ **and** $\exists g(t) \in G' :\ \forall g(\ell) \in G_i, \ell < t\ \Rightarrow$
$G_i = \left(G_i \setminus \bigcup_{0 \leq e < t} g(e)\right) \cup g(t)$
<u>S-POLYNOMIAL</u>
$\exists\{f, g\} \in gpq :\ f^*, g^* \in G_i\ \Rightarrow$
$gpq = gpq \setminus \{\ \{f, g\}\ \}$
$grq = grq \cup \{\langle f^*, g^*, \text{SPOL}(f^*, g^*)\rangle\}$
<u>AUGMENT BASIS AND INVALIDATE</u>
$G_i = \{g^* :\ g \in G'\}$, $\exists\langle p, q, r\rangle \in grq : r \neq 0$, **and** $\text{NORMAL}(r, G_i)\ \Rightarrow$
$grq = grq \setminus \{\langle p, q, r\rangle\}$
Create unique ID h for r, so that $h(0) = r$
$G' = G' \cup \{[h(0)]\}$ `/* create new version sequence */`
$gpq = gpq \cup \{\ \{g, h\},\ g^* \in G_i\ \}$
$G_i = G_i \cup \{r\}$
<u>REDUCE</u>
$\exists\langle p, q, r\rangle \in grq :\ \neg\text{NORMAL}(r, G_i)\ \Rightarrow$
$r = \text{REDUCE}(r, G_i)$
<u>INTERREDUCE AND INVALIDATE</u>
$\exists f^* = f(t), h(e) \in G_i :\ h(e)$ reduces f^*, f^* owned,
$\big(\text{HMONO}(f^*) \neq \text{HMONO}(h(e))$ **or** $\text{AGE}(f^*) > \text{AGE}(h(e))\big)\ \Rightarrow$
$f(t+1) = \text{REDUCE}(f(t), \{h(e)\})$
$G' = (G' \setminus \{f\}) \cup \{[f \xrightarrow{h(e)} f(t+1)]\}$ `/* append latest version */`
$gpq = gpq \cup \{\ \{f, g\} :\ g \in G'\ \}$

Fig. 3. IG-P: Transition axioms for interreduction using P copies of G. Note that since gpq now contains ID's, not polynomial values, we effectively generate new polynomial pairs in <u>INTERREDUCE</u> by requiring in the guard of <u>S-POLYNOMIAL</u> that the polynomials in G_i are the latest.

The local copy G_i consists of a selection of versions from a subset of version sequences in G'. A validation operation either puts the first element of a new version sequence in G_i or replaces version $g(t)$ from a version sequence g by $g(t+\ell)$, $\ell > 0$. The latest element in a version sequence f at a given time is special; we call it f^*.

As before, we will need to define an abstract basis $\mathcal{G}$ in terms of the physical data structures. The following definition will serve our purpose.

$$\mathcal{G} = \left\{\ g^* :\ g \in G'\ \right\}. \tag{11}$$

Using this model, we now write the transition axioms IG-P in figure 3. As mentioned before, <u>VALIDATE</u> picks some polynomial in the system of which processor i has no copy or a stale copy, and gets a copy or advances to a later version. Invalidation has two forms. When <u>AUGMENT BASIS</u> fires, processor i adds a new version sequence in G'; when <u>INTERREDUCE</u> fires, processor i appends the new version to the extant sequence. <u>REDUCE</u> and <u>S-POLYNOMIAL</u> are as before. Each polynomial (alias version sequence) has a unique ID from a totally ordered set (integer in our implementation) which will be used by AGE to break reduction loops as mentioned before.

The act of copying a version from a version sequence in G' to G_i models the communication step to update processor i's copy of the basis. The value of a polynomial cannot be used unless this is performed. However, we can still manipulate the ID, like putting it into gpq as in <u>INTERREDUCE</u>. ID's are very lightweight (8 bytes) compared to the polynomials they represent (hundreds to thousands of bytes). Hence communicating ID's is faster and cheaper than transporting polynomials. This has guided the formulation of the model.

