ICOT Technical Report: TR-0898

TR-0898

Parallel Computation of Gröbner Bases on Distributed Memory Machines

by

H. Sawada (MEL), S. Terasaki (Matsushita)

& A. Aiba

November, 1994

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191~5

Institute for New Generation Computer Technology

Parallel Computation of Gröbner Bases on Distributed Memory Machines

HIROYUKI SAWADA[†], SATOSHI TERASAKI[‡] AND AKIRA AIBA[*]
[†]Mechanical Engineering Laboratory,
1–2 Namiki, Tsukuba, Ibaraki 305, Japan
[‡]Matsushita Electric Industrial Co., Ltd.,
4–5–15 Higashi-shinagawa, Shinagawa-ku, Tokyo 140, Japan
[*]Institute for New Generation Computer Technology,
1–4–28 Mita, Minato–Ku, Tokyo 108, Japan

Abstract

This paper reports our work on parallelizing an algorithm computing Gröbner bases on a distributed memory parallel machine. When computing Gröbner bases, the efficiency of computation is dominated by the total number of S-polynomials. To decrease the total number of S-polynomials it is necessary to apply a selection strategy that selects the minimum polynomial as a new element of an intermediate base.

On a distributed memory parallel machine, as opposed to a shared memory parallel machine, we have to take into account non-trivial communication costs between processors. To reduce such communication costs, it is better to employ coarse grained parallelism rather than fine grained parallelism.

We adopt a manager-worker model. S-polynomials are reduced in worker processes in parallel, and the minimum polynomial is selected in the manager process. To implement the selection strategy in this parallel model, synchronization between worker processes is required for every selection of a new element of the intermediate base. However, in spite of synchronization, introducing the selection strategy produces not only a better absolute computation speed but also better speedup with multi-processors. We achieved about 8 times speedup with 64 processors for large problems, T-6 and Ex-17.

1 Introduction

Constraint logic programming (*CLP*) is a programming paradigm proposed by Jaffar & Lassez (1987) and Colmerauer (1987) and is an extension of logic programming. At ICOT, we have been researching CLP since 1987, and we have been developing two CLP languages, CAL (*Contrainte Avec Logique*), reported in (Aiba *et al.*, 1988), and GDCC (*Guarded Definite Clauses with Constraints*), reported in Hawley (1991) and (Terasaki *et al.*, 1992). CAL is a sequential CLP and GDCC is a parallel CLP.

CLP improves the descriptive power of a language by introducing a facility to handle relationships in certain domains other than syntactic equivalence of terms. To implement a CLP language, one has to implement a subsystem called a *constraint solver* to handle the extra relationships.

CAL and GDCC are both CLP languages with constraint solvers that have the ability to handle non-linear algebraic equations by employing the Buchberger algorithm (Buchberger, 1965, 1983) to compute the Gröbner base of given equations.

In our research and development of the GDCC parallel constraint solver, our major concern is the absolute speed of computing Gröbner bases by parallel processing. To parallelize the Buchberger algorithm, the absolute computation speed with a single processor must be first improved. Then, the speedup has to be improved with multi-processors. If we have a slow computation speed with a single processor, it is

[†‡]This research was conducted at ICOT.

easy to provide deceptive speedup with multi-processors. However, the efficiency of the constraint solver is determined by the absolute speed, not by the speedup. Therefore, the speedup must be evaluated along with the absolute computation speed, even though some works regard the speedup as more important than the absolute computation speed.

In general, the major issues in developing efficient parallel software are the machine architecture on which the software is developed, and the programming language used to write it. We implement our parallel constraint solver for GDCC on a parallel inference machine *PIM/m* developed at ICOT, as reported in Taki (1992), and by using the kernel language *KL1* reported in Chikayama (1992) and Ueda & Chikayama (1990) for the parallel inference machine. PIM/m consists of 256 processors, and each processor is connected to the others by a two dimensional mesh network.

When implementing the parallel constraint solver, we have to take into account the fact that PIM/m is a distributed memory machine. Unlike a shared memory parallel machine, communication between processors is not negligible compared to the computation on a processor. For this reason, on PIM/m, relatively coarse grained parallelism is more suitable than fine grained parallelism to reduce the amount of communication between processors.

In our research on the parallelization of the constraint solver, we made the following three models: a pipeline model, a distributed-rewriting model, and a manager-worker model.

The pipeline model was our first attempt to parallelize the Buchberger algorithm based on the parallelization of the Knuth-Bendix algorithm by Fujita (1989). The basic idea of this model is to form a looped pipeline in which polynomials flowed to parallelize processes connected by the pipeline. However, we found that this model was not successful. First of all, since the pipeline in this implementation formed a loop, the slowest process determined the overall computational efficiency. Furthermore, experiments showed that we were trying to parallelize inappropriate parts of the algorithm, that is, we did not parallelize the most computational intense part of the algorithm.

Next, we tried to implement the distributed-rewriting model in Hawley (1991). Since the most computational intense part of the Buchberger algorithm was the reduction, we tried to parallelize it. This implementation showed good absolute computation speed for small problems, but for large problems, both the speedup and the absolute computation speed was poor. In sequential implementations of the Buchberger algorithm, it is well known that there is a selection strategy of a new element of an intermediate base to greatly improve efficiency. To implement the strategy, synchronization between reduction processes was required. However, this parallel implementation did not synchronize these processes.

Lastly, we tried to implement the manager-worker model to overcome the difficulty of synchronizing reduction processes. One of our parallel constraint solvers with the manager-worker model could calculate the Gröbner base for the T-6 problem in about 240 minutes with a single processor and in about 30 minutes with 64 processors on PIM/m, giving a speedup factor of about 8. On a Sun Sparc Server 490, it took about 90 minutes for the same problem by Backelin & Fröberg (1991).

This paper is organized as follows. In Section 2, we describe works related to parallelizing the Buchberger algorithm. In Section 3, our approach to parallelization of the constraint solver and the parallel model of the manager-worker are described, and the results of the experiments are listed.

2 Related works

Several attempts have been made to parallelize the Buchberger algorithm on shared memory machines by Melenk & Neun (1988), Vidal (1990), Buchberger (1987), and Clarke *et al.* (1990), and on distributed memory machines by Ponder (1988), Senechaud (1989), and Siegl (1990).

For shared memory machines, two parallelisms were implemented: coarse grained parallelism and fine grained parallelism. Coarse grained parallelism parallelizes reduction by reducing several S-polynomials simultaneously, while fine grained parallelism parallelizes reduction of a polynomial by dividing it based on the fact that only access to the leading monomial is necessary to control the Buchberger algorithm and that an S-polynomial is a linear combination of two polynomials.

In 1988, Melenk & Neun (1988) proposed fine grained parallelism and achieved about 2 times speedup on a two-processor CRAY X-MP. In 1990, Vidal (1990) implemented coarse grained parallelism based on the idea, mentioned by Buchberger (1987), of reducing several S-polynomials simultaneously. His parallel algorithm was implemented on a 16 processor *Encore* machine, and achieved 14 times speedup with 12 processors. In (Clarke *et al.*,1990), these two techniques were combined and it was found that fine grained parallelism only worked for sufficiently large problems, such as Rose in (Boege *et al.*,1986).

On the other hand, for distributed memory machines, fine grained parallelism, which parallelizes reduction of a polynomial by dividing it, was not implemented because of the non-trivial communication costs between processors. Ponder (1988) described three parallel algorithms: an algorithm to reduce several S-polynomials simultaneously (*Parallel S-polys*), an algorithm to parallelize the interreduction between polynomials in the intermediate base (*Parallel Reduction*), and an algorithm to solve a problem under different orderings among variables to see which ordering is fastest. He achieved 1 to 2 times speedup with 4 processors using the Parallel S-polys and Parallel Reduction algorithms. By using alternative orderings, he found that the execution time of the Buchberger algorithm is highly sensitive to ordering, and obtained at best that the fastest ordering was 66 times faster than the slowest. However, the ordering of the fastest computed Gröbner base and the ordering of a user's request will often be different. Therefore, the fastest computed Gröbner base is not always the base which the user really wants.

For Boolean Gröbner bases, Senechaud (1989) parallelized generation and reduction of S-polynomials by distributing polynomials of the intermediate base to processors which form a ring structure and by making subsets of the intermediate base circulate around the ring. She achieved 8 times speedup with 16 processors for a problem with 32 polynomials and 5 variables. However, it is not clear that her method is also effective for algebraic polynomials because of their complicated coefficients. In 1990, Siegl (1990) used a medium grain pipeline principle that parallelized reduction by making a pipeline of polynomials of the intermediate base. That was implemented in STRAND88 on a transputer machine. He achieved 6 times speedup with 16 processors for a small problem.

Because our machine is a distributed memory machine and the communication costs between processors are not negligible, we also employ coarse grained parallelism to reduce several S-polynomials simultaneously and aim to solve relatively large problems efficiently. In the following section, we describe of our approach to parallel implementation of the Buchberger algorithm.

3 Parallelization

3.1 Notation and definition

The following notations and definitions are used in the following sections.

Definition 1 *(Power product and monomial)*
A power product is a product comprised of a finite number of nonzero variables, that is,

$$x_0 x_1 \ldots x_n \ (n \geq 0, \text{ each } x_i \text{ is variable}).$$

A monomial is the product of a coefficient (rational number) and a power product.

Definition 2 *(Polynomial)*
A polynomial is a sum of monomials.

Definition 3 *(Admissible order)*
An ordering, $\prec$, between power products is admissible when it satisfies the following properties. For all power products a, b, c,

$$1 \prec a$$
$$a \prec b \Rightarrow ac \prec bc.$$

We also express that the power product of monomial m is smaller than the power product of monomial n w.r.t. $\prec$ as $m \prec n$.

Definition 4 *(Leading monomial)*
For polynomial f, $Lm(f)$ represents the leading monomial of f, which is the maximum monomial w.r.t. $\prec$ contained in f. Furthermore, $Lc(f)$ represents the leading coefficient, the coefficient of $Lm(f)$.

Definition 5 *(Reducing)*
Let f and r be polynomials. If monomial m of f is a multiple of $Lm(r)$, then m is replaced by $m - \frac{m}{Lm(r)}r$

and f *is reduced to polynomial* $h = Lc(r)f - Lc(r)\frac{m}{Lm(r)}r$. *Notation* $f \Rightarrow_r h$ *represents that* f *is reduced to* h *by applying* r *to* f *once.*

Definition 6 *(Irreducible form)*
The irreducible form of polynomial f *w.r.t. polynomial set* R *is defined as the polynomial which cannot be further reduced by any polynomial in* R *after applying a polynomial in* R *to* f *a finite number of times. The irreducible form of* f *w.r.t.* R *is represented by* $f \downarrow_R$.

Definition 7 *(S-polynomial)*
The S-polynomial of polynomials f *and* g *is defined as*

$$\frac{lcm(Lm(f), Lm(g))}{Lm(f)}f - \frac{lcm(Lm(f), Lm(g))}{Lm(g)}g,$$

and is represented by $Spoly(f, g)$, *where* $lcm(Lm(f), Lm(g))$ *is the least common multiple of* $Lm(f)$ *and* $Lm(g)$.

Definition 8 *(Primitive part)*
$pp(f)$ *represents the primitive part of polynomial* f, *whose greatest common divisor of coefficients is 1.*

3.2 Approach to parallelizing the constraint solver

Figure 1 shows the sequential Buchberger algorithm. Since reduction is the most time consuming part of computing Gröbner bases as described in Hollman & Langemyr (1991), we try to parallelize this as shown by line (14) in Figure 1.

(1) **input** $F := F_{init}$ % F_{init} is a set of input polynomials.
(2) **input** $R := R_{init}$ % R_{init} is $\emptyset$ or an initial Gröbner base.
(3) $F := ReduceLm(F, R)$ % Reduce leading monomials of all polynomials in F by R.
(4) **while** $F \neq \emptyset$
(5) $f := Choose(F)$ % Choose the minimum polynomial from F'.
(6) $F := F \setminus \{f\}$
(7) $h := pp(f \downarrow_R)$
(8) **if** $h = 1$ **then**
(9) **return** $R = fail$
(10) **endif**
(11) $S := MakeSpolys(R, h)$ % Make S-polynomials.
(12) $F := F \cup S$ % Update a set of polynomials.
(13) $R := R \cup \{h\}$
(14) $F := ReduceLm(F, R)$
(15) **endwhile**
(16) $R := Interreduce(R)$ % Interreduce between elements of R.
(17) **return** R

Figure 1: Sequential Buchberger algorithm

The total number of S-polynomials generated during computation has a great influence on the ntotal efficiency of the Gröbner base computation. Thus, it is very important for efficient parallel implementation to decrease the number of S-polynomials. The number of S-polynomials is determined by the series of leading monomials of elements of the intermediate base R generated during computation. Thus, a new element of R should be selected so as to decrease the number of S-polynomials generated by the function $MakeSpolys(R, h)$. As described in Buchberger (1983), it is well known that a critical pair (f, g) is not necessary if the greatest common divisor of $Lm(f)$ and $Lm(g)$ is 1. To decrease the number of S-polynomials generated by $MakeSpolys(R, h)$, the minimum polynomial which has the minimum leading monomial should be selected, because a smaller monomial is apt to be the prime to other monomials rather than to a larger one. We should therefore choose the global minimum polynomial as a new element of R. We implement an algebraic constraint solver on a manager-worker model.

```
input F := F_init
input R := R_init
input n := n_init
  % n_init is the number of worker processes.
W := ∅
  % W is a set of polynomial sets.
for i = 0 to n − 1
  R_i := R
    % R_i is ith worker's intermediate base.
  W_i := ∅
    % W_i is ith worker's polynomial set.
  W := W ∪ {W_i}
endfor
F := ReduceLm(F, R)
m := Count(F)
  % Count polynomials in F.
for j = 0 to m − 1
  f := First(F)
    % f is the first polynomial in F.
  F := F \ {f}
  i := j mod n
  W_i := W_i ∪ {f}
endfor
while m ≠ 0
  F := ∅
  for i = 0 to n − 1
    if W_i ≠ ∅ then
      W := W \ {W_i}
      f_i := Choose(W_i)
      F := F ∪ {f_i}
      f'_i := f_i ↓R_i
      W_i := W_i ∪ {f'_i} \ {f_i}
      W := W ∪ {W_i}
      else f_i = 0
    endif
  endfor
  f := Choose(F)
  for i = 0 to n − 1
    if f_i = f then
      W := W \ {W_i}
      W_i := W_i \ {f'_i}
      W := W ∪ {W_i}
      h := pp(f'_i)
    endif
  endfor
  if h = 1 then
    return R = fail
  endif
  W := LoadBalance(R, W, h, n)
    % Generate S-polynomials
  R = R ∪ {h}
  m := 0
  for i = 0 to n − 1
    R_i := R_i ∪ {h}
    W_i := ReduceLm(W_i, R_i)
    m_i := Count(W_i)
    m := m + m_i
  endfor
endwhile
R := Interreduce(R)
return R
```

Figure 2: Manager-worker model

3.3 Manager-worker model

In the manager-worker model, reduction is parallelized by partitioning a polynomial set to be reduced. To decrease the communication cost between the manager and worker processes, all worker processes have the same intermediate bases on their own memories. They reduce polynomials in parallel and report the local minimum polynomials to the manager process. The manager process then chooses a global minimum polynomial from among these.

Figures 2 and 3 show our manager-worker model. The set of polynomials is partitioned and each worker process has a different subset W_i. The initial Gröbner base R_{init} is copied to all worker processes. New input polynomials are distributed to the worker processes by the manager process so that all worker processes have the same number of polynomials.

In order to choose a global minimum polynomial, it is necessary to reduce only the leading monomials of all polynomials by the intermediate base. To add the selected polynomial to the intermediate base, however, the selected polynomial must be reduced completely, while other polynomials need not be reduced completely. Thus, each worker process reduces polynomials by R_i according to the following reduction stages.

(1) A polynomial is not reduced.
(2) The leading monomial of a polynomial is reduced.
(3) A polynomial is completely reduced to an irreducible form.
(4) A polynomial is reduced to its primitive part.

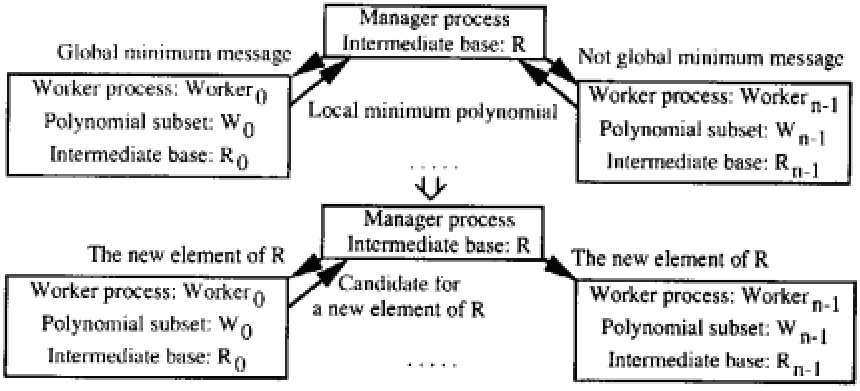


Figure 3: Architecture of the manager-worker model

All input polynomials and S-polynomials are input to the first stage. These polynomials are sent to the second stage after reducing the leading monomial completely by the function $ReduceLm(W_i, R_i)$. At the second stage, each worker process selects the local minimum polynomial, sends a copy of the local minimum polynomial to the manager process, and reduces the local minimum polynomial according to stages 3 and 4. After going through the reduction stages, the local minimum polynomial becomes the local candidate for the new element of R.

The manager process receives local minimum polynomials from all worker processes, selects the global minimum polynomial from among these, and sends a *global-minimum-message* or *not-global-minimum-message* to each worker process. When a worker process receives a global-minimum-message, it should send its local candidate to the manager process. The manager process receives the local new-element-candidate, adds it to R, and sends it to all worker processes. Each worker process receives the new element of R, adds it to R_i, then generates critical pairs and S-polynomials.

In the manager-worker model with synchronization, idling time exists in worker processes. In our implementation, to reduce such idling time, each worker process executes a subtask after computing its local candidate for a new element of R until it receives a global-minimum-message or the new element of R from the manager process. The main task is computation of the local candidate, and the subtask is the reduction of other polynomials, that is, the reduction of monomials other than the leading monomials. Since such subtasks are not always executed, these subtasks may cause *nondeterminacy* in our parallel algorithm, that is, the sequence of generating elements of R and the total number of generated S-polynomials are not the same for every execution. In our parallel algorithm, if several local candidates for the new element of R have the same leading monomials, then the manager process compares other monomials in those local candidates. Since the result of comparison by the manager process depends on whether such monomials of these local candidates have been reduced previously, the manager process does not always choose the same candidate at every execution, and nondeterminacy may occur. However, these subtasks may decrease the subsequent load.