Lemma 10. *If f reduces g then $g \succeq f$.*

Lemma 11. *Let an invocation of* <u>INTERREDUCE</u> *modify the basis from $\mathcal{G}_1$ to $\mathcal{G}_2$. For any polynomial y, if $0 \in \mathrm{NF}_{\mathcal{G}_1}(y)$ then $0 \in \mathrm{NF}_{\mathcal{G}_2}(y)$.*

Proof. Suppose $0 \in \mathrm{NF}_{\mathcal{G}_1}(y)$. We need to show that $0 \in \mathrm{NF}_{\mathcal{G}_2}(y)$. Say the invocation of <u>INTERREDUCE</u> reduces $f_1(t_1)$ to $f_1(t_1+1)$. Suppose $y_0 = y \rightarrow \cdots y_i \rightarrow y_{i+1} \cdots \rightarrow 0 = y_j$. The problem is that all reducers in the above reduction chain, even though elements in $\mathcal{G}_1$, may not be in $\mathcal{G}_2$ (using $f_1(t_1)$ as a reducer, for example).

We demonstrate how to replace *one* occurrence of a non-latest element in the reducers by latest elements alone. Since the reduction sequence is finite, we can replace such occurrences one by one.
We can use lemma 7 to replace old versions of reducers by new ones. Suppose G' contains the interreduction step

$$f_i(j) \xrightarrow{h_{ij}(e_{ij})} f_i(j+1). \tag{12}$$

Then any reduction step $y_{k-1} \xrightarrow{f_i(j)} y_k$ can be replaced as in the proof of lemma 7 by the equivalent computation

$$y_{k-1} \xrightarrow{h_{ij}(e_{ij})} y_{\text{new}} \xrightarrow{f_i(j+1)} y_k. \tag{13}$$

This seems closer to our goal: $f_i(j+1)$ is closer to f^* than $f_i(j)$, and we have brought in a different element from a finite set. We can continue this until all reducers that take x to x' are in $\mathcal{G}_2$. This gives rise to a transformation tree: latest version polynomials are leaf nodes. A non-latest reducer $f_i(j)$ has children $h_{ij}(e_{ij})$ and $f_i(j+1)$. How are we guaranteed that the tree is finite?
Define the lexicographic extension of ordering $\succeq$ and the ordering imposed by AGE on polynomials as $p > q$ if $p \succ q$ or $\mathrm{HMONO}(p) = \mathrm{HMONO}(q)$ and $\mathrm{AGE}(p) > \mathrm{AGE}(q)$. Without loss of generality, let the root of some subtree be $f_{i_1}(j_1)$. Since

the tree is heap ordered with respect to $>$ defined above (with the root as the greatest element), $f_{i_1}(j_1)$ cannot occur anywhere in the subtree. Since the total number of versions of all polynomials in the system is finite, the result follows.

Lemma 12. *Let an invocation of* $\underline{\text{InterReduce}}$ *modify the basis from* $\mathcal{G}_1$ *to* $\mathcal{G}_2$*. Then* $\text{Ideal}(\text{Hmono}(\mathcal{G}_1)) \subseteq \text{Ideal}(\text{Hmono}(\mathcal{G}_2))$.

Proof. Similar to the proof of lemma 11. Suppose $\underline{\text{InterReduce}}$ performed the reduction

$$f_1(t_1) \longrightarrow f_1(t_1+1). \tag{14}$$

If the reducer is in $\mathcal{G}_2$ there is nothing more to prove. Suppose the reducer is not a final element, i.e., there is a reduction

$$f_i(j) \stackrel{h_{ij}(e_{ij})}{\longrightarrow} f_i(j+1) \tag{15}$$

so that $\text{Hmono}(h_{ij}(e_{ij}))$ divides $\text{Hmono}(f_i(j))$. In that case

$$\text{Ideal}\left(M \cup \{\text{Hmono}(f_i(j))\}\right) \subseteq \text{Ideal}\left(M \cup \{\text{Hmono}(h_{ij}(e_{ij}))\}\right) \tag{16}$$

for any set of monomials M. Continue till a final version reducer is encountered. Since versions are drawn from a finite set and Age prevents repeating reducers, we must reach a final version reducer.