3.4 Load balancing

For parallel processing, it is necessary to balance the load on each processor. Since each worker process reduces its own polynomials, we should balance loads according to the reduction cost. It is quite difficult, however, to predict the reduction cost for every S-polynomial beforehand. Thus, we balance loads so that each worker process has the same number of polynomials.

The most naive method for this load balancing is for the manager process to generate all S-polynomials and distribute them to worker processes. But this method increases the communication cost between the manager and worker processes, and results in a communication bottleneck.

Since each worker process has the same intermediate base R_i, it can generate the same critical pairs

```
B := ∅
    % B is a load balance information number set.
M := ∅
    % M is a set of number of polynomials.
for i = 0 to n − 1
  B := B ∪ {i}
  m_i := Count(W_i)
  M := M ∪ {m_i}
endfor
for i = 0 to n − 1
  B' := B
  B := ∅
  M' := M
  M := ∅
  b_i := n
  while b_i = n
    b := Minimum(B')
        % b is the minimum number in B'.
    B' := B' \ {b}
    m := Minimum(M')
    M' := M' \ {m}
    if m_i = m then
      b_i := b
      B := B ∪ B'
      M := M ∪ M'
    else
      B := B ∪ {b}
      M := M ∪ {m}
    endif
  endwhile
endfor
for i = 0 to n − 1
  W := W \ {W_i}
  C_i := CriticalPairs(R_i, h)
      % Make critical pairs.
  m := Count(C_i)
      % Count critical pairs in C_i.
  for c = 0 to m − 1
    (f, h) := First(C_i)
    C_i := C_i \ {(f, h)}
    j := c mod n
    if j = b_i then
      s := Spoly(f, h)
      W_i := W_i ∪ {s}
    endif
  endfor
  W := W ∪ {W_i}
endfor
return W
```

Figure 4: Load balancing

in the same order. Based on this fact, we have implemented a method where each worker process generates critical pairs in parallel, as follows. Figure 4 shows the load balancing executed by the function $LoadBalance(R, W, h, n)$ in Figure 2. Each worker process reports the number of polynomials it has to the manager process. The manager process decides on a load balance information number b_i for each worker process, and sends that number to them. Each worker process assigns a critical pair number, c, to each generated critical pair, where c is common to all worker processes. Then, each worker process generates S-polynomials from such critical pairs such that $c \bmod n = b_i$.

Figure 5 shows an example of load balancing. There are three worker processes, $Worker_0$, $Worker_1$, and $Worker_2$. The total number of worker processes, n, is 3. $Worker_0$ has 3 polynomials, $Worker_1$ has 2 polynomials, and $Worker_2$ has 4 polynomials. The processes report the number of polynomials they have to the manager process. The manager process decides on a load balance information number

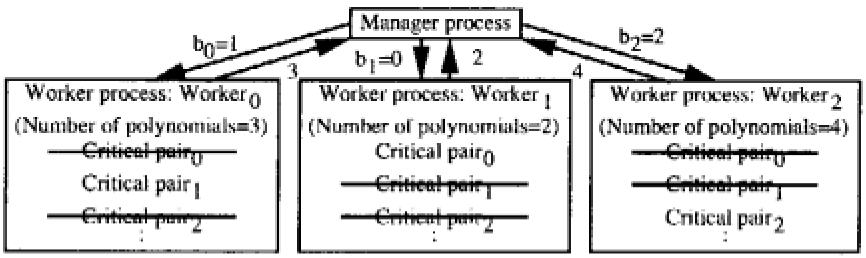


Figure 5: Example of load balancing

b_i. In this case, $b_0 = 1$, $b_1 = 0$, and $b_2 = 2$. The manager process then sends these load balance information numbers to all worker processes. Based on these load balance information numbers, $Worker_0$ generates S-polynomials from $Critical pair_c$ so that $c \bmod 3 = 1$, $Worker_1$ generates S-polynomials from $Critical\ pair_c$ so that $c \bmod 3 = 0$, and $Worker_2$ generates S-polynomials from $Critical\ pair_c$ so that $c \bmod 3 = 2$.

3.5 Various implementations

If monomial m of polynomial f is a multiple of leading monomial $Lm(g)$ of polynomial g in the intermediate base R, then f is reduced to $h = Lc(g)f - Lc(g)\frac{m}{Lm(g)}g$, as defined in Definition 5. The reduction may take a long time if the polynomials are complicated, that is, they have many monomials and their monomials have large coefficients. To improve the efficiency, we should avoid creating complicated polynomials by reduction.

In our parallel implementation, there are four issues that may have a great influence on complexity of the polynomials created by reduction: admissible ordering, order of using polynomials of the intermediate base for reduction, interreduction, and using redundant elements of the intermediate base to reduce polynomials.

As for the admissible ordering and the order of using polynomials, we have implementations that are regarded as the best for computing Gröbner bases efficiently: the total degree reverse lexicographic ordering and the order that *older* polynomials are used earlier than newer ones. The total degree reverse lexicographic ordering is said to give the best theoretical and practical complexity to the computation of Gröbner bases as described in Hollman (1992).

Definition 9 *(Total degree reverse lexicographic ordering)*
The total degree reverse lexicographic ordering is defined as:

$$x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} \prec x_1^{\beta_1} x_2^{\beta_2} \cdots x_n^{\beta_n} \quad (\textit{where } x_1 \prec x_2 \prec \cdots \prec x_n.)$$
$$\Leftrightarrow \sum \alpha_i < \sum \beta_i \text{ or } \sum \alpha_i = \sum \beta_i \ {}^{\exists} i \ \alpha_i > \beta_i \ \alpha_j = \beta_j \ (j < i).$$

The order of using older polynomials earlier is recommended by both Giovini *et al.* (1991) and Hollman & Langemyr (1991). Hollman (1992) also pointed out that the result of reduction of a polynomial tends to be more complicated than the polynomial before reduction. That is, there is a tendency for a newer polynomial to be more complicated than an older one, because a newer polynomial is reduced many more times than an older one. Therefore, to avoid creating complicated polynomials by reduction, older polynomials should be used before newer ones.

These implementations are also employed to our algebraic constraint solvers. As for interreduction and using redundant elements, however, it is necessary to examine their effect on the efficiency of the parallel computation of Gröbner bases.

3.5.1 Interreduction

Interreduction is the reduction of every polynomial in an intermediate base by other polynomials. Since newer polynomials have already been reduced by older polynomials, interreduction means that older polynomials are reduced by newer ones. Thus, since newer polynomials tend to be more complicated than older ones, interreduction may lead to complicated polynomials.

Giovini *et al.* (1991) described no interreduction as better. Though Czapor (1991) said that interreduction is necessary to efficiently compute Gröbner bases in lexicographic ordering, we use the total degree reverse lexicographic ordering instead of lexicographic ordering. In our parallel implementation, we compare a no interreduction strategy to a one-step interreduction strategy that replaces every polynomial r_i in an intermediate base with r_i', where $r_i \Rightarrow_h r_i'$ and h is a polynomial obtained as a new element of the intermediate base.

3.5.2 Using redundant polynomials for reduction

If a leading monomial of a polynomial in the intermediate base can be reduced by a newer polynomial, then the reducible polynomial is *redundant*, as described in Gebauer & Möller (1988). Is it better to use redundant polynomials for reduction or not? Since there is a tendency for older polynomials to be less complicated than newer ones, a redundant polynomial is expected to be less complicated than a newer polynomial which can reduce its leading monomial. Therefore, the result of reduction of a polynomial by the redundant polynomial is expected to be less complicated than that by the newer one. In this sense, a redundant polynomial may be effective for reduction. Giovini *et al.* (1991) described that redundant polynomials should be used. On the other hand, Gebauer & Möller (1988) said that the intermediate base should be kept as small as possible, that is, redundant polynomials should be cancelled.

In order to check whether using redundant polynomials is effective for reduction, we conducted several experiments.

3.6 Experiment

In this section, we describe the results of our experiments. There are two issues to be examined. One is interreduction, and the other is whether to use redundant polynomials for reduction. Thus, we made four parallel algebraic constraint solvers as below.

NI-UR: No Interreduction and Using Redundant polynomials.

NI-NR: No Interreduction and Not using Redundant polynomials.

OI-UR: One step Interreduction and Using Redundant polynomials.

OI-NR: One step Interreduction and Not using Redundant polynomials.

In each parallel algebraic constraint solver, the admissible ordering is the total degree reverse lexicographic ordering, and older polynomials are used earlier than newer ones for reduction.

The parallel algebraic constraint solvers are implemented on the parallel inference machine PIM/m developed at ICOT, and in the kernel language KL1 for the parallel inference machine. To measure the speed of the machine, we use a unit called *LIPS* (logical inference per second) described in Matsumoto (1990). Nakashima (1992) reports that each PIM/m processor has a clock of 15.4 MHz and a speed of 615 KLIPS at its highest peak performance, while the performance of SICStus Prolog systems on a SUN Sparc Station 10/30 is 1053 KLIPS. PIM/m is a distributed memory machine, and Matsumoto (1990) reports that it takes 30 logical inference steps in PIM/m to read one primitive data, such as a variable in a monomial, from the other processor.

To examine the performance of our parallel algebraic constraint solvers, we use the twelve benchmarks listed below. The details of these benchmarks are described in the Appendix.

(1) Katsura-4 (Boege *et al.*, 1986): (5 variables and 5 polynomials)

(2) Katsura-5 (Boege *et al.*, 1986): (6 variables and 6 polynomials)

(3) Katsura-6 (Katsura, 1986): (7 variables and 7 polynomials)

(4) Butcher (Boege *et al.*, 1986): (8 variables and 8 polynomials)

(5) Modified Hairer-2 (Boege *et al.*, 1986): (13 variables and 11 polynomials)

(6) Modified Hairer-3 (Boege *et al.*, 1986): (13 variables and 13 polynomials)

(7) Modified Gerdt (Boege *et al.*, 1986): (8 variables and 13 polynomials)

(8) Cyclic 4-roots: (4 variables and 4 polynomials)

(9) Cyclic 5-roots: (5 variables and 5 polynomials)

(10) Cyclic 6-roots: (6 variables and 6 polynomials)

(11) T-6 (Backelin & Fröberg, 1991): (7 variables and 6 polynomials)

(12) Ex-17 (Gebauer, 1985): (12 variables and 12 polynomials)

3.6.1 Results of experiments

Tables 1 to 6 and Figures 6 to 8 show the results of our experiments. We found that the computation time is not decreased by using two processors instead of one. Although this may seem strange, it is in fact quite reasonable. In our parallel implementations, each worker process is allocated to a different processor, and the manager process is also allocated to a different processor. With a single processor, however, the manager process and the worker process are, obviously, allocated to the same processor. The relationship between the number of worker processes n and the number of processors pn is given by

$$n = \begin{cases} 1 & (pn = 1) \\ pn - 1 & (pn \geq 2). \end{cases}$$

Thus, there is only a single worker process in both cases of using one and two processors, that is, the computation power is not increased by using two processors instead of one.

The effect of *nondeterminacy* in our experiments should be mentioned. As described in Section 3.3, the subtask causes nondeterminacy in our parallel algorithm. The effect of nondeterminacy is significant in the problems Modified Hairer-2 (Table 3 (a) and Figure 7 (a)) and Modified Hairer-3 (Table 3 (b) and Figure 7 (b)). In Modified Hairer-2, a remarkable speedup by using two processors is obtained by NI-UR. As described above, however, the computing power does not increase by using two processors; the obtained speedup is due to the nondeterminacy caused by the subtask. With a single processor, since the manager process and the worker processes are allocated to the same processor, the subtask is not executed because the worker process cannot work when the manager process works. With two processors, however, since the worker process can work when the manager process works, the subtask is executed. Thus, the sequence of generated polynomials of the intermediate set R is different from that with a single processor, and the total computational cost is decreased. The details of the computation are shown below.

(1) 1 processor:
Number of generated polynomials of $R = 100$
Number of redundant polynomials of $R = 62$
Number of generated S-polynomials = 284

(2) 2 processors:
Number of generated polynomials of $R = 82$
Number of redundant polynomials of $R = 44$
Number of generated S-polynomials = 257

In this case, the nondeterminacy has a good effect on the efficiency of the computation.

In Modified Hairer-3, however, the nondeterminacy has an adverse effect as shown in NI-NR and OI-UR. In these cases, the total computational cost increases drastically due to the nondeterminacy. Especially in OI-UR, the Gröbner base could not be computed in 24 hours with two processors, yet it was computed in 40 minutes with a single processor. The details of the computation by NI-NR are shown below.

(1) 1 processor:
Number of generated polynomials of $R = 109$
Number of redundant polynomials of $R = 69$
Number of generated S-polynomials = 399

(2) 2 processors:
Number of generated polynomials of $R = 109$
Number of redundant polynomials of $R = 69$
Number of generated S-polynomials = 406

The number of polynomials and S-polynomials generated during the computation are almost the same in both cases. Thus, the higher computational cost is due to the complexity of the polynomials.

Table 1: Timing and speedup for Katsura-4 and Katsura-5

(a) Timing and speedup for Katsura-4 (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	3.6	3.4	1.9	1.5	1.6	1.8	2.5	8.4	12.3
	Speedup	1.00	1.06	1.89	2.40	2.25	2.00	1.44	0.43	0.29
NI-NR	Timing	3.7	4.2	1.9	1.5	1.7	1.8	2.5	3.7	7.6
	Speedup	1.00	0.88	1.95	2.47	2.18	2.06	1.48	1.00	0.49
OI-UR	Timing	3.3	3.3	1.7	1.9	2.0	3.0	4.5	9.9	15.0
	Speedup	1.00	1.00	1.94	1.74	1.65	1.10	0.73	0.33	0.22
OI-NR	Timing	3.3	3.7	1.6	2.4	2.1	3.4	8.7	8.0	14.9
	Speedup	1.00	0.89	2.06	1.38	1.57	0.97	0.38	0.41	0.22

(b) Timing and speedup for Katsura-5 (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	38.3	37.3	15.8	9.4	10.6	13.9	16.1	19.5	29.4
	Speedup	1.00	1.03	2.42	4.07	3.61	2.76	2.38	1.96	1.30
NI-NR	Timing	37.0	36.6	15.0	9.1	11.3	14.4	17.2	23.9	27.7
	Speedup	1.00	1.01	2.47	4.07	3.27	2.57	2.15	1.55	1.34
OI-UR	Timing	32.2	34.6	13.7	9.1	11.5	17.9	23.9	48.8	72.5
	Speedup	1.00	0.93	2.35	3.54	2.80	1.80	1.35	0.66	0.44
OI-NR	Timing	32.7	33.9	15.1	9.2	17.3	22.3	23.1	50.1	70.0
	Speedup	1.00	0.96	2.17	3.55	1.89	1.47	1.42	0.65	0.47

Table 2: Timing and speedup for Katsura-6 and Butcher

(a) Timing and speedup for Katsura-6 (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	641	692	549	195	100	105	158	214	272
	Speedup	1.00	0.93	1.17	3.29	6.41	6.10	4.06	3.00	2.37
NI-NR	Timing	646	690	548	152	111	107	158	255	226
	Speedup	1.00	0.94	1.18	4.25	5.82	6.04	4.09	2.53	2.86
OI-UR	Timing	513	563	223	111	84	112	189	306	442
	Speedup	1.00	0.91	2.30	4.62	6.11	4.58	2.71	1.68	1.16
OI-NR	Timing	516	563	307	171	101	132	194	294	443
	Speedup	1.00	0.92	1.68	3.02	5.11	3.91	2.66	1.76	1.16

(b)Timing and speedup for Butcher (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	47.4	45.2	23.5	15.6	15.2	16.2	19.5	22.7	29.3
	Speedup	1.00	1.05	2.02	3.04	3.12	2.93	2.43	2.09	1.62
NI-NR	Timing	42.6	43.3	23.9	16.9	16.0	17.4	17.5	21.8	35.1
	Speedup	1.00	0.98	1.78	2.52	2.66	2.45	2.43	1.95	1.21
OI-UR	Timing	34.0	35.1	20.8	14.0	15.3	18.9	28.4	51.3	35.8
	Speedup	1.00	0.97	1.63	2.43	2.22	1.80	1.20	0.66	0.95
OI-NR	Timing	28.7	29.3	16.6	13.4	19.0	19.4	26.3	51.0	94.6
	Speedup	1.00	0.98	1.73	2.14	1.51	1.48	1.09	0.56	0.30

Table 3: Timing and speedup for Modified Hairer-2 and Modified Hairer-3

(a) Timing and speedup for Modified Hairer-2 (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	413	191	91	61	56	62	78	96	117
	Speedup	1.00	2.16	4.53	6.77	7.38	6.66	5.29	4.30	3.53
NI-NR	Timing	329	373	67	51	45	54	67	95	102
	Speedup	1.00	0.88	4.91	6.45	7.31	6.09	4.91	3.46	3.23
OI-UR	Timing	213	212	74	54	54	71	105	179	330
	Speedup	1.00	1.00	2.88	3.94	3.94	3.00	2.03	1.19	0.65
OI-NR	Timing	170	168	61	48	56	74	103	169	326
	Speedup	1.00	1.01	2.79	3.54	3.04	2.30	1.65	1.01	0.52

(b) Timing and speedup for Modified Hairer-3 (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	3572	5379	1086	665	449	391	591	566	602
	Speedup	1.00	0.66	3.29	5.37	7.96	9.14	6.04	6.31	5.93
NI-NR	Timing	6234	59734	3825	5897	1014	1052	973	800	868
	Speedup	1.00	0.10	1.63	1.06	6.15	5.93	6.41	7.71	7.18
OI-UR	Timing	2292	>1day	957	1215	705	>1day	>1day	>1day	>1day
	Speedup	1.00	-	2.39	1.89	3.25	-	-	-	-
OI-NR	Timing	2150	3002	756	877	2162	306	450	564	832
	Speedup	1.00	0.72	2.84	2.45	0.99	7.03	4.78	3.81	2.58