Lemma 13. *IG-P terminates, computing a Gröbner basis of* F*.*

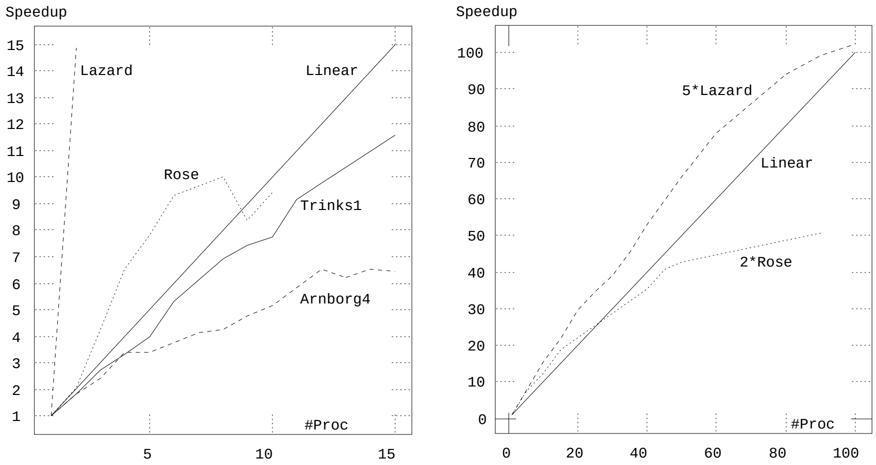
Proof. Partial correctness follows from lemma 9 and 11. For termination, we adapt the proof of lemma 9, using lemma 12 and replacing G by $\mathcal{G}$ in the tuple in lemma 9.

4 Implementation and Performance

Our prototype runs on the CM-5 multiprocessor [6]. Each processor is a 33 MHz (15–20 MIPS) Sparc with about 8 MB of memory. The network is a fat-tree supporting at most 20 MB/s point-to-point data transfer. The prototype is in C, with the *active message* layer [20] for communication. For the benchmarks we used, there was no remarkable difference in performance with and without interreduction. The results are quoted without interreduction. Also, we reduced only head terms in Reduce, not all terms. This also produces a Gröbner basis, but not necessarily a unique one. In Figure 4 some of the speedups on the CM-5 multiprocessor are given. The first set (a) is done on a small number of processors using some standards benchmarks [19].

Scalability

The standard benchmarks complete in a few seconds. Scalability is better than some previous shared memory implementations, but is still limited by the small total number of tasks (pairs added to the pair queue). To see if this is a fundamental limiting factor, we synthesized problems that lead to a large



(a) Benchmarks (b) Scaled Input

Fig. 4. Speedups for (a) standard benchmarks on a few processors and (b) Inputs created with multiple copies of a benchmark on many processors. X-axis: number of processors; Y-axis: ratio of 1-processor running time and P-processor running time. Superlinear behavior is seen in some cases.

number of tasks, using multiple copies (two and five, respectively) of the standard benchmarks, with variables renamed between the copies. The results are shown in figure 4(b). We also proved a geometry theorem using Gröbner basis that was too large to be run sequentially.

Even though the algorithm has good time scalability for long running problems, the indiscriminate replication makes it scale poorly in space. We came across a few examples that are extremely long-running, but replication exhausts memory. It is clear from our analysis [8] that time scalability is favored by replication. Solving large real problems seems to need a compromise with partitioning. We are designing a general object library that permits replication as far as memory capacity permits, thus making the compromise on a continuum.