Table 4: Timing and speedup for Modified Gerdt and Cyclic 4-roots

(a) Timing and speedup for Modified Gerdt (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	665	679	296	181	129	140	164	259	330
	Speedup	1.00	0.98	2.25	3.67	5.16	4.75	4.05	2.57	2.02
NI-NR	Timing	976	1021	389	257	187	196	211	328	386
	Speedup	1.00	0.96	2.51	3.80	5.22	4.98	4.63	2.98	2.53
OI-UR	Timing	574	702	325	234	198	201	242	356	541
	Speedup	1.00	0.82	1.77	2.45	2.90	2.86	2.37	1.61	1.06
OI-NR	Timing	549	664	279	196	239	233	240	319	544
	Speedup	1.00	0.83	1.97	2.80	2.30	2.36	2.29	1.72	1.01

(b) Timing and speedup for Cyclic 4-roots (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	0.8	1.4	1.0	1.0	1.1	1.0	3.2	6.6	15.2
	Speedup	1.00	0.57	0.80	0.80	0.73	0.80	0.25	0.12	0.05
NI-NR	Timing	0.8	0.8	1.0	1.3	1.1	1.4	3.1	4.7	13.1
	Speedup	1.00	1.00	0.80	0.62	0.73	0.57	0.26	0.17	0.06
OI-UR	Timing	0.8	0.8	1.0	1.6	1.2	1.5	3.3	6.7	13.8
	Speedup	1.00	1.00	0.80	0.50	0.67	0.53	0.24	0.12	0.06
OI-NR	Timing	0.8	0.7	0.6	1.1	1.3	1.4	3.0	5.1	13.8
	Speedup	1.00	1.14	1.33	0.73	0.62	0.57	0.27	0.16	0.06

Table 5: Timing and speedup for Cyclic 5-roots and Cyclic 6-roots

(a) Timing and speedup for Cyclic 5-roots (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	10.6	10.7	7.3	5.1	6.5	9.6	10.4	13.1	39.8
	Speedup	1.00	0.99	1.45	2.08	1.63	1.10	1.02	0.81	0.27
NI-NR	Timing	10.2	10.7	6.0	5.4	6.2	8.5	8.6	10.5	20.9
	Speedup	1.00	0.95	1.70	1.89	1.65	1.20	1.19	0.97	0.49
OI-UR	Timing	10.3	9.8	6.5	5.9	7.0	10.2	14.1	27.0	52.6
	Speedup	1.00	1.05	1.58	1.75	1.47	1.01	0.73	0.38	0.20
OI-NR	Timing	10.4	9.7	6.0	5.6	7.1	11.3	14.5	28.2	52.9
	Speedup	1.00	1.07	1.73	1.86	1.46	0.92	0.72	0.37	0.20

(b) Timing and speedup for Cyclic 6-roots (in seconds)

Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	490	582	179	115	101	136	187	291	338
	Speedup	1.00	0.84	2.73	4.26	4.85	3.60	2.62	1.68	1.45
NI-NR	Timing	474	599	211	100	94	120	171	225	308
	Speedup	1.00	0.79	2.25	4.74	5.04	3.95	2.77	2.11	1.54
OI-UR	Timing	493	529	195	117	150	153	223	378	637
	Speedup	1.00	0.93	2.53	4.21	3.29	3.22	2.21	1.30	0.73
OI-NR	Timing	543	565	350	111	131	143	223	421	618
	Speedup	1.00	0.96	1.55	4.89	4.15	3.80	2.43	1.29	0.88

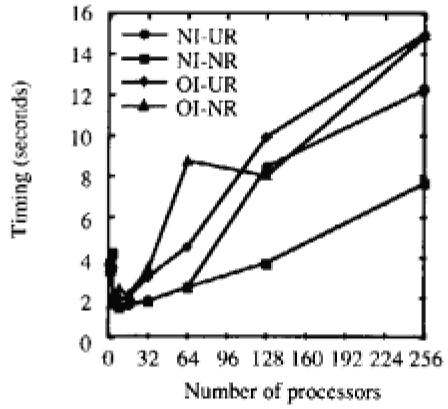
Table 6: Timing and speedup for T-6 and Ex-17

(a) Timing and speedup for T-6 (in minutes)

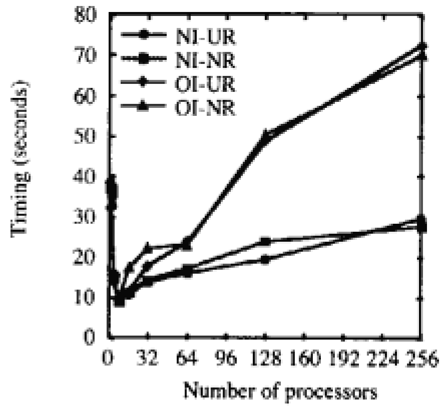
Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	239	245	100	54	41	30	28	33	34
	Speedup	1.00	0.98	2.39	4.43	5.83	7.97	8.54	7.24	7.03
NI-NR	Timing	243	245	102	53	37	28	28	33	34
	Speedup	1.00	0.99	2.38	4.58	6.57	8.68	8.68	7.36	7.15
OI-UR	Timing	216	242	95	50	33	28	31	43	69
	Speedup	1.00	0.89	2.27	4.32	6.55	7.71	6.97	5.02	3.13
OI-NR	Timing	220	245	91	50	34	29	29	44	70
	Speedup	1.00	0.90	2.42	4.40	6.47	7.59	7.59	5.00	3.14

(b) Timing and speedup for Ex-17 (in minutes)

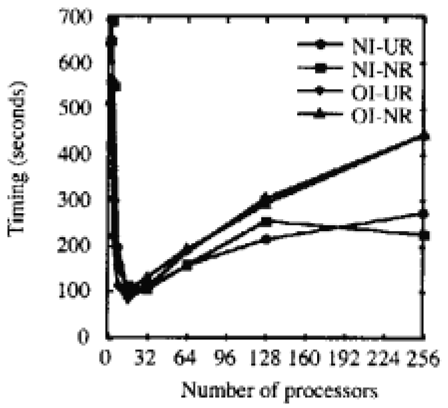
Solver		Number of processors								
		1	2	4	8	16	32	64	128	256
NI-UR	Timing	265	968	105	58	38	29	32	29	30
	Speedup	1.00	0.27	2.52	4.57	6.97	9.14	8.28	9.14	8.83
NI-NR	Timing	655	580	511	58	38	26	26	26	26
	Speedup	1.00	1.13	1.28	11.3	17.2	25.2	25.2	25.2	25.2
OI-UR	Timing	190	191	121	103	48	39	35	38	39
	Speedup	1.00	0.99	1.57	1.84	3.96	4.87	5.43	5.00	4.87
OI-NR	Timing	151	152	93	88	40	38	39	44	42
	Speedup	1.00	0.99	1.62	1.72	3.78	3.97	3.87	3.43	3.60



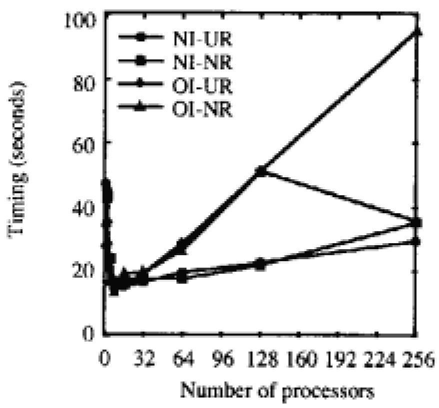
(a) Timing for Katsura-4



(b) Timing for Katsura-5



(c) Timing for Katsura-6



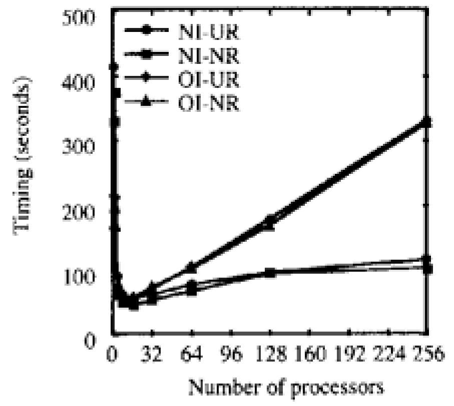
(d) Timing for Butcher

Figure 6: Timing data-1

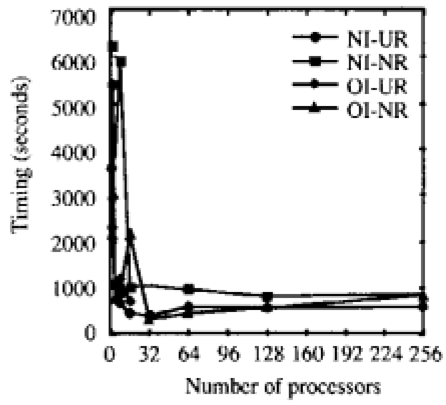
3.6.2 Influence of nondeterminacy caused by the subtask

Based on the results of experiments, NI-UR is employed as our parallel algebraic constraint solver. As described in the previous section, the subtask causes nondeterminacy and may adversely increase the total computational cost. However, the subtask is necessary to improve the efficiency of computing Gröbner bases by decreasing the idling time of processors. Therefore, a parallel algebraic constraint solver in which the nondeterminacy has little effect on the efficiency should be employed. NI-UR, which is a parallel algebraic constraint solver without interreduction and using redundant polynomials, showed a good absolute speed of computation and relatively stable speedup with multi-processors compared to others, though it does not compute all the Gröbner bases faster than others.