5 Applications to Term Rewriting

Many of the parallelization techniques and correctness results in this paper could be applied to Knuth-Bendix and other completion procedures. A precursor to this work was a shared memory implementation of the Knuth-Bendix procedure, which was also based on a transition axiom style [22]. See also [18] for a generic parallel completion procedure and [3] for some related correctness results for distributed computations. The pragmatic question of whether other completion procedures would perform well on distributed memory machines is beyond the scope of this paper. However, in this section we discuss the ways in which the algorithms and proofs could be extended to a distributed memory Knuth-Bendix procedure.

Proving partial correctness of a distributed Knuth-Bendix procedure follows roughly the same lines as for Gröbner basis. However, the completion problem for term rewriting systems is undecidable. For some sets of rewrite rules, no finite complete system exists. The termination requirement for Gröbner basis is therefore replaced by a *liveness* condition, stating that the procedure must continually make progress towards a complete system. The procedure is usable as a semi-decision procedure, in that any equational theorem must eventually be provable by rewriting. In addition, the Knuth-Bendix procedure may fail. Failure also impacts the correctness criteria for the procedure, although incorporating it into the proofs should be straightforward. Interreduction is a performance improvement in both procedures, and is not essential for correctness. Whereas its value is arguable in the Gröbner basis computation, it is considered essential in Knuth-Bendix.

The basic outline of the liveness proofs could also be extended. Note that the nontermination property of Knuth-Bendix creates a subtle distinction in distributing the two computations. Hilbert's Basis Theorem guarantees that *all* increasing chains of ideals were finite, so we could have relaxed the guard of the AUGMENT BASIS AND INVALIDATE axiom in Figure 3 by not requiring the local copy G_i to be up to date. This would still terminate, but would not necessarily be practical. A Knuth-Bendix procedure must have regularly scheduled validations, since there is no analog to Hilbert's Theorem for term rewriting systems. Completing a subset of rewrite rules could lead to nonterminating executing, forever missing some critical pair. Requiring validations at all add points, as our Gröbner basis algorithm does, is sufficient in either domain, and a less stringent policy might also be possible.

In spite of these differences, the similarity between the two sequential procedures carries over to the distributed case. The race condition mentioned in §3.4, that comes with parallel interreduction, also exists for rewrite rules: two copies of the same rule can be used to reduce one another so that both disappear. It also has a similar solution, in that rewrite rules can be time stamped to prevent reduction cycles. Informally, the analog to Lemma 11 says that one can reduce using out-of-date copies of the rewrite rule set, since any reductions done there could have been performed with the latest set. Similarly, the liveness argument is analogous to termination for Gröbner basis. However, Hilbert's Basis Theorem would be replaced by the proof ordering notion of Bachmair *et al* as the basic measure of progress [2]. The proof ordering results are already quite general, giving the correctness of nondeterministic algorithm with interreduction, similar to IG-1 here.

6 Future work and Conclusion

In this paper, we have described the design and implementation of a parallel Gröbner basis procedure. We believe that current performance can be further improved in the following ways.

- As mentioned in §4, a replicated basis favors scalability in terms of achievable speedup. For large problems, it is not practical to maintain complete copies at all processors. We are implementing a generic library for application level caching of data structures with some weak consistency models that are profitable for the application.

- After each interreduction step reducing p to p', the algorithm has to add pairs involving p' (to maintain the correctness invariant). In the sequential algorithm (where a random access on the pair "queue" is assumed), one can also remove all pairs involving p, for efficiency reasons. This is not feasible in a distributed memory setting with high communication expense. Are there efficient techniques to reorganize the distributed pair queue? Are there algebraic properties that obviate adding all new pairs with p'?
- In our design, the limit to granularity is a reduction step. This appeared reasonable for the target architecture. A general parallelization recipe for a variety of architectures will be useful. In particular, vectorizing the infinite precision coefficient computations should improve absolute performance.

In conclusion, we have presented a distributed memory MIMD algorithm for computing Gröbner basis. Our implementation out-performs the shared memory implementation of Vidal [19] fairly consistently, and has the additional advantage that shared memory hardware is not assumed. The transition-based approach, previously used for shared memory [21], is extended here for distributed memory. The key idea is to replace the shared data structures with distributed data structures, for which replication and partitioning of the data is hidden. The encapsulation of distributed objects and the structure provided by the transition axioms helps in both the algorithm presentation and in the correctness proof, and we believe it will be useful in other problems that have irregular patterns of communication and control.