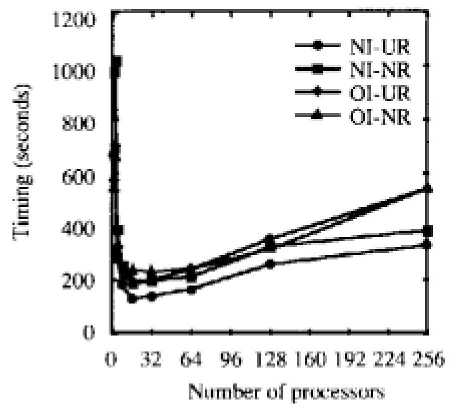
The reason why the nondeterminacy has little effect on the total computational cost in NI-UR is as follows. When we have two polynomials of the same leading monomial, we compare the rest of the polynomials and choose the smaller polynomial as a new element of the intermediate base. Thus, the subtask causes the nondeterminacy since the subtask is not always executed, as described in Section 3.3. Furthermore, when polynomials in the intermediate base are reduced or cancelled, the irreducible form of a polynomial is not unique but dependent on the reduction timing. Thus, in NI-UR, the influence of nondeterminacy is smaller than in any other solvers.



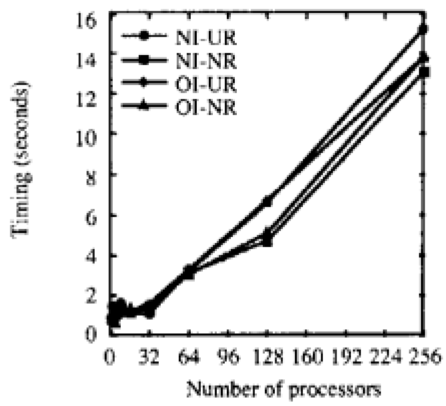
(a) Timing for Modified Hairer-2



(b) Timing for Modified Harier-3



(c) Timing for Modified Gerdt



(d) Timing for Cyclic 4-roots

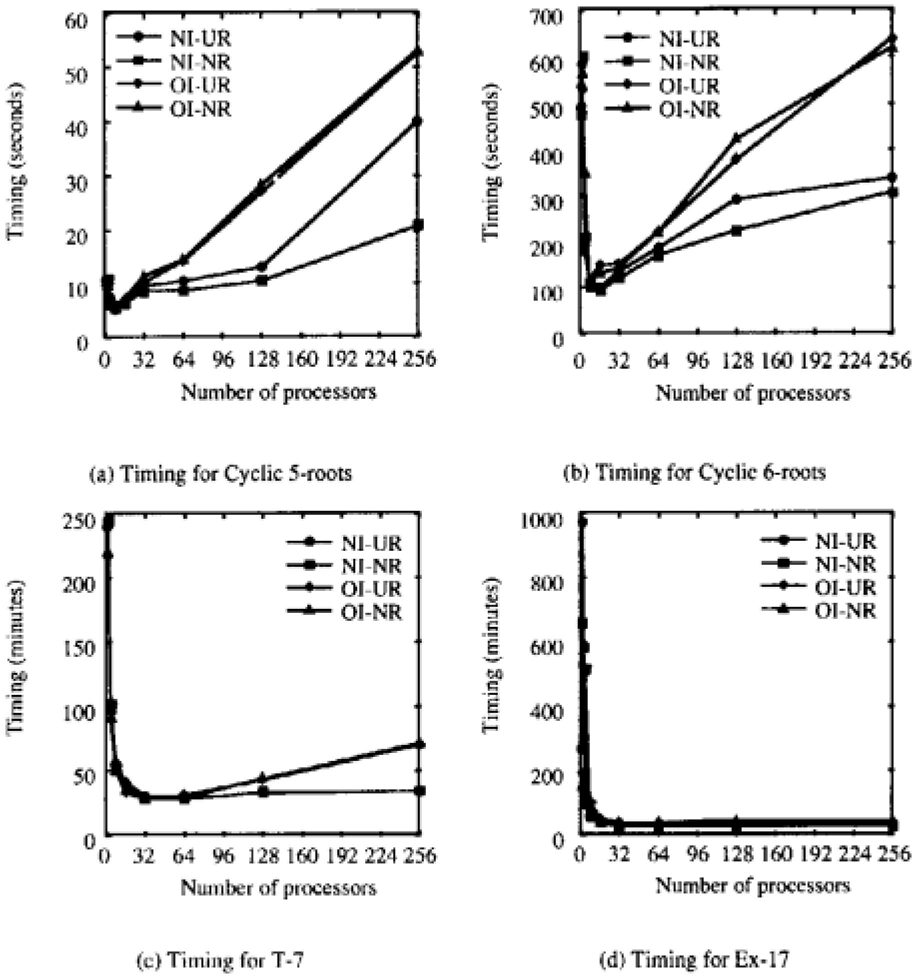
Figure 7: Timing data-2

4 Conclusion

We have developed several parallel algebraic constraint solvers in order to improve the absolute computation speed, because the efficiency of parallel algebraic constraint solvers are determined by the absolute computational speed and not by the speedup due to multi-processors. For example, for the Modified Gerdt problem in Table 4 (a) and Figure 7 (c), though the absolute computation speed of NI-UR is superior to that of NI-NR, the speedup of NI-NR is better than that of NI-UR because the computation speed of NI-NR is slow with a single processor. From this example, we find that the speedup must be evaluated along with the absolute computation speed.

In our research and development of parallel algebraic constraint solvers on the distributed memory machine PIM/m, there are three major issues: the communication costs between processors, the nondeterminacy caused by the subtask, and the selection of a minimum polynomial as a new element of the intermediate base.

To decrease the communication costs, we have implemented our algebraic constraint solvers so that each processor has the same intermediate base on its memory and so that each worker process generates S-polynomials for itself by exchanging an integer with the manager process. This implementation requires



(a) Timing for Cyclic 5-roots

(b) Timing for Cyclic 6-roots

(c) Timing for T-7

(d) Timing for Ex-17

Figure 8: Timing data-3

sufficient memory for each processor because the total memory necessary for the computation increases almost linearly with the increase of worker processes. However, we used this implementation since it is essential to improve the efficiency by decreasing the communication costs.

The nondeterminacy may have either a good or bad effect on the efficiency, and it is very difficult to predict whether nondeterminacy improves the efficiency. If the subtask is not executed, then nondeterminacy does not occur. The subtask is, however, necessary to improve the efficiency by decreasing the idling time of each processor. Thus, we have chosen NI-UR as our parallel algebraic constraint solver because the influence of nondeterminacy in NI-UR is less than that in any other.

Selecting a minimum polynomial as a new element of the intermediate base is known to decrease the total computational cost in the sequential Buchberger algorithm. To parallelize the Buchberger algorithm, it is also necessary to choose a minimum polynomial. We apply this selection strategy in our manager-worker model by synchronizing parallel worker processes. Such synchronization seems to decrease the speedup with multi-processors. By applying the selection strategy, however, it is guaranteed that the global minimum polynomial is selected as a new element, and the total computational cost does not increase with multi-processors. Furthermore, since each worker process reduces S-polynomials using the same algorithm, the total computation power increases even though synchronization may make some processors idle on every selection of a new element. As a result, the effect of the selection strategy on improving efficiency is greater than the decrease in efficiency caused by the defect in synchronization, and the speedup with multi-processors is also improved when compared with algebraic constraint solvers without the selection strategy.

Finally, we must note the influence of the bignum operation. The calculation of coefficients, that is, the bignum operation, occupies a large part of the total computation. Therefore, the performance of the bignum operation affects the strategy used to compute the Gröbner bases. If the performance of the bignum operation is improved in the future, then the strategy might be changed for the same benchmarks employed in our experiment.

5 Acknowledgments

We would like to thank Prof. Kazuhiro Fuchi and Dr. Ryuzo Hasegawa for their encouragements and and support of this research. We would also like to thank Mr. David J. Hawley, an important member of this research in the early stage, and Dr. Joachim Hollman for productive discussions and suggestions.

References

Aiba,A., Sakai,K., Sato,Y., Hawley,D. and Hasegawa,R.(1988). Constraint Logic Programming Language CAL. *Proc. International Conference on Fifth Generation Computer Systems*, 263-276.

Backelin,J. and Fröberg,R.(1991). How we proved that there are exactly 924 cyclic 7-roots. In: (Watt,S.M. editor) *Proc. ISSAC'91*, ACM, July, 103-111.

Boege,W., Gebauer,R. and Kredel,H.(1986). Some Examples for Solving Systems of Algebraic Equations by Calculating Groebner Bases *J. Symbolic Computation*, 83-98.

Buchberger,B.(1965). Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD Thesis. University of Innsbruck.

Buchberger,B.(1979). A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. *Proc. EUROSAM 1979*, Lecture Notes in Computer Science 72, Springer Verlag, Berlin-Heidelberg-New York, 3-21.

Buchberger,B.(1983). Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory. *Technical report*, RISC-LINZ.

Buchberger,B.(1987). The Parallelization of Critical-Pair/Completion Procedures on the L-Machine. *Proc. The Japanese Symposium on Functional Programming*, February, 54-61.

Chikayama,T.(1992). Operating System PIMOS and Kernel Language KL1. *Proc. International Conference on Fifth Generation Computer Systems*, 73-88.

Clarke,E.M., Long,D.E., Michaylov,S., Schwab,S.A., Vidal,J.P. and Kimura,S.(1990). Parallel Symbolic Computation Algorithms. *Technical Report CMU-CS-90-182*, School of Computer Science, Carnegie Mellon University, October.

Colmerauer,A.(1987). Opening the Prolog III Universe: A new generation of Prolog promises some powerful capabilities. *BYTE*, August, 177-182.

Czapor, S.R.(1991). A Heuristic Selection Strategy for Lexicographic Gröbner Bases? *Proc. ISSAC'91*, ACM, July, 39-48.

Fujita,H.(1989). Parallelizing the Knuth-Bendix Competition Algorithm. *Techinical Memorandum TM-731*, Institute for New Generation Computer Technology.

Gebauer,R.(1985). A collection of examples for Groebner calculations. IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

Gebauer,R. and Möller,H.M.(1988). On an Installation of Buchberger's Algorithm. *J.Symbolic Computation*, 6(2 and 3), 275-286.

Giovini,A., Mora,T., Niesei,G., Robbiano,L. and Traverso,C.(1991). "One sugar cube, please" or selection strategies in the Buchberger algorithm. In: (Watt,S.M., editor) *Proc. ISSAC'91*, ACM, July, 49-54.

Hawley,D.J.(1991). The Concurrent Constraint Language GDCC and Its Parallel Constraint Solver. *Technical Report TR-713*, Institute for New Generation Computer Technology.

Hollman,J. and Langemyr,L.(1991). Algorithms for Non-linear Algebraic Constraints. *Proc. WCLP 91*, Marseille, January.

Hollman,J.(1992). Theory and Applications of Gröbner bases. Ph.D.Thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden.

Jaffar,J. and Lassez,J.-L.(1987). Constraint Logic Programming. In *4th IEEE Symposium on Logic Programming*.

Katsura,S.(1986). Theory of spin glass by the method of the distribution function of an effective field. *Progress of Theoretical Physics, Supplement*, No.87, 139-154.

Matsumoto,Y.(1990). Text for KL1 class: Basic Data for Hackers. *Technical Memorandum TM-898*, Institute for New Generation Computer Technology.

Melenk,H. and Neun,W.(1988). Parallel Polynomial Operations in the Large Buchberger Algorithm. *Computer Algebra and Parallelism*, Workshop at the TIM3 Laboratory, University of Grenoble, France, Academic Press, London, June, 143-158.

Nakashima,H., Nakajima,K., Kondo,S., Takeda,Y., Inamura,Y. and Onishi,S. (1992). Arichitecture and Implementation of PIM/m. *Proc. International Conference on Fifth Generation Computer Systems*, 425-435.

Ponder,C.G.(1988). Evaluation of "Performance Enhancements" in Algebraic Manipulation Systems. PhD.Thesis, Computer Science Division, University of California, September.

Senechaud,P.(1989). Implementation of a parallel algorithm to compute a Gröbner basis on boolean polynomials. *Computer Algebra and Parallelism*, Academic Press, 159-166.

Siegl,K.(1990). Gröbner basis computation in STRAND: A case study for concurrent symbolic computation in logic programming languages. DIPLOMA Thesis, RISC-LINZ, November.

Taki,K.(1992). Parallel Inference Machine PIM. *Proc. International Conference on Fifth Generation Computer Systems*, 50-72.

Terasaki,S., Hawley,D.J., Sawada,H., Satoh,K.,Menju,S., Kawagishi,T., Iwayama,N. and Aiba,A.(1992). Parallel Constraint Logic Programming Language GDCC and its Parallel Constraint Solvers. *Proc. International Conference on Fifth Generation Computer Systems*, 330-346.

Ueda,K. and Chikayama,T.(1990). Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, Vol.33, No.6, 494-500.

Vidal,J.-P.(1990). The computation of Gröbner bases on a shared memory multiprocessor. *Proc. DISCO'90*.

Appendix Benchmarks

The following benchmarks are used in this paper.

(1) Katsura-4 (Boege *et al.*, 1986): $(U_0 \prec U_1 \prec U_2 \prec U_3 \prec U_4)$
Number of input polynomials = 5
Number of polynomials in the final Gröbner base = 13
$(U_0^2 - U_0 + 2U_1^2 + 2U_2^2 + 2U_3^2 + 2U_4^2)$
$(2U_0U_1 + 2U_1U_2 + 2U_2U_3 + 2U_3U_4 - U_1)$
$(2U_0U_2 + 2U_1^2 + 2U_1U_3 + 2U_2U_4 - U_2)$
$(2U_0U_3 + 2U_1U_2 + 2U_1U_4 - U_3)$
$(U_0 + 2U_1 + 2U_2 + 2U_3 + 2U_4 - 1)$

(2) Katsura-5 (Boege *et al.*, 1986): $(U_0 \prec U_1 \prec U_2 \prec U_3 \prec U_4 \prec U_5)$
Number of input polynomials = 6
Number of polynomials in the final Gröbner base = 23
$(U_0^2 - U_0 + 2U_1^2 + 2U_2^2 + 2U_3^2 + 2U_4^2 + 2U_5^2)$
$(2U_0U_1 + 2U_1U_2 + 2U_2U_3 + 2U_3U_4 + 2U_4U_5 - U_1)$
$(2U_0U_2 + U_1^2 + 2U_1U_3 + 2U_2U_4 + 2U_3U_5 - U_2)$
$(2U_0U_3 + 2U_1U_2 + 2U_1U_4 + 2U_2U_5 - U_3)$
$(2U_0U_4 + 2U_1U_3 + 2U_1U_5 + U_2^2 - U_4)$
$(U_0 + 2U_1 + 2U_2 + 2U_3 + 2U_4 + 2U_5 - 1)$

(3) Katsura-6 (Katsura, 1986): $(U_0 \prec U_1 \prec U_2 \prec U_3 \prec U_4 \prec U_5 \prec U_6)$
Number of input polynomials = 7
Number of polynomials in the final Gröbner base = 41
$(U_0 + 2U_5 + 2U_4 + 2U_3 + 2U_2 + 2U_1 + 2U_6 - 1)$
$(2U_5U_0 - U_5 + 2U_1U_4 + 2U_2U_3 + 2U_6U_1)$
$(2U_4U_0 + 2U_1U_5 - U_4 + 2U_1U_3 + U_2^2 + 2U_6U_2)$
$(2U_3U_0 + 2U_2U_5 + 2U_1U_4 + (2U_6 - 1)U_3 + 2U_1U_2)$
$(2U_2U_0 + 2U_3U_5 + (2U_2 + 2U_6)U_4 + 2U_1U_3 - U_2 + U_1^2)$
$(2U_1U_0 + (2U_4 + 2U_6)U_5 + 2U_3U_4 + 2U_2U_3 + 2U_1U_2 - U_1)$
$(U_0^2 - U_0 + 2U_5^2 + 2U_4^2 + 2U_3^2 + 2U_2^2 + 2U_1^2 + 2U_6^2)$

(4) Butcher (Boege *et al.*, 1986): $(A \prec A_{32} \prec B \prec B_1 \prec B_2 \prec B_3 \prec C_2 \prec C_3)$ Number of input polynomials = 8
Number of polynomials in the final Gröbner base = 25
$(B_1 + B_2 + B_3 - (A + B))$
$(B_2C_2 + B_3C_3 - (\frac{1}{2} + \frac{1}{2}B + B^2 - AB))$
$(B_2C_2^2 + B_3C_3^2 - (A(\frac{1}{3} + B^2) - \frac{4}{3}B - B^2 - B^3))$
$(B_3A_{32}C_2 - (A(\frac{1}{6} + \frac{1}{2}B + B^2) - \frac{2}{3}B - B^2 - B^3))$
$(B_2C_2^3 + B_3C_3^3 - (\frac{1}{4} + \frac{1}{4}B + \frac{5}{2}B^2 + \frac{3}{2}B^3 + B^4 - A(B + B^3)))$
$(B_3C_3A_{32}C_2 - (\frac{1}{8} + \frac{3}{8}B + \frac{7}{4}B^2 + \frac{3}{2}B^3 + B^4 - A(\frac{1}{2}B + \frac{1}{2}B^2 + B^3)))$
$(B_3A_{32}C_2^2 - (\frac{1}{12} + \frac{1}{12}B + \frac{7}{6}B^2 + \frac{3}{2}B^3 + B^4 - A(\frac{2}{3}B + B^2 + B^3)))$
$(\frac{1}{24} + \frac{7}{24}B + \frac{13}{12}B^2 + \frac{3}{2}B^3 + B^4 - A(\frac{1}{3}B + B^2 + B^3))$

(5) Modified Hairer-2 (Boege *et al.*, 1986): $(A_{21} \prec A_{31} \prec A_{32} \prec A_{41} \prec A_{42} \prec A_{43} \prec B_1 \prec B_2 \prec B_3 \prec B_4 \prec C_2 \prec C_3 \prec C_4)$
Number of input polynomials = 11
Number of polynomials in the final Gröbner base = 38
$(B_1 + B_2 + B_3 + B_4 - 1)$
$(B_2C_2 + B_3C_3 + B_4C_4 - \frac{1}{2})$
$(B_2C_2^2 + B_3C_3^2 + B_4C_4^2 - \frac{1}{3})$
$(B_3A_{32}C_2 + B_4A_{42}C_2 + B_4A_{43}C_3 - \frac{1}{6})$
$(B_2C_2^3 + B_3C_3^3 + B_4C_4^3 - \frac{1}{4})$
$(B_3C_3A_{32}C_2 + B_4C_4A_{42}C_2 + B_4C_4A_{43}C_3 - \frac{1}{8})$