Acknowledgements

Steve Schwab provided the packages for *bignum* and polynomial arithmetic and a shared memory Gröbner basis program developed at CMU. Chih-Po Wen contributed a distributed memory task queue package. We are also grateful to the referees for their review and comments.

References

1. G. Attardi and C. Traverso. A Network Implementation of Buchberger Algorithm. Technical Report 1177, University di Pisa, January 1991.
2. L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for Equational Proofs. In *Proceedings of the Symposium on Logic in Computer Science*, pages 346–357. IEEE, 1986.
3. M. P. Bonacina. *Distributed Automated Deduction.* PhD thesis, Department of Computer Science, SUNY at Stony Brook, December 1992.
4. B. Buchberger. Gröbner basis: an algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, chapter 6, pages 184–232. D. Reidel Publishing Company, 1985.
5. B. Buchberger. A Criterion for detecting Unnecessary Reductions in the construction of Gröbner Bases. In *Proceedings of the EUROSAM '79, An International Symposium on Symbolic and Algebraic Manipulation*, pages 3–21, Marseille, France, June 1979.

6. N. J. Burnett. The Architecture of the CM-5. In *IEEE Colloquium on 'Medium Grain Distributed Computing' (Digest 070)*, pages 1–2, London, 26 March 1992.
7. S. Chakrabarti. A distributed memory Gröbner basis algorithm. Master's thesis, University of California, Berkeley, December 1992.
8. S. Chakrabarti and K. Yelick. Implementing an Irregular Application on a Distributed Memory Multiprocessor. In *Principles and Practices of Parallel Programming*, May 1993.
9. K. M. Chandy and J. Misra. *Parallel Program Design : a Foundation.* Addison-Wesley Publishing Company, Reading, Mass., 1988.
10. N. Dershowitz and Z. Manna. Proving Termination with Multiset Orderings. *Communications of the ACM*, 22:465–476, 1979.
11. E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
12. A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. "One sugar cube, please" OR Selection strategies in the Buchberger algorithm. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 49–54, Bonn, Germany, 15–17 July 1992.
13. D. J. Hawley. A Buchberger algorithm for Distributed Memory Multi-processors. In *Proceedings of the 1st International ACPC Conference on Parallel Computation*, pages 385–390, Salzburg, Austria, 30 September – 2 October 1991. Springer-Verlag.
14. B. Mishra and C. Yap. Notes on Gröbner basis. In *Information Sciences 48*, pages 219–252. Elsevier Science Publishing Company, 1989.
15. Nathan Jacobson. *Basic Algebra — Volume 2.* W. H. Freeman and Company, New York, 1989.
16. C. G. Ponder. Evaluation of "performance enhancements" in algebraic manipulation systems. Technical Report UCB/CSD 88/438, University of California, Berkeley, 1988. Chapter 7, Parallel Algorithms for Gröbner Basis Reduction.
17. K. Siegl. Parallel Gröbner basis computation in ||MAPLE||. Technical Report 92-11, Research Institute for Symbolic Computation, Linz, Austria, 1992.
18. J. K. Slaney and E. W. Lusk. Parallelizing the Closure Computation in Automated Deduction. In *Proceedings of the 10th International Conference on Automated Deduction*, pages 28–29. Springer-Verlag, LNCS 449, 1990.
19. J.-P. Vidal. The computation of Gröbner bases on a shared memory multiprocessor. Technical Report CMU-CS-90-163, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1990.
20. T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 256–266, 1992.
21. K. Yelick. Using abstraction in explicitly parallel programs. Technical Report MIT/LCS/TR-507, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, July 1991.
22. K. A. Yelick and S. J. Garland. A parallel completion procedure for term rewriting systems. In *Conference on Automated Deduction*, Saratoga Springs, NY, 1992.