$(B_3A_{32}C_2^2 + B_4A_{42}C_2^2 + B_4A_{43}C_3^2 - \frac{1}{12})$
$(B_4A_{43}A_{32}C_2 - \frac{1}{24})$
$(C_2 - A_{21})$
$(C_3 - A_{31} - A_{32})$
$(C_4 - A_{41} - A_{42} - A_{43})$
$(C_4 - 1)$

(6) Modified Hairer-3 (Boege *et al.*, 1986): $(A_{32} \prec A_{42} \prec A_{43} \prec A_{52} \prec A_{53} \prec A_{54} \prec B_3 \prec B_4 \prec B_5 \prec C_2 \prec C_3 \prec C_4 \prec C_5)$
Number of input polynomials = 13
Number of polynomials in the final Gröbner base = 46
$(15B_3C_2^2C_3 + 15B_4C_2^2C_4 + 15B_4C_3^2C_4 + 15B_5C_2^2C_2 + 15B_5C_3^2C_5 + 15B_5C_4^2C_5 - 1)$
$(30B_4C_4C_2 + 60B_5C_5C_2 + 30B_5C_5C_3)$
$(20B_3C_2^2 + 20B_4C_2^2 + 40B_4C_2C_3 + 20B_4C_3^2 + 20B_5C_2^2 + 20B_5C_3^2 + 20B_5C_4^2 + 40B_5C_2C_3 + 40B_5C_2C_4 + 40B_5C_3C_4 - 1)$
$(20B_3C_2^3 + 20B_4C_2^3 + 20B_4C_3^3 + 20B_5C_2^3 + 20B_5C_3^3 + 20B_5C_4^3)$
$(40B_4C_3C_2 + 40B_5C_3C_2 + 40B_5C_4C_2 + 40B_5C_4C_3 - 1)$
$(60B_4C_2^2 + 120B_5C_2^2 + 60B_5C_3^2 - 1)$
$(120B_5C_2 - 1)$

(7) Modified Gerdt (Boege *et al.*, 1986): $(L_0 \prec L_1 \prec L_2 \prec L_3 \prec L_4 \prec L_5 \prec L_6 \prec L_7)$
Number of input polynomials = 13
Number of polynomials in the final Gröbner base = 69
$(L_1(2L_4 - L_5 + 2L_6))$
$((2L_1^2 - 7L_4)(-10L_0 + 5L_2 - L_3))$
$((2L_1^2 - 7L_4)(3L_4 - L_5 + L_6))$
$((-2 * L_1^2 + L_1L_2 + 2L_1L_3 - L_2^2 - 7L_5 + 21L_6)(-3L_1 + 2L_2) + 3(343 - 14L_1L_4 + 3L_1^3))$
$(7(-2L_1^2 + L_1L_2 + 2L_1L_3 - L_2^2 - 7L_5 + 21L_6)(2L_4 - 2L_5) + (49L_7 - 14L_1L_4 + 3L_1^3)(-45L_1 + 15L_2 - 3L_3))$
$(14(-2L_1^2 + L_1L_2 + 2L_1L_3 - L_2^2 - 7L_5 + 21L_6)L_7 + (49L_7 - 14L_1L_4 + 3L_1^3)(12L_4 - 3L_5 + 2L_6))$
$(L_1(5L_1 - 3L_2 + L_3)(2L_2 - L_1) + 7L_1(2L_6 - 4L_4))$
$(L_1(5L_1 - 3L_2 + L_3)L_3 + 7L_1(2L_6 - 4L_4))$
$(L_1(5L_1 - 3L_2 + L_3)(-2L_4 - 2L_5) + L_1(2L_6 - 4L_4)(2L_2 - 8L_1) + 42L_1L_7)$
$(L_1(5L_1 - 3L_2 + L_3)(8L_5 + 18L_6) + L_1(2L_6 - 4L_4)(33L_1 - 17L_2 + 5L_3) - 252L_1L_7)$
$(15L_7L_1(5L_1 - 3L_2 + L_3) + L_1(2L_6 - 4L_4)(5L_4 - 2L_5) + L_1L_7(-60L_1 + 15L_2 - 3L_3))$
$(-6L_1(5L_1 - 3L_2 + L_3)L_7 + L_1(L_6 - 2L_4)(-2L_4 + L_5 - 2L_6) + L_1L_7(24L_1 - 6L_2))$
$(3L_1(2L_6 - 4L_4)L_7 + L_1L_7(20L_4 - 4L_5 + 2L_6))$

(8) Cyclic 4-roots: $(X_1 \prec X_2 \prec X_3 \prec X_4)$
Number of input polynomials = 4
Number of polynomials in the final Gröbner base = 7
$(X_1 + X_2 + X_3 + X_4)$
$(X_1X_2 + X_2X_3 + X_3X_4 + X_4X_1)$
$(X_1X_2X_3 + X_2X_3X_4 + X_3X_4X_1 + X_4X_1X_2)$
$(X_1X_2X_3X_4 - 1)$

(9) Cyclic 5-roots: $(X_1 \prec X_2 \prec X_3 \prec X_4 \prec X_5)$
Number of input polynomials = 5
Number of polynomials in the final Gröbner base = 20
$(X_1 + X_2 + X_3 + X_4 + X_5)$
$(X_1X_2 + X_2X_3 + X_3X_4 + X_4X_5 + X_5X_1)$
$(X_1X_2X_3 + X_2X_3X_4 + X_3X_4X_5 + X_4X_5X_1 + X_5X_1X_2)$
$(X_1X_2X_3X_4 + X_2X_3X_4X_5 + X_3X_4X_5X_1 + X_4X_5X_1X_2 + X_5X_1X_2X_3)$
$(X_1X_2X_3X_4X_5 - 1)$

(10) Cyclic 6-roots: $(X_1 \prec X_2 \prec X_3 \prec X_4 \prec X_5 \prec X_6)$
Number of input polynomials = 6

Number of polynomials in the final Gröbner base = 45
$(X_1 + X_2 + X_3 + X_4 + X_5 + X_6)$
$(X_1X_2 + X_2X_3 + X_3X_4 + X_4X_5 + X_5X_6 + X_6X_1)$
$(X_1X_2X_3 + X_2X_3X_4 + X_3X_4X_5 + X_4X_5X_6 + X_5X_6X_1 + X_6X_1X_2)$
$(X_1X_2X_3X_4 + X_2X_3X_4X_5 + X_3X_4X_5X_6 + X_4X_5X_6X_1 + X_5X_6X_1X_2 + X_6X_1X_2X_3)$
$(X_1X_2X_3X_4X_5 + X_2X_3X_4X_5X_6 + X_3X_4X_5X_6X_1 + X_4X_5X_6X_1X_2 + X_5X_6X_1X_2X_3$
$+X_6X_1X_2X_3X_4)$
$(X_1X_2X_3X_4X_5X_6 - 1)$

(11) T-6 (Backelin & Fröberg, 1991): $(X_1 \prec X_2 \prec X_3 \prec X_4 \prec X_5 \prec X_6 \prec X_7)$

Number of input polynomials = 6
Number of polynomials in the final Gröbner base = 120
$(X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7)$
$(X_1X_2 + X_2X_3 + X_3X_4 + X_4X_5 + X_5X_6 + X_6X_7 + X_7X_1)$
$(X_1X_2X_3 + X_2X_3X_4 + X_3X_4X_5 + X_4X_5X_6 + X_5X_6X_7 + X_6X_7X_1 + X_7X_1X_2)$
$(X_1X_2X_3X_4 + X_2X_3X_4X_5 + X_3X_4X_5X_6 + X_4X_5X_6X_7 + X_5X_6X_7X_1 + X_6X_7X_1X_2$
$+X_7X_1X_2X_3)$
$(X_1X_2X_3X_4X_5 + X_2X_3X_4X_5X_6 + X_3X_4X_5X_6X_7 + X_4X_5X_6X_7X_1 + X_5X_6X_7X_1X_2$
$+X_6X_7X_1X_2X_3 + X_7X_1X_2X_3X_4)$
$(X_1X_2X_3X_4X_5X_6+X_2X_3X_4X_5X_6X_7+X_3X_4X_5X_6X_7X_1+X_4X_5X_6X_7X_1X_2 +X_5X_6X_7X_1X_2X_3+$
$X_6X_7X_1X_2X_3X_4 + X_7X_1X_2X_3X_4X_5)$

(12) Ex-17 (Gebauer, 1985): $(A \prec B \prec C \prec D \prec E \prec F \prec K \prec L \prec M \prec N \prec O \prec P)$
Number of input polynomials = 12
Number of polynomials in the final Gröbner base = 50
$(F - 21)$
$(29F - 21P)$
$(529A - K)$
$(735E - 361FP - 399O + 529F^2)$
$(820D - 272PE - 306FO - 340N + 1058FE)$
$(893C - 195PD - 225EO - 255FN - 285M + 1058FD + 529E^2)$
$(954B - 130PC - 156OD - 182EN - 208FM - 234L + 1058FC + 1058ED)$
$(1003A - 77PB - 99OC - 121DN - 143EM - 165FL - 187K + 1058FB + 1058EC + 529D^2)$
$(1058FA + 1058EB + 1058DC - 36PA - 54OB - 72NC - 90DM - 108EL - 126FK)$
$(1058AE + 1058DB + 529C^2 - 21AO - 35NB - 49CM - 63DL - 77KE)$
$(1058AD + 1058CB - 10AN - 20MB - 30CL - 40KD)$
$(1058AC + 529B^2 - 3AM - 9BL - 15KC)$